Table of Contents

- Introduction 1.1
- Glossary 1.2
- Safety tips 1.3
- Assembly 1.4
  - Clover 4.2 assembly 1.4.1
  - Clover 4.2 WorldSkills 1.4.2
  - Clover 4 assembly 1.4.3
  - Clover 3 assembly 1.4.4
  - Clover 2 assembly 1.4.5
- Configuration 1.5
  - Sensor calibration 1.5.1
  - RC setup 1.5.2
    - Using FS-A8S 1.5.2.1
  - Flight modes 1.5.3
  - Power setup 1.5.4
  - Failsafe configuration 1.5.5
- Manual flight 1.6
  - Basics 1.6.1
  - Exercises 1.6.2
- Working with Raspberry Pi 1.7
  - RPi Image 1.7.1
  - Wi-Fi connection 1.7.2
  - Connection to the Pixracer 1.7.3
  - Using QGroundControl over Wi-Fi 1.7.4
  - Remote shell 1.7.5
  - Command line interface 1.7.6
  - Automated self-checks 1.7.7
  - Viewing images from cameras 1.7.8
- Programming 1.8
  - Camera setup 1.8.1
  - Fiducial markers (ArUco) 1.8.2
    - Marker detection 1.8.2.1
    - Map-based navigation 1.8.2.2
  - Optical Flow 1.8.3
  - Autonomous flight (OFFBOARD) 1.8.4

- Coordinate systems (frames) 1.8.5
- Code examples 1.8.6
- Interfacing with a laser rangefinder 1.8.7
- LED strip 1.8.8
- Working with GPIO 1.8.9
- Interfacing with a sonar 1.8.10
- Computer vision basics 1.8.11
- Using rviz and rqt 1.8.12
- Software autorun 1.8.13
- Using JavaScript 1.8.14
- Blocks programming 1.8.15
- Simulation 1.8.16
  - Native setup 1.8.16.1
  - VM setup 1.8.16.2
  - Usage 1.8.16.3
  - Setup on M1 computers 1.8.16.4
- ROS 1.8.17
- MAVROS 1.8.18

Supplementary materials 1.9

- COEX Pix 1.9.1
- COEX PDB 1.9.2
- COEX GPS 1.9.3
- Using SSH keys 1.9.4
- Guide on autonomous flight 1.9.5
- Hostname 1.9.6
- PX4 Simulation 1.9.7
- Navigation using vertical ArUco-markers 1.9.8
- PID Setup 1.9.9
- Model files for parts 1.9.10
- ROS Melodic installation 1.9.11
- Camera calibration 1.9.12
- VPN ZeroTier Connection 1.9.13
- Quadcopter control with 4G communication 1.9.14
- Clover and Jetson Nano 1.9.15
- Remote control app 1.9.16
- Wi-Fi Configuration 1.9.17
- UART settings 1.9.18
- PX4 Parameters 1.9.19

- PX4 Logs and Topics 1.9.20
- PX4 Firmware 1.9.21
- MAVLink 1.9.22
- Multimeter usage 1.9.23
- RC Troubleshooting 1.9.24
- Flashing ESCs 1.9.25
- Interfacing with Arduino 1.9.26
- Connecting GPS 1.9.27
- Working with IR sensors 1.9.28
- FPV Setup 1.9.29
- FPV Setup (Clover 3) 1.9.30
- Magnetic grip 1.9.31
- Mechanical grip 1.9.32
- Trainer mode 1.9.33
- Tinning 1.9.34
- Types of power connectors 1.9.35
- Connecting 4 in 1 ESCs 1.9.36
- Soldering safety 1.9.37
- LED strip (legacy) 1.9.38
- Contribution Guidelines 1.9.39
- COEX packages repository 1.9.40
- Migration to v0.20 1.9.41
- Migration to v0.22 1.9.42

Events 1.10

- CopterHack-2023 1.10.1
- CopterHack-2022 1.10.2
- CopterHack-2021 1.10.3
- CopterHack-2019 1.10.4
- CopterHack-2018 1.10.5
- CopterHack-2017 1.10.6
- Video contest 1.10.7
- Educational contests 1.10.8

Clover-based projects 1.11

- Clover Cloud Platform 1.11.1
- Autonomous Racing Drone 1.11.2
- Motion Capture System 1.11.3
- Swarm in Blocks 2 1.11.4
- Advanced Clover 2 1.11.5

Network of charging stations	1.11.6
Swarm-in-blocks	1.11.7
Obstacle avoidance using artificial potential fields method	1.11.8
The Clover Rescue Project	1.11.9
CopterCat CM4	1.11.10
Autonomous valet parking drone assistance	1.11.11
Autonomous Multirotor Landing System (AMLS)	1.11.12
Drone show	1.11.13
Innopolis Open 2020 (L22_ÆRO)	1.11.14
Copter spheric guard	1.11.15
Face recognition system	1.11.16
Android RC app	1.11.17
3D-scanning drone	1.11.18
Human pose estimation drone control	1.11.19
Robocross-2019	1.11.20
Camera calibration (legacy)	1.11.21
Recognition of crop types in agriculture	1.11.22
Drones to fight Coronavirus	1.11.23
D-drone Copter Hack 2021 by AT Makers	1.11.24
3D-printed Generative Design Frame	1.11.25
Retail Drone	1.11.26
The Indoor Mapping Drone	1.11.27
Seeding Drone	1.11.28
Blue Jay Eindhoven	1.11.29

COEX Clover



Clover is an educational kit of a programmable quadcopter that consists of popular open source components, and a set of necessary documentation and libraries for working with it.

The kit includes a COEX Pix flight controller with the PX4 flight stack, a Raspberry Pi 4 as a controlling onboard computer, and a camera module for performing flights with the use of computer vision, as well as a set of various sensors and other peripherals.

The Clover platform contains a pre-configured image for Raspberry Pi with the full set of required software for working with peripheral devices and programming autonomous flights. The source code of the platform and of the documentation is open and available on GitHub.

If you have studied the documentation but have not found an answer to your question, join our support chat and our specialists will be happy to answer you: @COEXHelpdesk.

We also have a chat for programmers coding for PX4, autonomous navigation indoors, and drone swarms: @DroneCode.

You can download PDF-version of this documentation.

Glossary

Drone

An unmanned aircraft. Typical examples are: quadrotors, hexacopters, model airplanes, fixed wings, VTOLs, model helicopters.

Quadcopter

An unmanned aerial vehicle with 4 propellers and an electronic stabilization system.

Multicopter

An unmanned aerial vehicle with an electronic stabilization system and the number of propellers equal to 3 (tricopter), 4 (quadcopter), 6 (hexacopter), 8 (octocopter), or more.

Flight controller / autopilot

1. A specialized circuit-board designed for controlling a multicopter, a plane or another vehicle. Examples: Pixhawk, ArduPilot, Naze32, CC3D.

2. Software for the multicopter control circuit-board. Examples: PX4, APM, CleanFlight, BetaFlight.

Firmware

Software primarily for embedded systems, for example, a flight controller or an ESC.

Motor

An electric motor that rotates propellers of the multicopter. Brushless motors are commonly used. These motors require an ESC.

ESC / motor controller

An Electronic Speed Controller. A specialized circuit-board that controls the speed of the brushless motor. It is controlled by a flight controller using pulse width modulation (PWM).

ESC has the firmware that determines the characteristics of its operation.

Battery

A rechargeable power source for the drone. Quadrotors typically use LiPo (lithium-ion polymer) batteries.

Battery cell

Single element of the battery pack. Typical drone batteries contain several (2 to 6) cells connected in series. Maximum LiPo cell voltage is 4.2 v; battery voltage is a sum of each cell's voltage (if they are connected in series). The number of cells connected in series is marked by the letter *S*, as in *2S* (two cells in series), *3S*, *4S*.

Clover kits typically use *3S* batteries.

Remote control / radio control equipment

A radio-operated quadcopter remote control. Operation of the remote control requires connecting a receiver to the flight controller.

Clover may also be controlled from a smartphone.

Telemetry

1. Transmitting the data about the state of a quadcopter or another aircraft over a distance.

2. The data about the aircraft state (height, orientation, global coordinates, etc.).

3. A system for transmitting the data about the aircraft state or commands to it over the air. Examples: radio modems (RFD900, 3DR Radio Modem), Wi-Fi modules (ESP-07). Raspberry Pi may also be used in Clover as a telemetry module: the use of QGroundControl via Wi-Fi.

Arming

Armed is the state of copter readiness for the flight. When the gas stick is lifted, or when an external command with the target point is sent, the copter will fly.
Usually, a copter starts rotating its propellers when it is switched to the "armed" state, even if the gas stick is down.

The opposite state is Disarmed.

PX4

A popular open source flight controller software that works with the Pixhawk series of flight controllers, Pixracer, and others. PX4 is recommended to be used with Clover.

Raspberry Pi

A popular single-board computer that is used in the Clover kit.

SD card image

A complete digital copy of SD card contents stored in a single file. This file may be written to an SD card using special software like Etcher. A Raspberry Pi's SD card is the only long-term memory of the single-board computer.

The Clover kit includes a recommended SD card image

APM / ArduPilot

An open source flight controller originally created for the Arduino boards. It was later ported to Pixhawk, Pixracer and other boards.

MAVLink

A communication protocol for drones, ground stations and other devices over radio channels. This protocol is widely used for telemetry.

ROS

A popular framework for writing complex robotics applications.

MAVROS

A library that is a link between the aircraft operating using the MAVLink protocol, and ROS.

UART

A serial asynchronous data transfer interface used in many devices. For example, GPS antennas, Wi-Fi routers, or Pixhawk.

IMU

Inertial measurement unit. A set of inertial sensors (a gyroscope and an accelerometer; a magnetometer is typically added as well) that allow the drone to compute its orientation (and, to a lesser extent, position) in space.

Estimation

A process of current state (position, rotation, velocity, angular rates, etc.) estimation performed by the flight controller software. A Kalman Filter is typically used for sensor fusion; other filters are typically applied to raw sensor data.

PX4 has two estimation modules: LPE and ECL EKF (EKF2).

APM utilizes its EKF2 subsystem.

Safety tips

Soldering



Soldering and tinning should be performed in specially prepared rooms. A ventilation system and a fume extractor are mandatory.

Before you start:

1. Prepare your workplace. Nothing should interrupt the process. Your workbench should be well lit.
2. Check the integrity of wiring and plugs of all electrical appliances.
3. Place your soldering iron near the fume extractor. Use a soldering iron stand when the soldering iron is not in use.
4. Wear safety goggles and gloves.

During soldering:

1. Use the soldering iron grip to hold the soldering iron. The tip is very hot!



2. The soldering iron tip can easily damage insulation which may lead to short circuits.
3. Use pliers, tweezers and other tools to safely handle wires and boards during soldering.
4. Do not pull wires too hard during desoldering operations to avoid burns by molten solder.
5. Use a special holder (a "helping hand") for soldering small components.



6. Don't hold the soldering iron by its wire or tip. Unplug the iron during breaks and after work.

> Unplug the soldering iron in case of malfunctions and/or fire.

Rosin and solder emit a considerable amount of harmful substances when heated. Vent the room after each soldering session. Take breaks every 30 minutes for full room ventilation; don't forget to unplug the soldering iron during these breaks.

Flights

Safety during pre-flight preparations

- Make sure that the Li-ion batteries are charged.
- Make sure the batteries in the control equipment are charged.
- Attach the propellers just before flying.

Check the following:

- Tightness of propeller nuts.
- Attachment and integrity of propellers guards.
- Reliability of wires attachment, absence of loose wires.

Safety before flight

- Place the spectators behind the pilot, or behind the line passing through both shoulders of the pilot behind the pilot.
- Do not allow spectators into the hemisphere in front of the pilot.
- Know and remember the flight duration that the copter and its battery are designed for.

- BEFORE connecting the Li-ion battery enable control equipment (the remote), and set the left stick (throttle) to the zero position.
- Connect the Li-ion battery immediately before takeoff, disconnect it immediately after landing.
- Stay at least 3 m away from the copter.
- Take off from a level flat site at the distance of at least 3 meters away from obstacles.

Flight safety

- Follow all instructions of the teacher or the flight instructor.
- Specify the flying area in advance. Only fly in the specified area, and avoid flying outside it. Not to fly over behind your back.
- When learning to fly, fly below the level of your height.
- Fly in proximity to yourself at a distance at which you can see the copter orientation in space. Do no fly far away from yourself. If you doubt copter orientation, immediately land on the spot. Do not try to take off. Approach the copter and take off.
- During the flight, move the control sticks carefully and smoothly. Avoid abrupt movements. If you have to change the flight direction, move the sticks vigorously, but not abruptly.
- Fly carefully, and perform only those flight elements that you are sure you can perform. Never perform the flight maneuvers that you doubt you can perform, and the maneuvers involving risks.
- Observe the speed limit. The copter speed should be maintained within the speed of a walking man.
- Return the copter to the landing location by the estimated time, prevent complete discharge of the battery during the flight.
- Land only on a flat open area away from obstacles

Emergency landing

In case of hitting the ground or a heavy landing, do the following:

1. Stop the flight. Land the copter on the ground. Set the left stick (throttle) to the minimum
2. Disarm (Move the left stick left-down for 3 seconds)
3. Disconnect the Li-ion battery on the copter.
4. Turn off the remote.
5. Inspect the copter, and repair if necessary.

Scheduled landing

After a scheduled landing, do the following:

1. Disarm (Move the left stick left-down for 3 seconds)
2. Disconnect the Li-ion battery on the copter.
3. Turn off the remote.

Clover drone assembly

This section contains articles describing the assembly of each version of Clover.

Version	Image
Clover 4.2 (4.2 WorldSkills)	
Clover 4	
Clover 3	
Clover 2	

Links to Clover's parts CAD-models are available in the "CAD-models" article.

Determination of version

Use serial number to determine the version of your Clover.



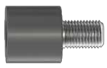
In this case the number *420* indicates the Clover version **4.2**.

Clover 4.2 assembly

Dimensional drawing – clover-4.2.pdf.

Fasteners size

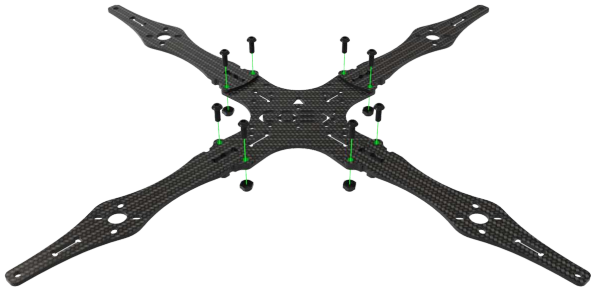
> During assembly, screws and racks of various sizes are used, using fasteners of the wrong size can damage the copter.

	Screw M3x10		Aluminium rack 40 mm
	Screw M3x8		Aluminium rack 15 mm
	Screw M3x5		Nylon rack 40 mm
	Screw M2x5		Nylon rack 30 mm
	Nut M3 (self-locking)		Nylon rack 20 mm
	Nut M3 (nylon)		Nylon rack 15 mm
	Damper rack		Nylon rack 6 mm

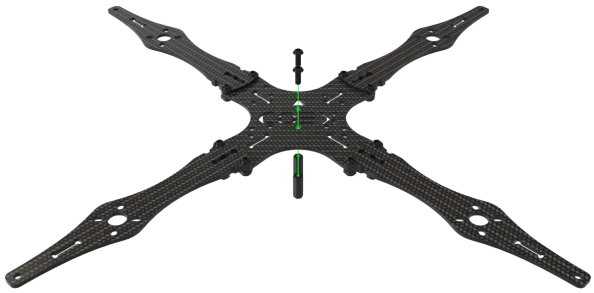
Frame Assembly

1. Align the 4 beams with the center deck, fix them with the M3x8 screws and nuts with a nylon insert.

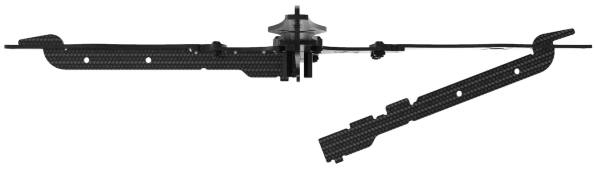




2. Install 2 aluminum 15mm posts on the center holes in the main deck and fix them with the M3x8 screws.



3. Install the stiffener hook into the groove in the beam.



4. Press the stiffeners onto the main deck.



5. Tighten the stiffeners with a small carbon deck.

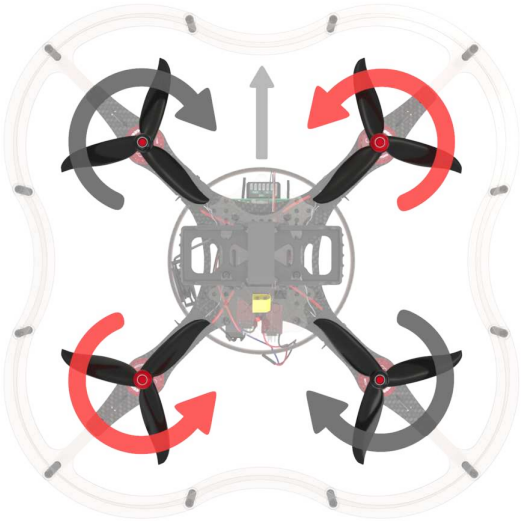


6. Install 4 6mm nylon posts and fix them with the M3x5 screws.



Installing motors

1. When installing motors, pay attention to the rotation scheme of the motors. The rotation marking on the motors must match the rotation pattern.



2. Mount the motor on the corresponding holes in the beam using **M3x5 screws**.

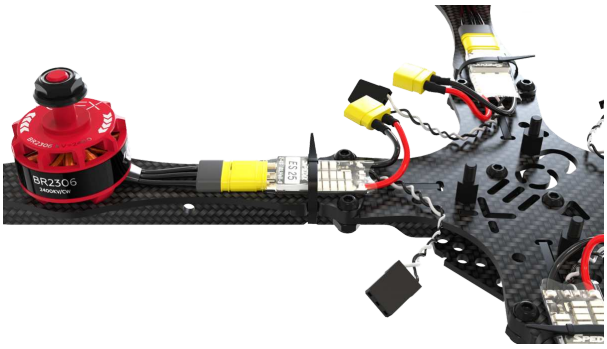


> Make sure that the motors are secured with M3x5 screws, otherwise a short circuit between the windings may occur.

Installing ESC and PDB

1. Connect the speed controllers (ESC) to the motors using the MR30 connectors and fasten them to the beams using clamps.





2. Install the power distribution board (PDB) on the pre-mounted racks and secure it with 6mm racks. The power distribution board must be installed so that the power connection cable points toward the tail of the aircraft.



3. Connect the power outputs of the speed controllers to the power distribution board.



Installing Flight Controller

Clover 4 drone kit allows you to install various flight controllers, for example COEX Pix and Pixracer.

> During installation the flight controller, pay attention to the arrow located on the board, it should be directed to the copter bow.

COEX Pix

> Before installing the damper struts, screw 2 layers of nylon nuts, for a stronger fix or bite off the excess thread using side cutters.

Install the damper struts, fix COEX Pix on them with nylon nuts.



Pixracer

1. Place the small deck on the racks and secure it with nylon nuts.

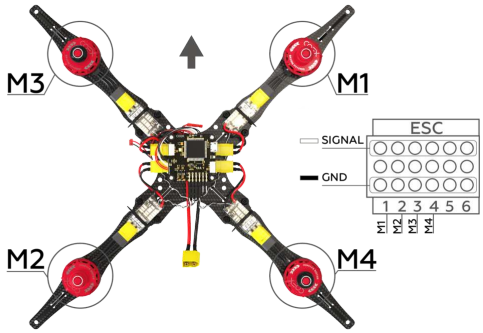


2. Glue 3-4 layers of double-sided tape, glue it in the center of the small deck and install Pixracer on top.



Connect Flight Controller

1. Connect the speed controllers to the flight controller in accordance with the diagram.



2. Connect the power cable to the power distribution board(PDB) and the corresponding connector on the flight controller.



3. Install 40mm aluminum racks on the M3x10 screws.



Installing Raspberry Pi

1. Install the 20 mm racks on the main deck, fix them with the M3x8 screws.



2. On a mounting deck, install M2.5x6 mm racks and M3x30 mm racks, fasten them with the M2.5x4 and M3x10 bolts, respectively.



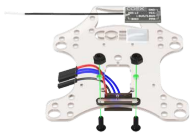
3. Install the assembled mounting deck on the main deck and fix with the M3x8 screws.



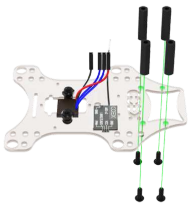
4. Install the Raspberry Pi circuit board and fix with M2.5x4 bolts.



5. On the capture deck, install the rangefinder using self-locking nuts and M3x8 screws, and glue the radio using double-sided tape.



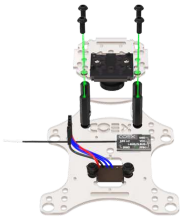
6. Install 4 20 mm racks and fix them with the M3x8 screws.



7. Install the camera on the small mounting deck and fix it with 2 M2x5 self-tapping screws in the upper left and lower right corners.



8. Install the camera module on the capture deck and fix with the M3x8 screws.



9. Install the assembled pickup deck and fix with the M3x8 screws.



10. Connect to the Raspberry Pi rangefinder and power cable.



11. Connect the camera cable to Raspberry Pi.

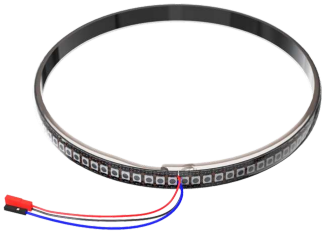


Installing LED strip and legs

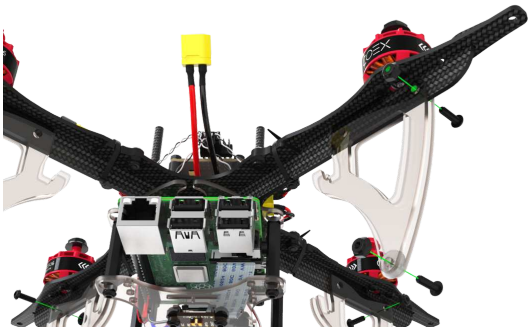
1. Assemble the hoop for the LED strip by combining the lock on the ends.



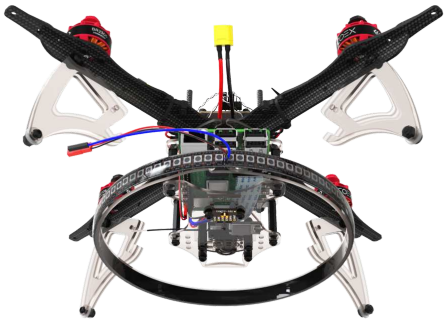
2. Stick the LED strip on the hoop, for better fastening, pull it with 3-4 clamps.



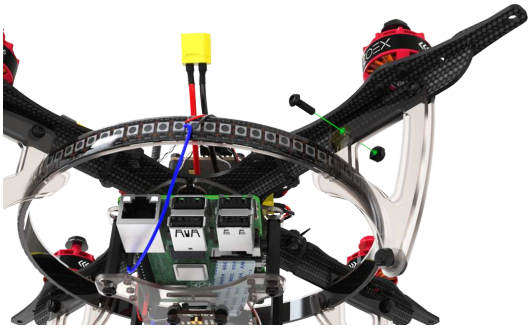
3. Install the legs on the stiffening plate using self-locking nuts and M3x8 screws using only the extreme mounting holes. From below, between the plates of the legs, install a damper silicone ring.



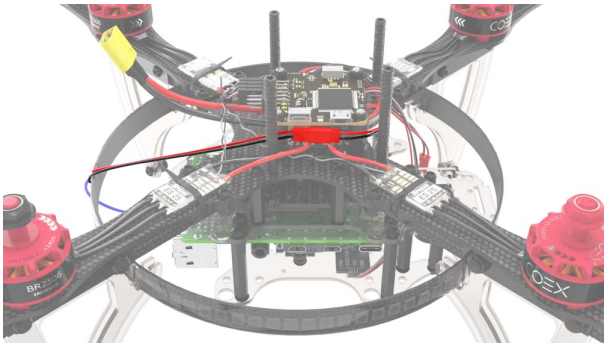
4. Bend the legs back and install a hoop with LED strip in a special groove on them so that the connection cables exit from the tail side of the copter.



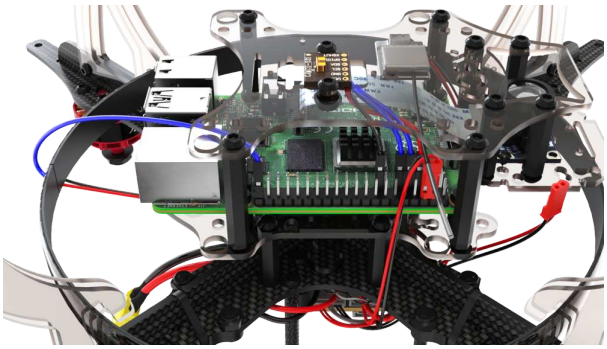
5. Behind fasten the legs with self-locking nuts and M3x10 screws.



6. Connect the LED strip power (red, black cables) to the short JST connector on the PDB.

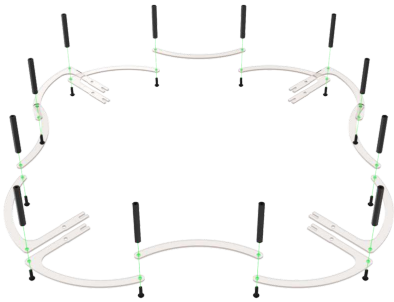


7. Connect the signal output of the LED strip (white cable) to Raspberry Ri, to pin *GPIO21*.

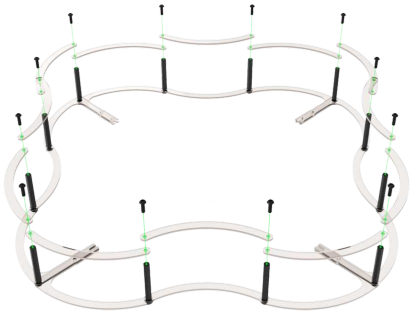


Installing guard

1. Assemble the lower level of guard with 40mm racks and M3x10 screws.



2. Assemble the top level of protection with the M3x10 screws.



3. Install the mounting deck and fix it with M3x8 bolts.



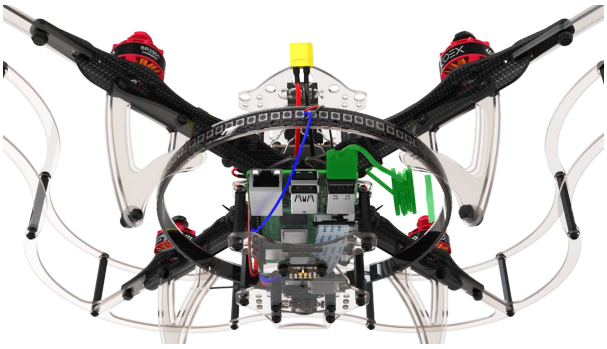
4. Establish protection and fix on beams by means of self-locking nuts and M3x10 screws.



Flight preparation

1. Install the battery strap and connect the flight controller to the Raspberry Pi using a USB cable.





2. Install the propellers in accordance with the directional diagram of the motors.



3. Install the battery.



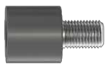
The drone is assembled, then perform setup.

Clover 4 assembly

Dimensional drawing – clover-4.2-ws.pdf.

Fasteners size

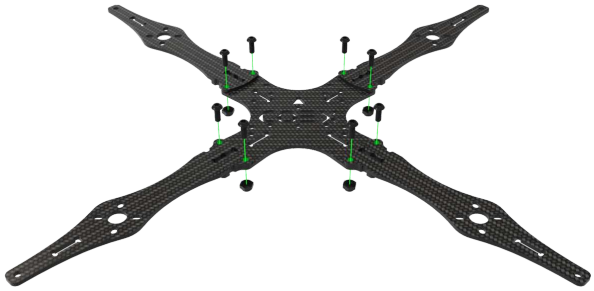
> During assembly, screws and racks of various sizes are used, using fasteners of the wrong size can damage the copter.

	Screw M3x10		Aluminium rack 40mm
	Screw M3x8		Aluminium rack 15mm
	Screw M3x5		Nylon rack 40mm
	Screw M2x5		Nylon rack 30mm
	Nut M3 (self-locking)		Nylon rack 20mm
	Nut M3 (nylon)		Nylon rack 15mm
	Damper rack		Nylon rack 6mm

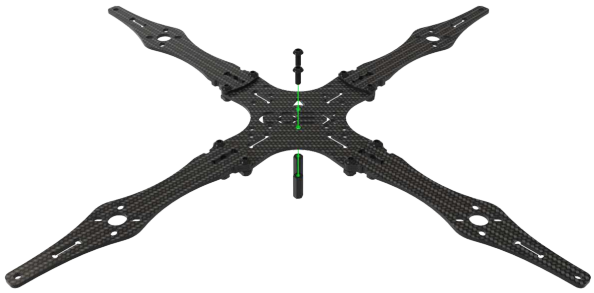
Frame Assembly

1. Align the 4 beams with the center deck, fix them with the M3x8 screws and self-locking nuts.

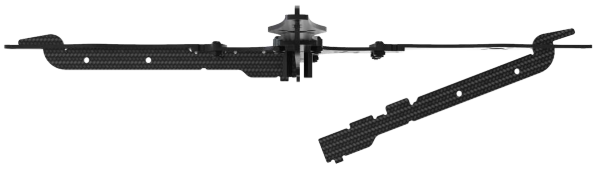




2. Install 2 15mm posts on the center holes in the main deck and fix them with the M3x8 screws.



3. Install the stiffener hook into the groove in the beam.



4. Hold the stiffeners tight to the main deck.



5. Tighten the stiffeners with a small carbon deck.

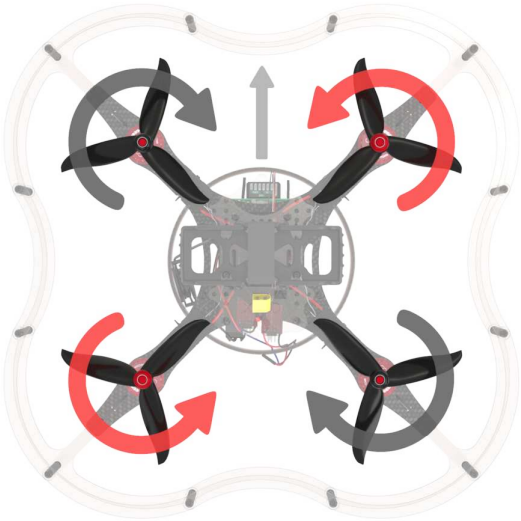


6. Install 4 6mm nylon posts and fix them with the M3x5 screws.



Installing of motors

1. When installing motors, pay attention to the rotation scheme of the motors. The rotation mark on the motors must match the rotation scheme.



2. Mount the motor on the corresponding holes in the beam using **M3x5 screws**.



> Make sure that the motors are secured with M3x5 screws, otherwise a short circuit between the windings may occur.

Installing ESC and PDB

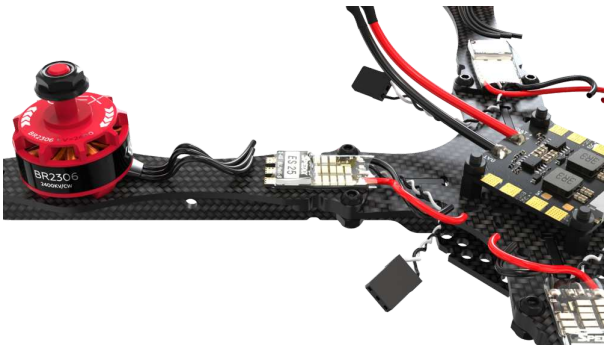
1. Install the Power Distribution Board (PDB) on the pre-mounted stands, it must be installed with the power cable pointing towards the rear of the aircraft.



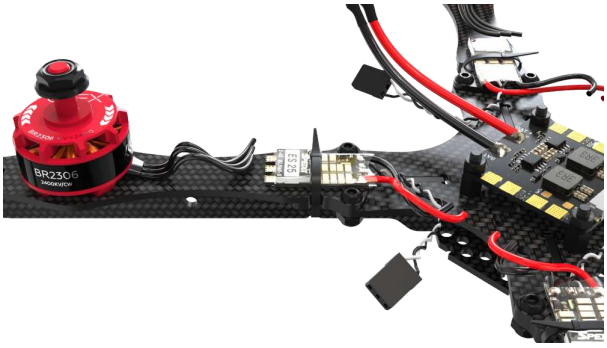


2. Install the speed controllers (ESC) to the appropriate positions on the beam.

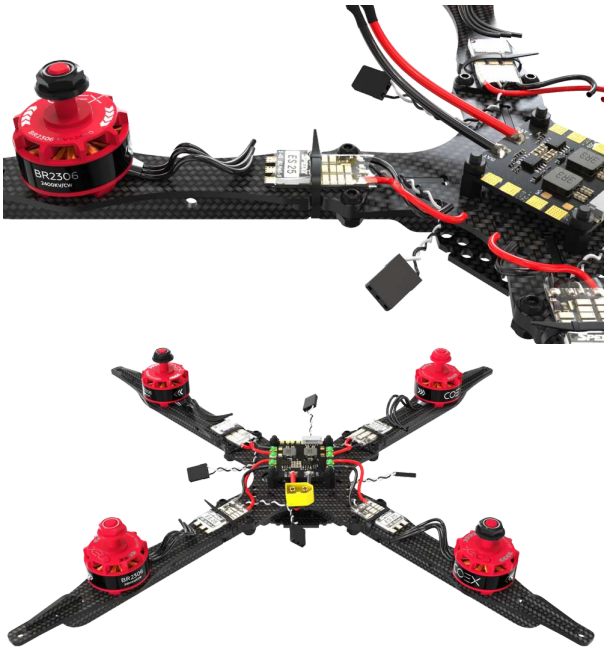




3. Tighten the speed controllers (ESC) with cable ties.

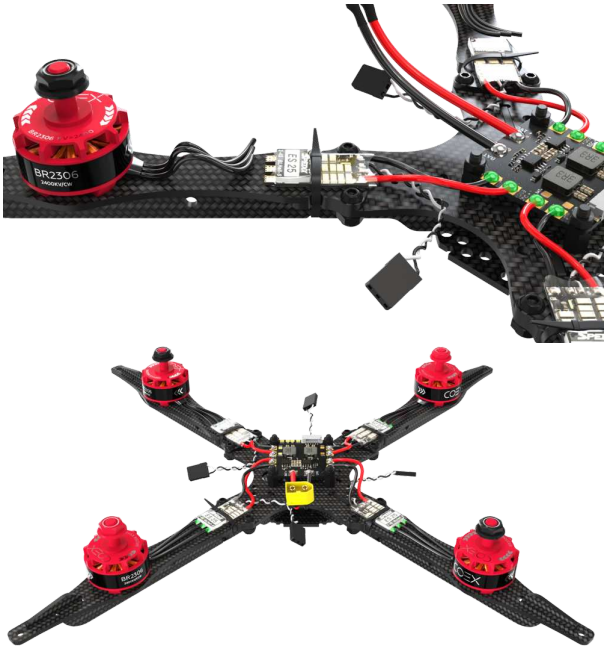


4. Measure out the required amount of ESC power wire, and cut off the excess.
5. Strip and tin the cut wires
6. Tin the contact pads on the power distribution board.
7. Solder the ESC power wires to the power distribution board.

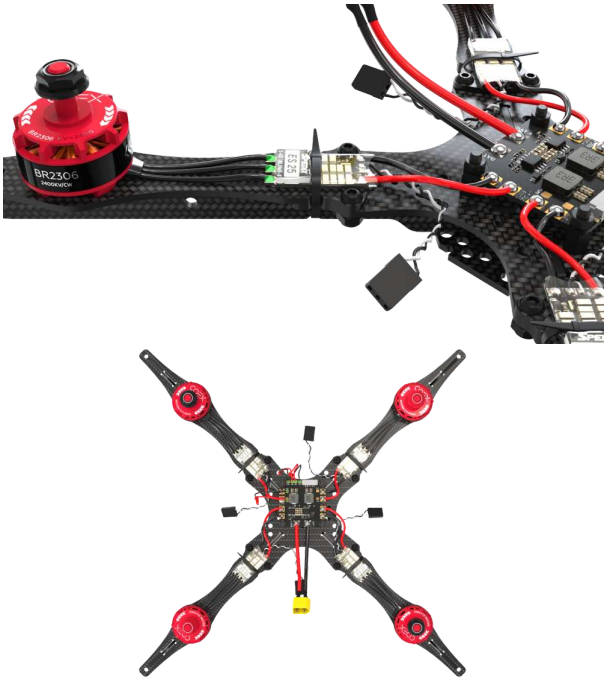


> Be careful with the pin signatures on the board. The red wire should go to the site with the signature +, and the black one to the signature -.

8. Cut off the excess phase cable coming from the motors.
9. Strip and tin the phase cables.
10. Tin the contact pads of the governors.
11. Solder the phase cables to the contact pads of the regulators in any order.



12. Solder 3 female JST connectors to *5V* pads and *bat+* pad



Installing the flight controller

The *Clover 4* set allows you to install various flight controllers, for example *COEX Pix* and *Pixracer*.

> When installing the flight controller, pay attention to the arrow located on the board, when installing it must be directed towards the nose of the aircraft.

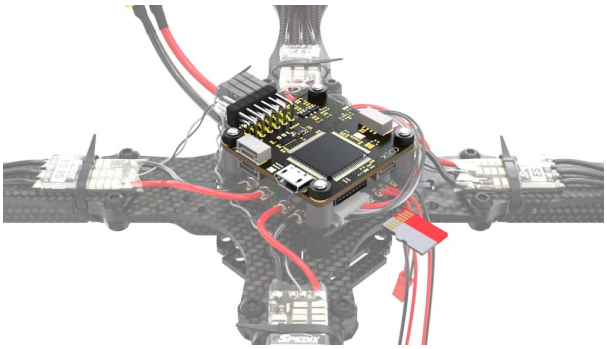
COEX Pix

> Before installing the damper struts, screw in 2 layers of nylon nuts for a stronger hold or bite off excess threads with side cutters.

1. Secure the power distribution board with nylon nuts and mount the damper posts on top.
2. Install the flight controller and secure with nylon nuts.



3. Insert the flash card for logging into the flight controller.



Pixracer

1. Secure the power distribution board with 6mm nylon struts.
2. Install the small mounting deck and secure with nylon nuts.



3. Glue 3-4 layers of double-sided tape, glue it in the center of the small deck and place the *Pixracer* on top.

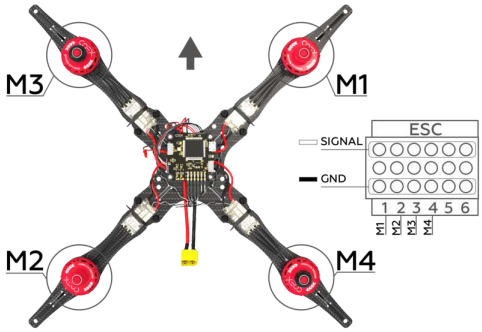


4. Insert the MicroSD card into the flight controller.

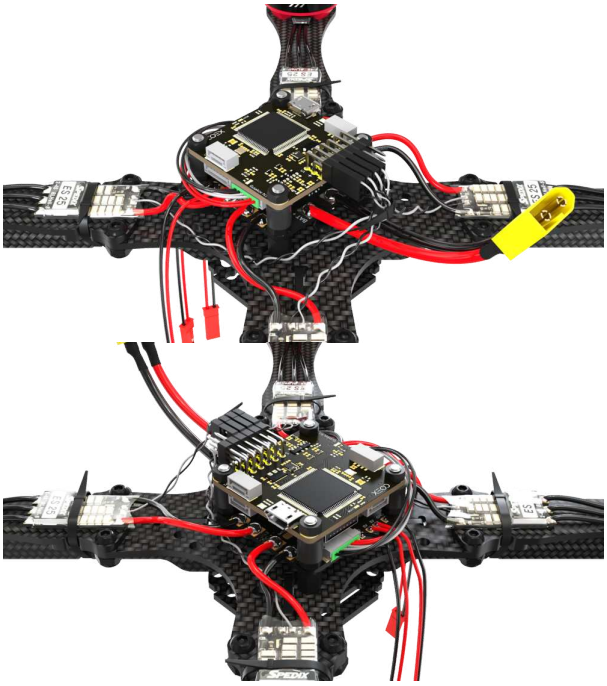


Connecting a flight controller

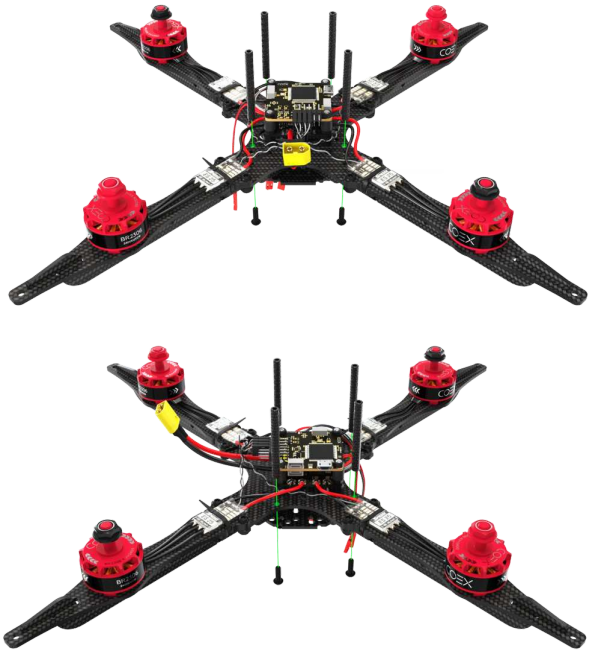
1. Connect the ESCs to the flight controller according to the scheme.



2. Connect the power cable to the power distribution board (PDB) and the corresponding connector on the flight controller.

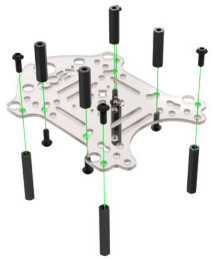


3. Install the 40mm aluminum posts onto the M3x10 screws.



Installing Raspberry Pi

1. Place the 20 mm and 40 mm posts on the mounting deck, fix them with M3x8 screws.

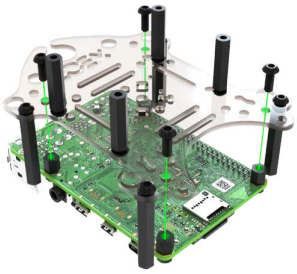


2. Use an M3x10 bolt to cut M3 carving in the Raspberry Pi mounting holes.
3. Screw the 6mm racks into the Raspberry Pi board, secure them with nylon nuts if necessary.





4. Mount the Raspberry Pi onto the mounting deck using M3x6 screws.



5. Install the assembled module into the corresponding slots on the main deck of the drone.



6. Plug the 5V JST into the corresponding power pins on the Raspberry Pi.

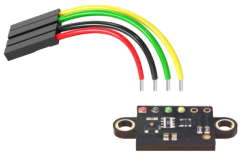


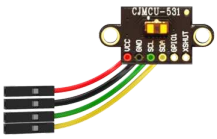
7. Take 4 Dupont wires, cut 5–7 cm of cable and solder to the corresponding pins of the rangefinder.





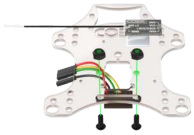
| Wire | Rangefinder pin | |------|----------------| | Red | 5v | | Black | GND | | Yellow | SDA | | Green | SCL |



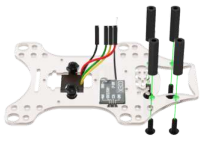


8. Install the rangefinder on the grip deck and glue the radio to the 3M tape.

> Install the rangefinder so that the nuts do not rest directly on the board. With this installation, if there is a high probability of damaging the board elements.



9. Install 4 20mm nylon posts and fix them with M3x8 bolts.



10. Place the camera on the small mounting deck and secure it with two short self-tapping screws.

> If you attach the camera to the upper right corner and the screw head touches the element on the camera, the camera will not work.



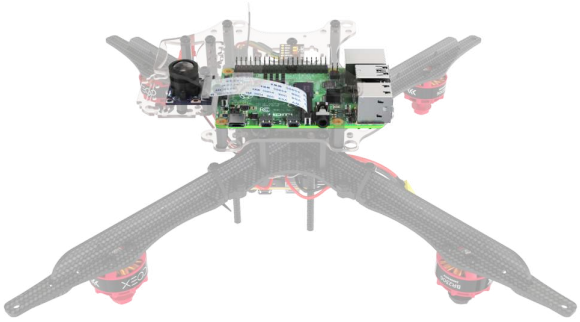
11. Place the small mounting deck with the camera on the stands and secure with the M3x8 bolts.



12. Place the assembled module over the Raspberry Pi module and fix it with M3x8 bolts.



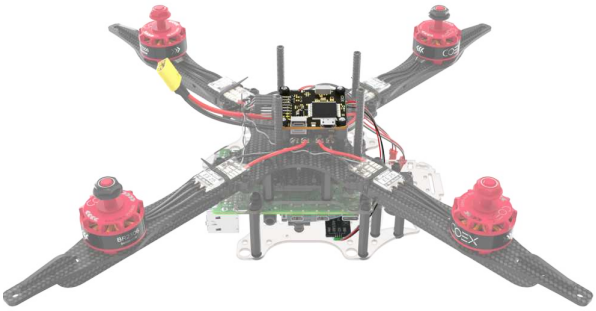
13. Connect the camera and Raspberry Pi with a ribbon cable.



14. Connect the rangefinder to the Raspberry Pi into the appropriate pins.



15. Connect the radio and the flight controller with a 4-pin cable.

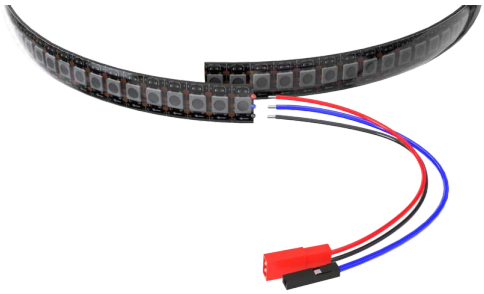


Installing LED strip and legs

1. Assemble the hoop for the LED strip by locking the ends together.



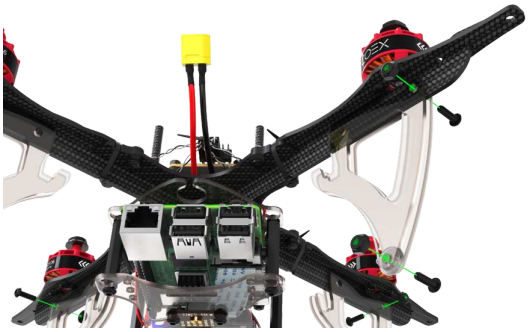
2. Solder the JST male to the power pad and the Dupont-female to the signal pad.



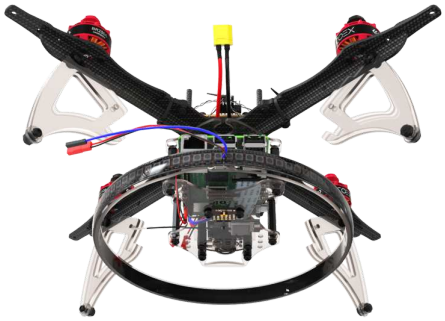
3. Stick the LED strip to the hoop, for greater strength, tighten it with 3-4 clamps.



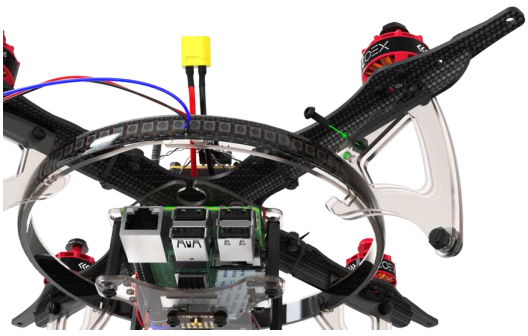
4. Install the feet on the stiffener plate with self-locking nuts and M3x8 screws only using the outermost mounting holes. From below, between the plates of the legs, install a damper silicone ring.



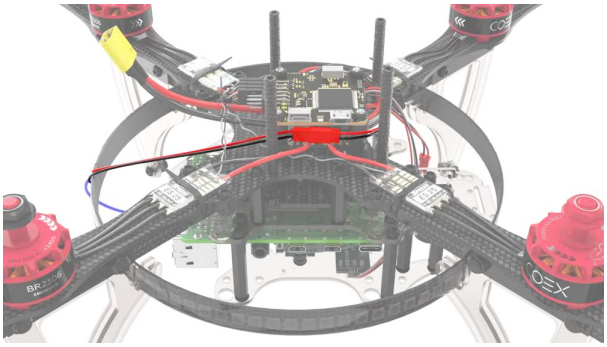
5. Bend the legs back and install the hoop with the LED strip in the special groove on them, so that the connection cables exit from the tail side of the copter.



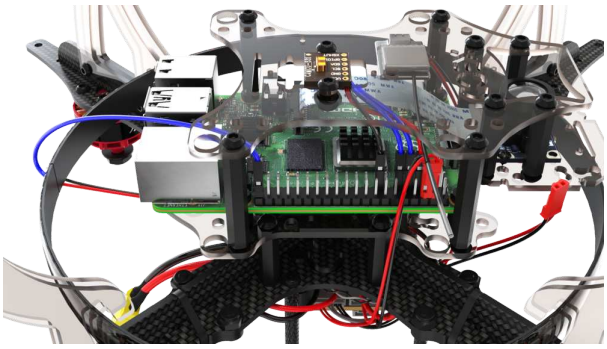
6. Secure the legs with self-locking nuts and M3x10 screws.



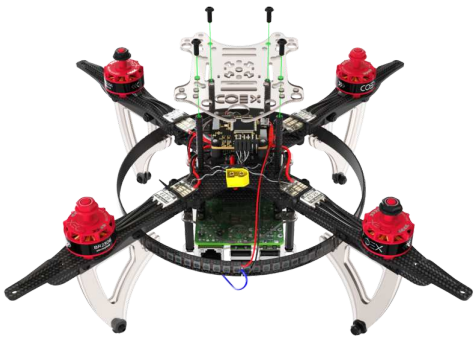
7. Connect the LED strip power to the JST 5V connector on the power distribution board.



8. Connect the signal output of the LED strip in the Raspberry Ri to pin *GPIO21*.

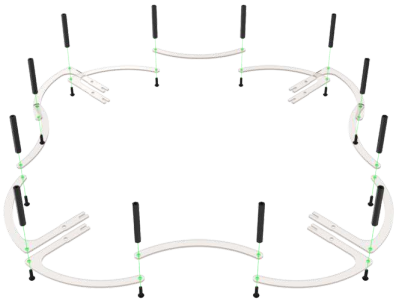


9. Install the mounting deck and secure it with M3x8 screws.

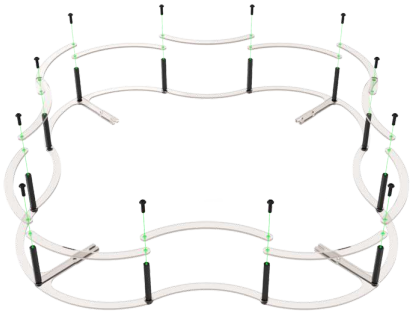


Setting protection

1. Assemble the lower level of protection using 40mm posts and M3x10 screws.



2. Assemble the upper layer of protection using the M3x10 screws.



3. Install the protection and fix it to the beams with self-locking nuts and M3x10 screws.



4. Connect the flight controller to your Raspberry Pi using the USB cable.

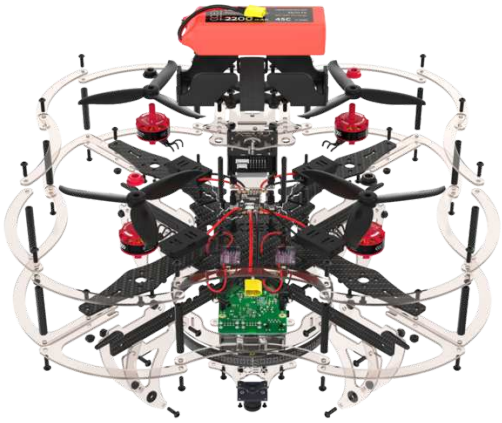


5. Install the battery strap.



The drone is assembled, next perform the "setup" step.

Clover 4 assembly



Frame base assembly

> To increase the strength of the frame, you can print on a 3D printer or cut on a laser cutter reinforcing pads.

1. Mount the reinforcement pads on the stiffening ribs if you have them. Proceed without them if you don't.



2. Align two carbon stiffening ribs using the center notch.



3. Install the top carbon deck using notches as guides.



4. Place self-locking steel nuts into the slots in reinforcement plates and tighten the assembly with M3x8 screws.

Installing motors

1. Unbox the motors.
2. Shorten the motor wires using wire strippers or side cutters:
    - Cut wires to 30 mm.
    - Strip 5 mm of insulation while taking care to not damage the cores



- Twist the cores.
- Tin the wires. You may want to use tweezers to hold the wire.

3. Place the motor on the support arm.
4. Use hexagonal M3x5 screws to attach the motor to its arm.



Perform these actions for each motor.

Frame assembly

1. Install the support arms on the frame base according to their rotation direction. Use notches as guides.



> Note the motor nut colors when installing the arms. Motors with red nuts should be placed on the front right and back left arms, with black ones - on the front left and back right arms.

2. Attach the arms to the frame base using 8 M3x8 screws, 6 steel nuts, and 2 15 mm spacers.



Preparing the power distribution board

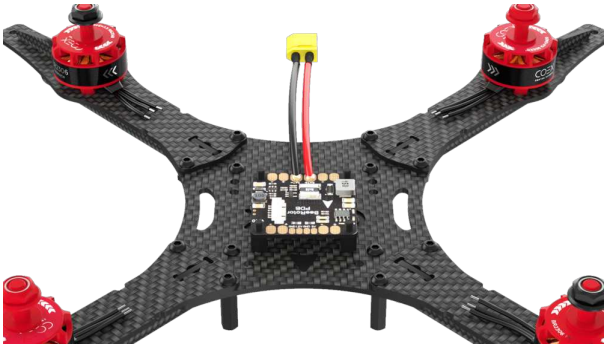
1. Tin the pads on the power distribution board.
2. Check the board for shorts using a multimeter:
   - Set your multimeter to the Continuity Test mode.
   - Ensure your multimeter works by connecting the probes to each other. The multimeter should beep.
   - Connect one of the probes to the **«+»** pad and the other to **«-»/GND**. If there is a short circuit, the multimeter will beep.

Mounting the PDB

1. Attach four 6 mm standoffs on the top carbon deck using M3x6 screws.



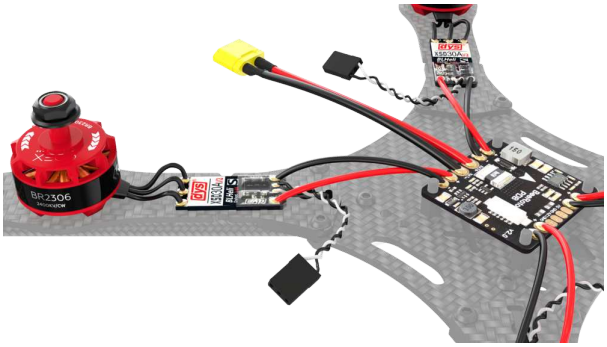
2. Place the PDB on the standoffs.



3. Make sure the arrow on the PDB is pointing in the same direction as the arrow on the top carbon deck.

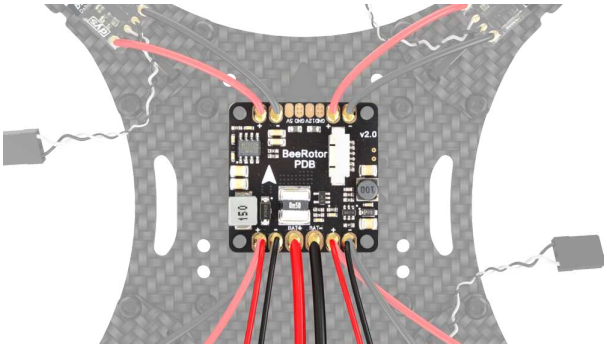
Soldering the speed controllers and the BEC

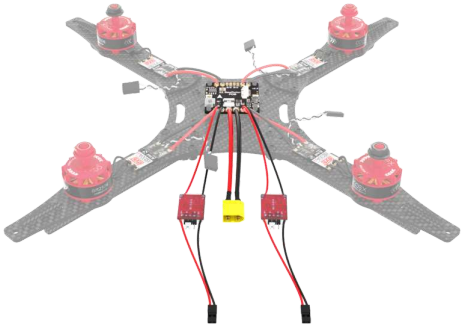
1. Solder the motor wires to the electronic speed controllers (ESCs).
2. Solder the ESC power wires to the power distribution board (**red** to **«+»**, **black** to **«-»**).





3. Solder power wires of the battery elimination circuit in parallel to one of the ESC power wires (**red** to **«+»**, **black** to **«-»**).





4. Check the board for shorts using a multimeter.

Setting up PWM mode on RC

Turn on your transmitter using the **POWER** slider. If the RC transmitter is locked, place all controls in their neutral position:

1. Left stick should be in the **lower center position**.
2. Right stick should be **centered**.
3. The switches (A, B, C, D) should be in the **top position**.



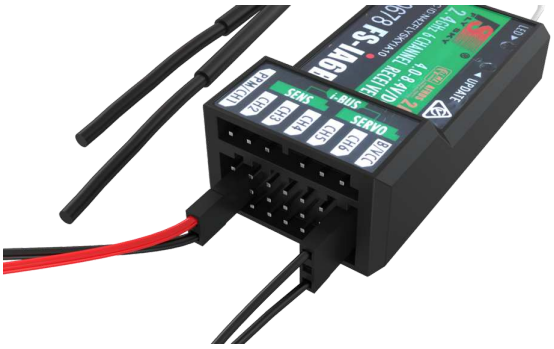
Make sure the transmitter operates in the PWM mode:

1. Power down the receiver.
2. Hold down the "OK" button to enter the menu.
3. Select the "System setup" option, press "OK" to enter the submenu.
4. Select "RX Setup" option.
5. Select "Output mode".
6. Make sure the "PWM" option is selected.
7. Save settings by holding the "Cancel" button.

Binding the RC transmitter and receiver

1. Turn off the RC transmitter with the **POWER** slider.
2. Connect the RC receiver to the 5 V BEC output. Connect the black wire into one of the bottom pins and the red wire to one of the central pins.
3. Place the binding jumper on the B/VCC output.
4. Connect the battery pack.
5. The LED on the RC receiver should start to blink.





6. Hold down the **BIND KEY** on the RC transmitter.
7. Turn on the RC transmitter while holding the **BIND KEY**

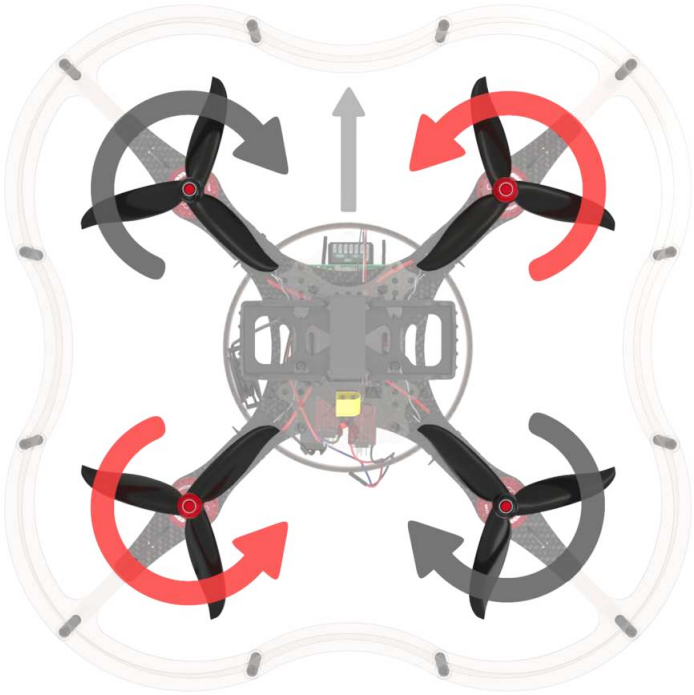


8. Wait for the **RXBind ok** message on the RC transmitter
9. Disconnect the binding jumper.

10. The LED on the RC receiver should be lit continuously.

Checking the motor rotation direction

Motors with **red** nuts should rotate **counterclockwise**, the ones with **black** nuts should rotate **clockwise**. Correct rotation direction should also be printed on the motors. You can use a servo tester or your RC transmitter and receiver to check rotation direction.



1. Disconnect the battery pack and power down the transmitter.
2. Connect the signal wires from the ESC to CH3 pins on the output. The white wire should go to the top pin, the black one should go to the bottom one.
3. Power on the transmitter. Make sure the left stick is in the bottom position.
4. Connect the battery pack.
5. Slowly move the left stick up until the motor starts to spin.

If the motor rotation direction is wrong, switch any two motor wires.

> You can also change motor direction by reprogramming the speed controllers. The process is described in the ESC firmware flashing article.

Do this for each motor.

Switching the transmitter back to PPM mode

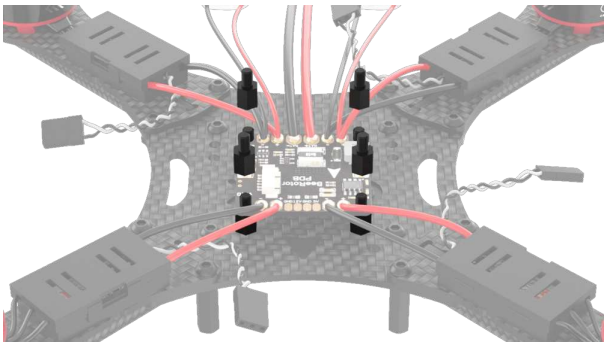
The flight controller expects PPM signal from your RC gear. Switch your transmitter back to PPM before flight.

1. Make sure the receiver is not powered.

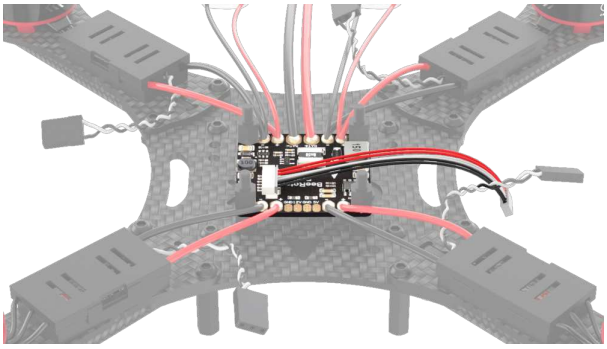
2. Hold down the "OK" button to enter the menu.
3. Select the "System setup" option, press "OK" to enter the submenu.
4. Select "RX Setup" option.
5. Select "Output mode".
6. Make sure the "PPM" option is selected.
7. Save settings by holding the "Cancel" button.

Mounting the flight controller plate

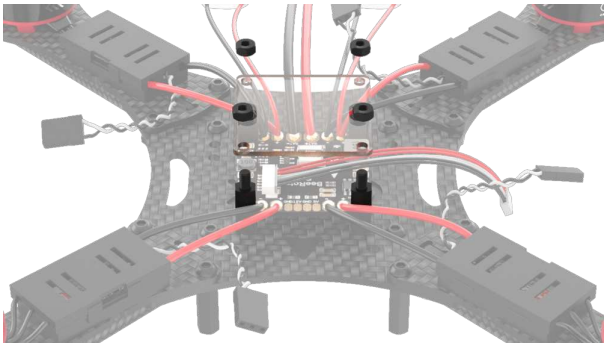
1. Attach four 6 mm standoffs on top of PDB.



2. Connect the flight controller power cable to the PDB.



3. Place the polycarbonate plate on the standoffs and fix them with plastic nuts.

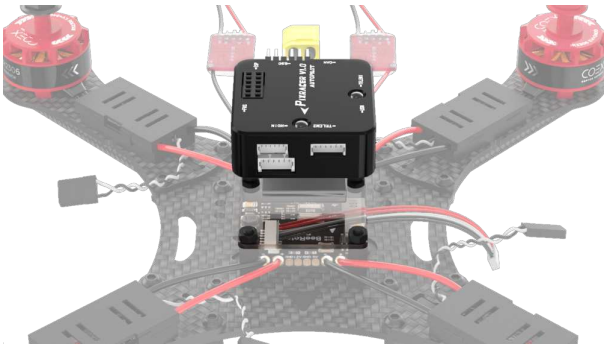


Mounting the flight controller

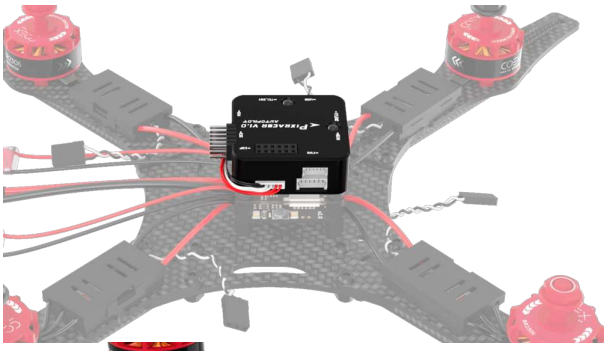
1. Insert the microSD card into your flight controller.



2. Align the flight controller so that the arrows on the controller and on the top carbon deck point in the same direction.

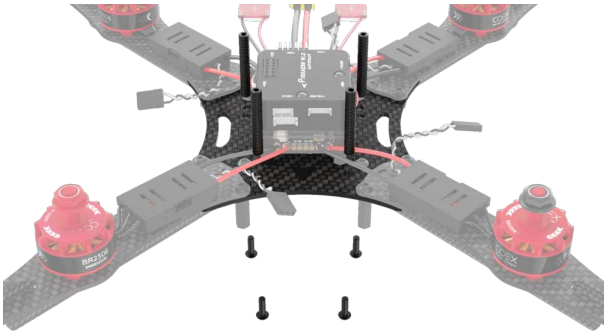


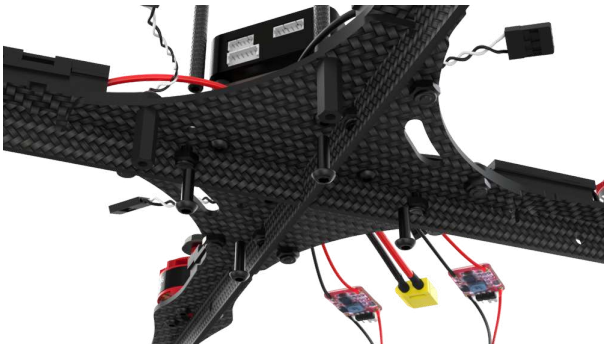
3. Attach the flight controller to the flight controller plate using 3M double-sided adhesive pads.
4. Connect the power cable to the **"POWER"** input of the flight controller.



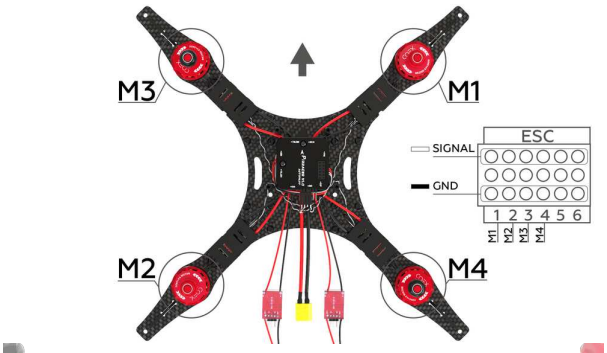


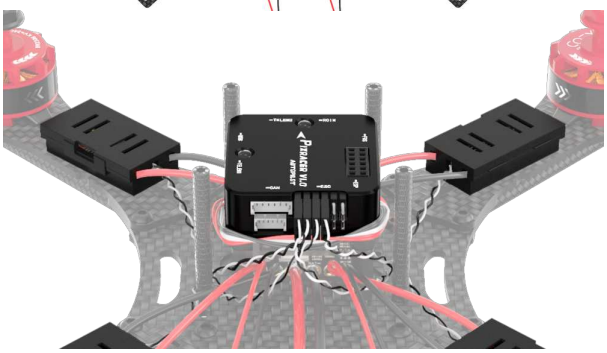
5. Attach four 40 mm aluminum spacers to the top carbon deck using M3x10 screws.





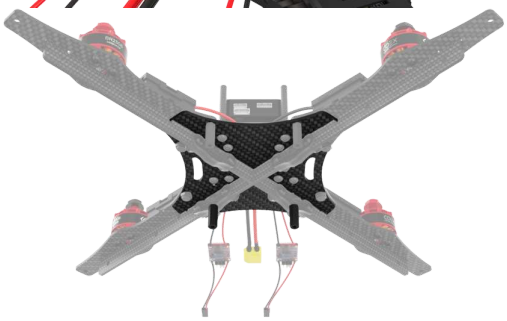
6. Connect signal wires to the flight controller as shown in these pictures:





7. Attach two 15 mm spacers to the top carbon deck using M3x8 screws.





8. Attach two 15 mm spacers to the top carbon deck and the front arms using M3x10 screws (this was already described in the "Frame Assembly" section, p. 2).

Mounting the LED strip ring

1. Bend the polycarbonate strip into a ring and use the locks to fix it in this shape.
2. Fix the ring on the frame using appropriate notches.



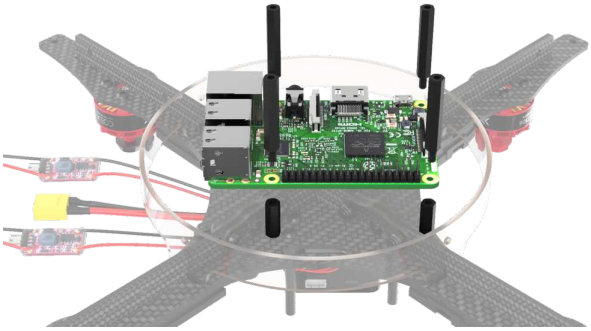


Installing the Raspberry Pi

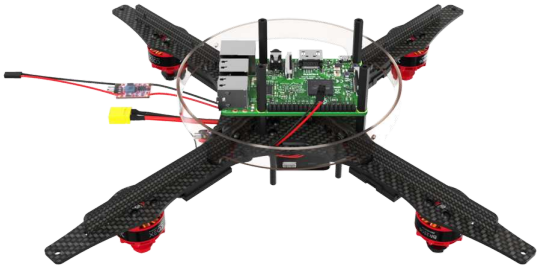
1. Insert your microSD card with our image into the Raspberry Pi



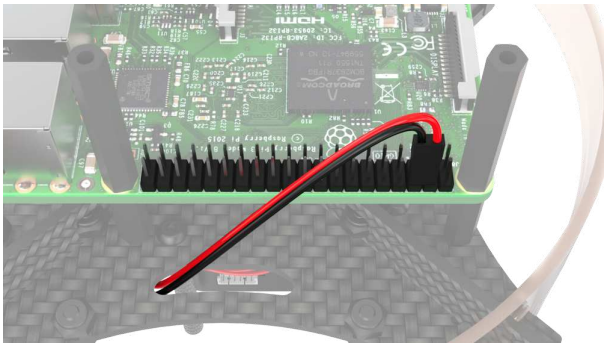
2. Attach the Raspberry Pi using four standoffs.



3. Route the BEC wires through the channel in the top carbon deck.



4. Connect the BEC outputs according to the following image:



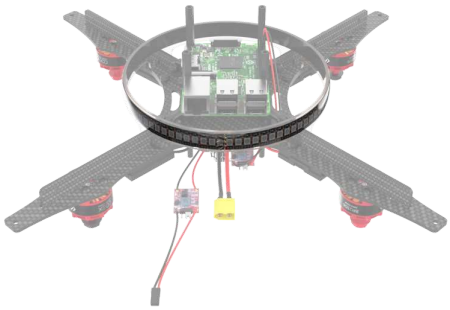
Installing the LED strip on the LED strip ring

1. Check wires on the strip (and solder them on if they're missing)



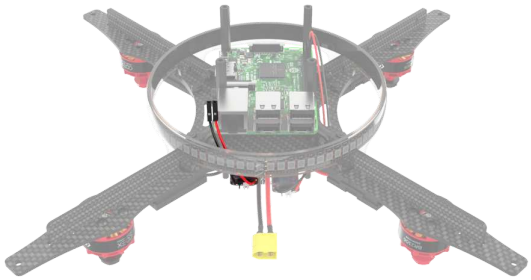


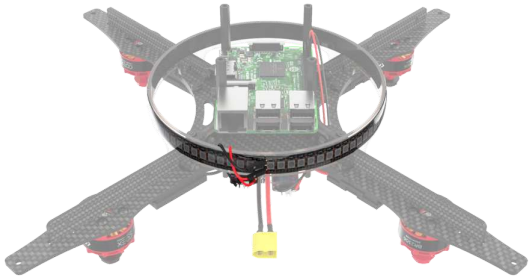
2. Attach the LED strip to the ring using the adhesive layer on the strip. Use zip ties to fix it in place.



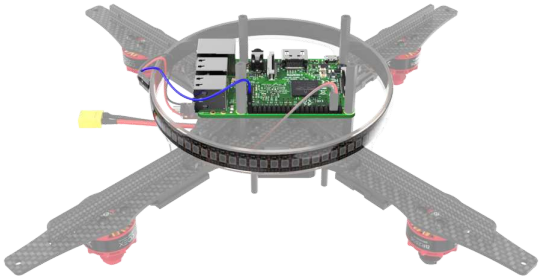
Connecting the LED strip to Raspberry Pi

1. Power the LED strip from a separate BEC. Connect the **«+»** and **«-»** leads to **5v** and **Ground** respectively.



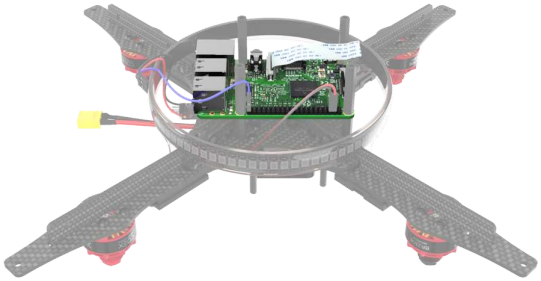


2. Connect the **D** lead to GPIO21 (consult the relevant article for more information).



Installing the camera cable

1. Open the slot connector by lifting the T-clip.
2. Insert the ribbon cable.
3. Press the T-clip down to secure the cable.



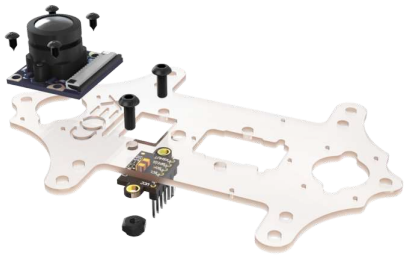
Mounting the lower deck periphery

1. Prepare the laser rangefinder by soldering leads to it.
2. Use four 2x5 self-tapping screws to secure the camera.

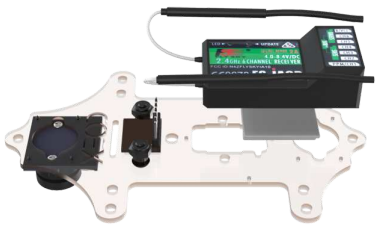
> Make sure the screws don't touch any components on the camera PCB! Otherwise the camera may not function properly.

3. Mount the laser rangefinder on the lower deck using two M3x8 screws and steel nuts.

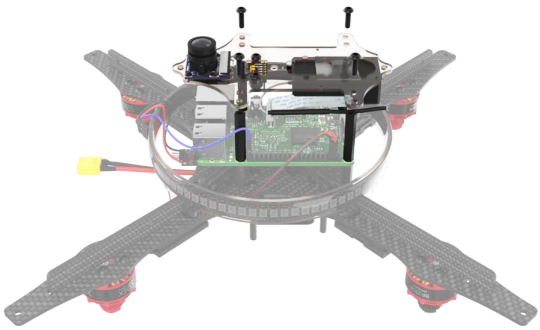




4. Attach RC receiver to the lower deck using 3M double-sided adhesive pads.

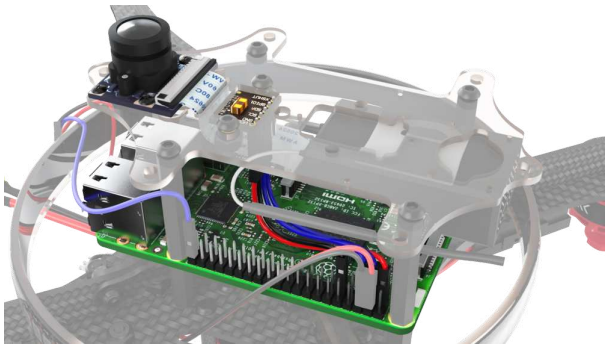


5. Mount the lower deck assembly using four M3x10 screws.



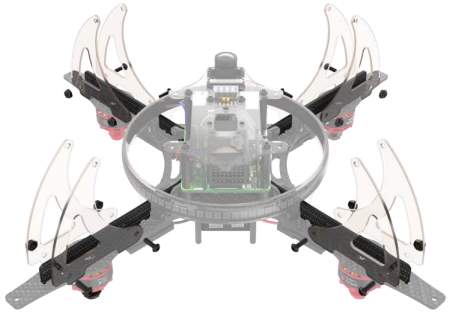
6. Connect the camera ribbon cable to the camera.

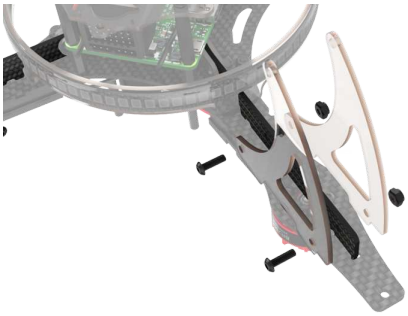
7. Connect the laser rangefinder to the Raspberry Pi using the following pinout:
    - Connect **VCC** to pin 1 (**3v3**)
    - Connect **GND** to pin 9 (**Ground**)
    - Connect **SDA** to pin 3 (**GPIO2**)
    - Connect **SCL** to pin 5 (**GPIO3**)



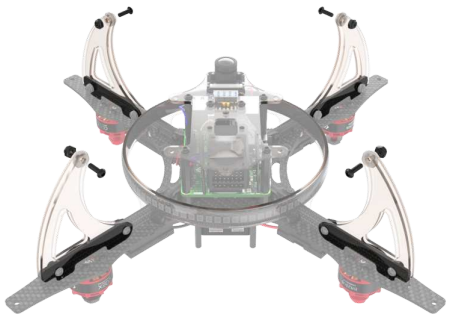
Mounting the landing gear

1. Attach 8 landing gear pieces using M3x10 screws and steel nuts.





2. Attach dampening pads to the landing gear pieces using M3x10 screws and steel nuts.

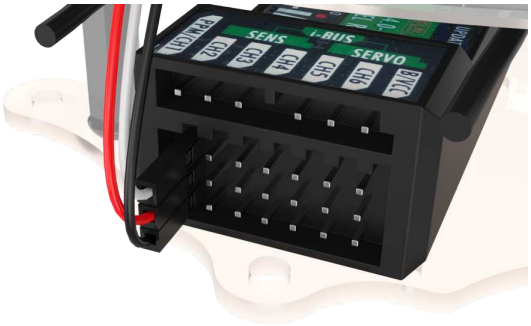


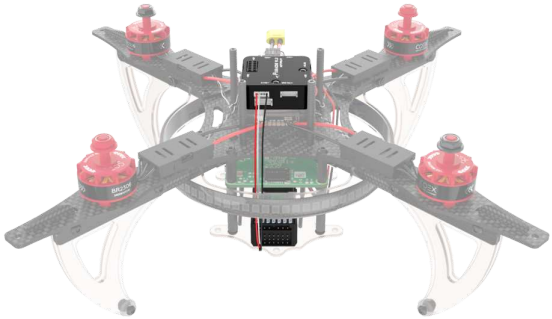
Connecting the cables

1. Connect RC cable to the **RCIN** port on the flight controller.



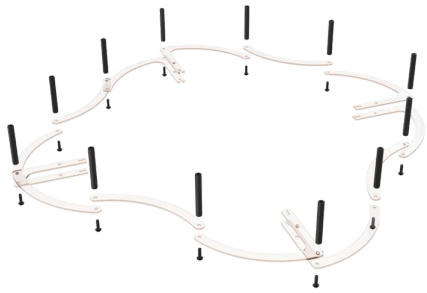
2. Connect RC cable to RC receiver.



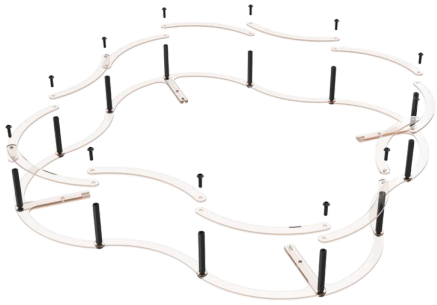


Mounting the propeller guards

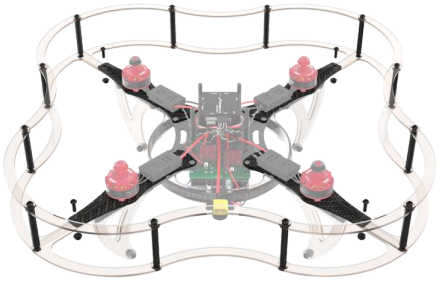
1. Assemble the lower part of the guards using twelve M3x10 screws and twelve 40 mm plastic spacers.



2. Assemble the top part using twelve M3x10 screws.



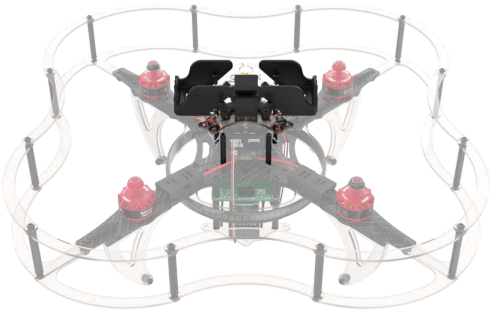
3. Attach the assembly to the drone using four M3x10 screws and steel nuts.



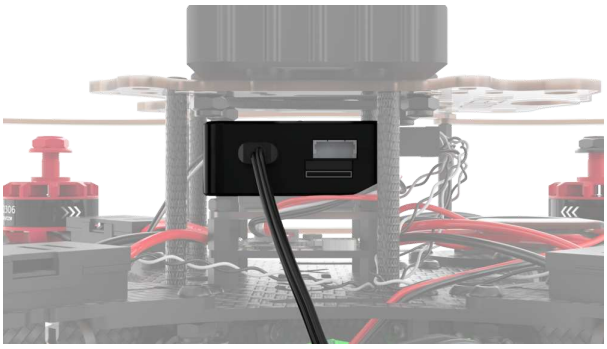


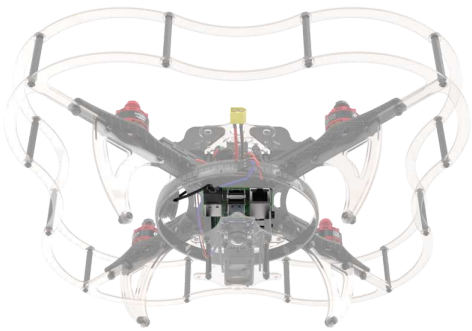
Mounting the top deck

1. Attach the battery holder to the top deck with four M3x8 screws and steel nuts.
2. Thread the battery strap through the slots in the deck.
3. Attach the top deck using four M3x10 screws.



4. Connect the flight controller to the Raspberry Pi using retractable USB cable.





5. Attach the USB cable reel where convenient using 3M double-sided adhesive pads while making sure the cable does not interfere with the propellers.



Mounting the propellers and preparing for flight

Perform the quadrotor components setup according to the "Configuration" section.

> Be sure to **not** mount the propellers **until the setup is complete**. Do it only when you are ready to fly.

Attach the propellers according to their rotation direction. The battery should be disconnected during propeller installation.



Installing the battery

> Make sure all cables are secured and nothing interferes with the propellers!

Check the quadrotor assembly:

- The balance connector should be fixed under the battery strap.
- The ESCs should be zip tied to the frame.
- All wires from the PDB and flight controller should be tucked under a velcro strap wound around aluminum spacers.



Be sure to install and setup the voltage indicator before flying, so as not to overdischarge the battery. To configure the indicator, use the button located at its base. The displayed numbers during setup indicate the minimum possible voltage in each cell of the battery, the recommended value is **3.5**.

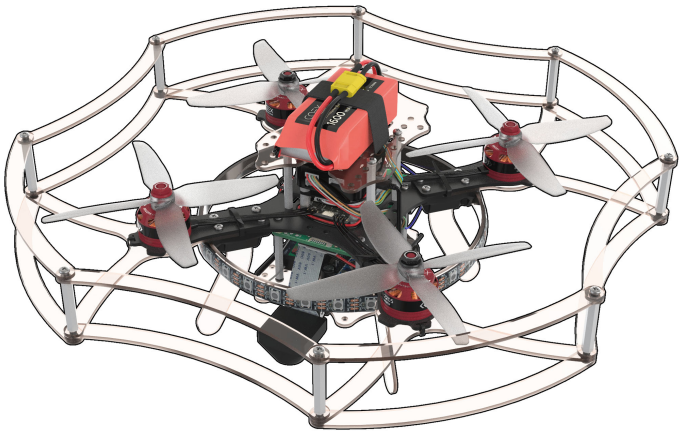
Sound indication means that your battery is low and needs to be charged.



The drone is ready to fly!

Assembly of Clover 3

This manual discusses the assembly of the COEX Clover 3 kit with a 4 in 1 EDC circuit-board.

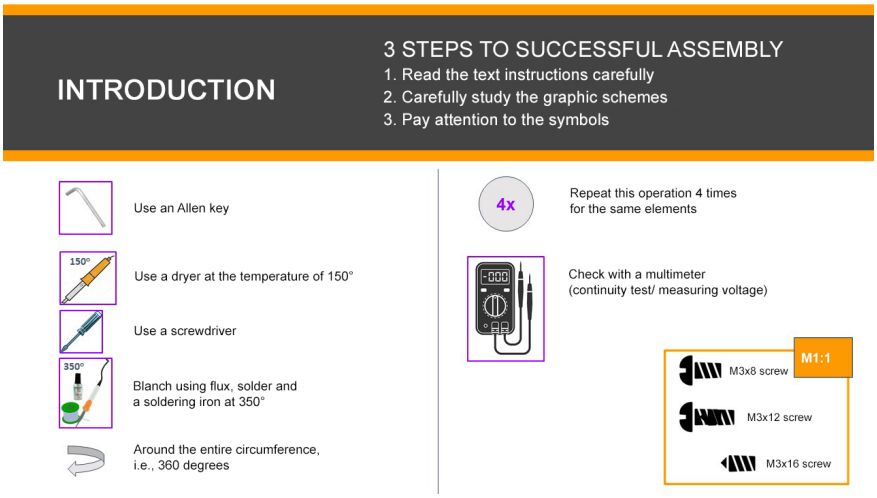


> Before using soldering equipment, be sure to read the safety precautions when soldering.

Additional equipment



Conventional symbols



Motor installation

1. Unpack the motors.
2. Attach a motor to the motor mount with M3x6 hex screws (the shortest screws supplied with the motors).

   A hex wrench is included.

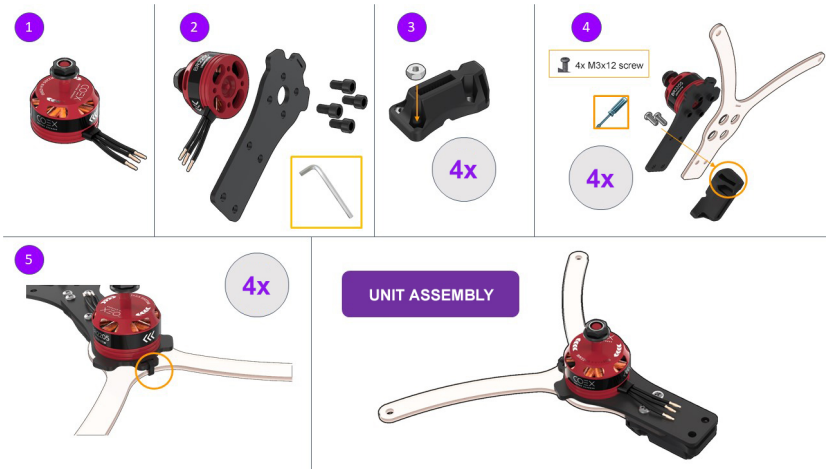
3. Insert M3 nuts (4 pcs) into the plastic holder.

   The choice is yours to use a long screw or pliers.

4. Secure the motor mount, the lower motor mount guard and the holder with M3x12 screws, using a Phillips screwdriver.

5. Using a clamp connect the motor mount and its bottom guard.

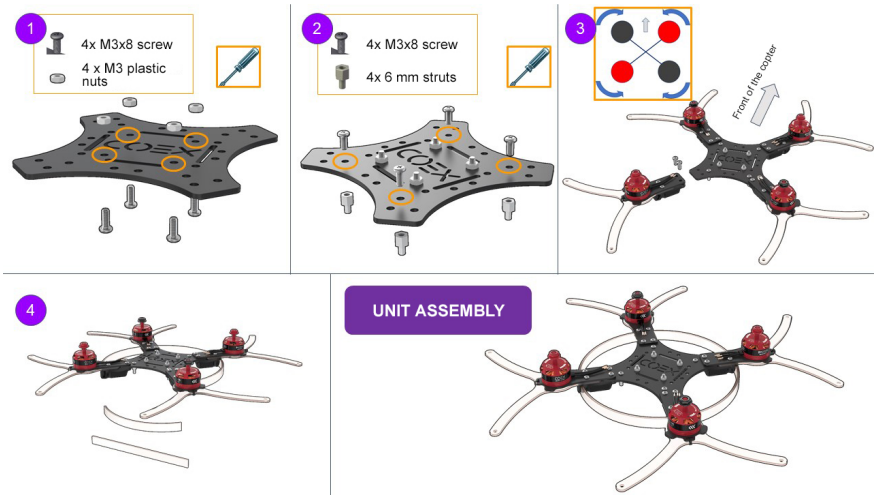
   Cut the remaining part of the clamp (cable tie) with scissors.



Frame elements installation

1. Insert the M3 plastic nuts (4 pcs) for mounting the PDB on the frame with M3x8 screws.

2. Install 6 mm legs (4 pcs) for attaching the Raspberry Pi to the frame with M3x8 screws.
3. Attach the assembled unit to the frame with M3x16 screws, complying with the layout.
4. Install the frame for the LED strip, using the slots in the leg holders.



BEC voltage converter installation(to be soldered and tested)

1. Unpack the power board and install the power ribbon cable.
2. Switch the multimeter in the DC voltage measurement mode (20V or 200V range).
3. Check the correct functioning of the power board by connecting the battery.
   - Voltage measurements are to be made between black and red wires.
   - Output voltage at the XT30 connector should be equal to the battery voltage (10 V to 12.6 V).
   - The output voltage at the power ribbon cable should be between 4.9 V to 5.3 V.
4. Unpack the voltage converter and remove the transparent insulation.
5. Solder two additional wires to the BEC
   - Take 3 male-female wires from the kit (red, black, and any color)
   - The red and black wires are to be tinned on both ends using tweezers. The blue wire is to be tinned from the side of the MALE connector.

     To tin means:

     - Apply flux to the exposed part of the wire.
     - Cover with tin.
   - Solder the red and the black wires to BEC:

     ```
     BLACK -> OUT-
     RED -> OUT+
     ```

6. Check BEC functioning.

- Solder the BEC to the power board:

```
BLACK -> -
RED -> +
```

- Connect the battery and check the voltage at the wires soldered to BEC (from step 5).

  5 V - great, everything is working properly!

  more than 10 V - disconnect the power and move the yellow jumper to the other tweezers.

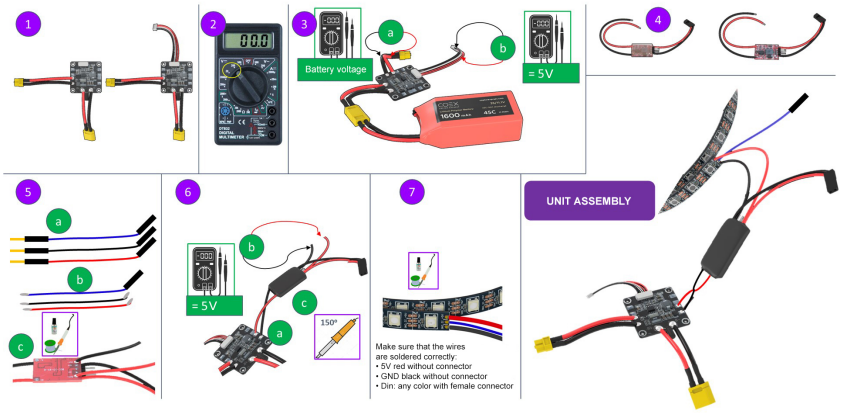
  0 V - not soldered properly.

- If the BEC outputs 5 V, isolate the soldered connection with a black heat-shrink tubing.

7. LED strip installation

- Solder the wires from BEC (from step 5) to the LED strip.
- Remove the silicone layer on the strip (make an incision with a knife and tear).
- Tin the contacts of the LED strip.

```
Red -> +5V
Black -> GND
Blue -> Din
```



4 in 1 ESC board and the PDB power-board installation

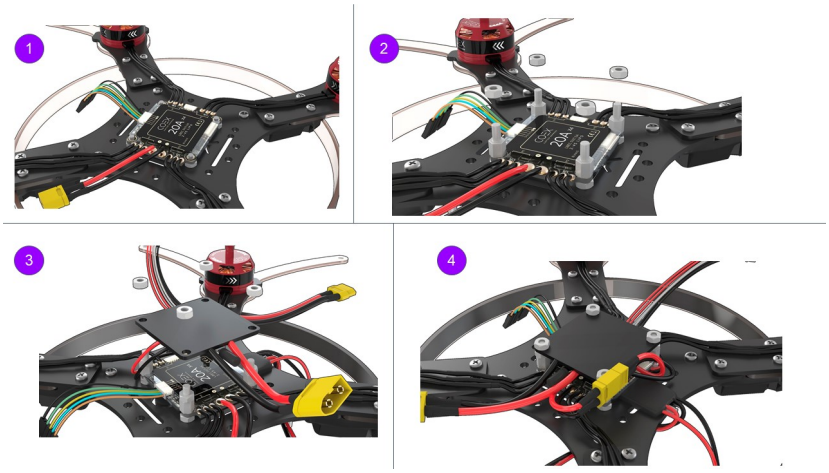
1. Install the 4 in 1 ESC circuit-board as shown in the picture.

   Connect the phase wires of the motors with ESCs wires.

2. Attach the ESC board with 6 mm legs (4 pcs.).

   Screw M3 plastic nuts (4 PCs.) to the legs.

3. Install the PDB power distribution board as shown in the picture (the XT60 connector should point to the tail of the drone).
4. Connect the wires of the PCB power supply board and ESC XT30 board.



Pairing the receiver and transmitter

1. Connect the 5V wire from BEC to the connector of the receiver.

   Insert the BIND connector into the rightmost B/VCC port.
2. Connect the battery. The indicator on the receiver should flash rapidly (reset mode).
3. Press and hold the BIND button on the remote, and switch the remote on.

   The RXBinding process will be displayed on the remote.
4. After pairing (additional lines will be displayed on the remote):
   - Remove the BIND connector from the receiver.
   - Disconnect the battery.



If the remote cannot be powered on, or is blocked, see article remote faults.

Checking the directions of motors rotation

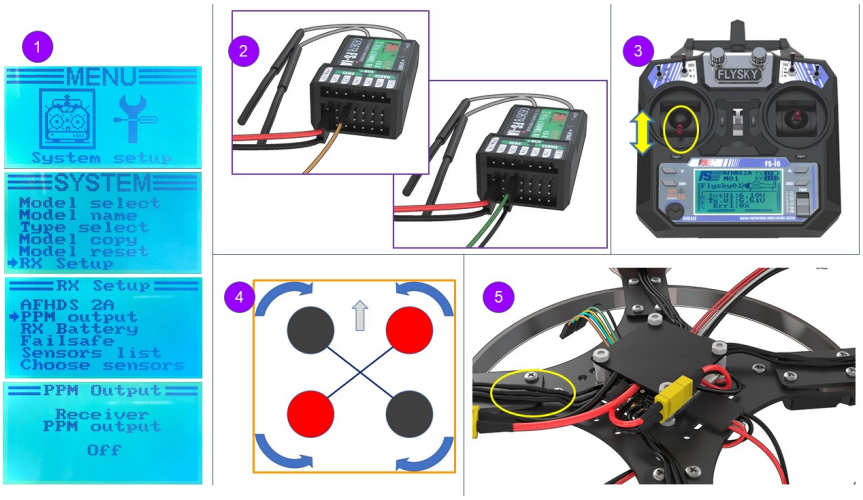
1. Turn the transmitter ON

Make sure PPM in the RX Setup menu is disabled (section "No communications with the flight controller")

In point 3, select “RX setup” > “PPM OUTPUT” > “Off”.

Save changes (hold pressed the “CANCEL” button).

2. Connect the S1 orange wire from the ESC board to CH3 on the receiver. Connect external power.
3. Using the left stick, set throttle to 10 %.
4. Check the motor rotation direction according to the scheme. Repeat for each motor. Thus, it will be clear which motor is controlled.
5. If you have to change the rotation direction, swap any two phase wires of the motor (needs re-connection).

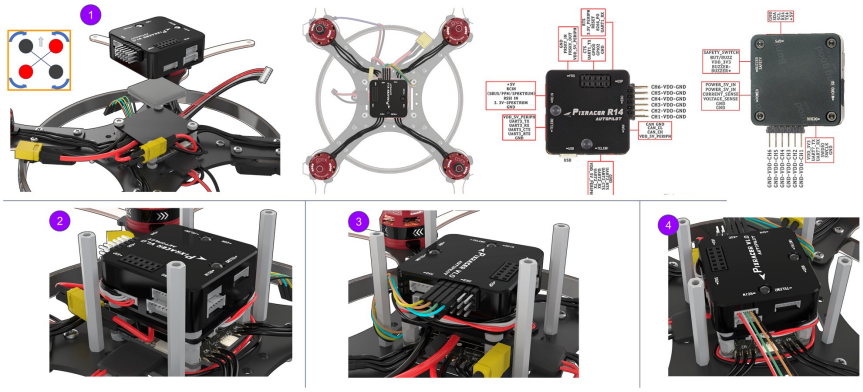


Installation and connection of the Pixracer flight controller

1. Install the Pixracer flight controller on double-sided 3M adhesive tape (2 – 3 layers). The flight controller may also be removed from the housing and firmly mounted on the M3x6 leg.
2. Install 40 mm legs using M3x8 screws.

   Connect the POWER connector.
3. Connect ESCs as shown in the picture.

   More about connecting 4 in 1 ESCs.
4. Connect the ribbon cable from the radio receiver to the RCIN connector in Pixracer.



Raspberry installation

1. Turn the drone upside down.

   Install the Raspberry on the legs using Raspberry mounting holes.

   USB connectors should point to the tail of the drone.

2. Installation of the ribbon cable for the camera:

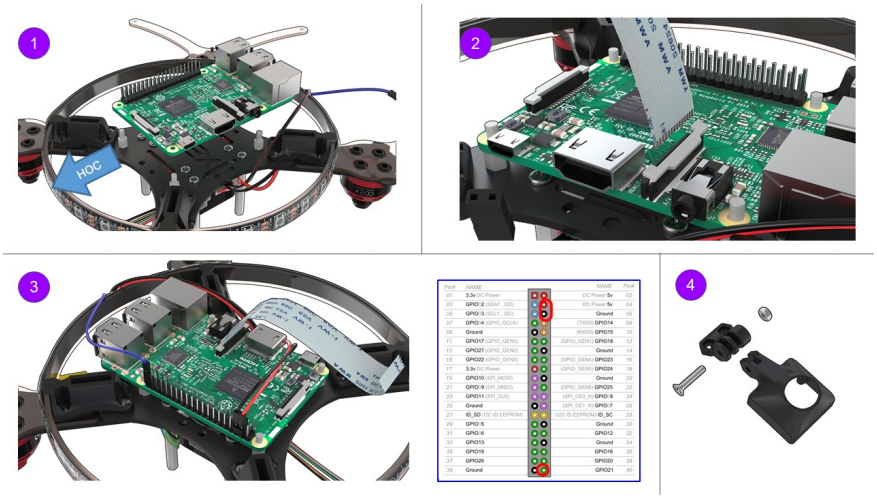
   - lift the latch;
   - connect the ribbon cable;
   - close the latch.

3. Connecting Raspberry to power supply:

```
5V -> pin 04 (DC power 5 V)
GND -> pin 06 (Ground)
Connecting the LED strip pin 40 (GPIO21)
```

4. Assembling the mount for the RPi camera.

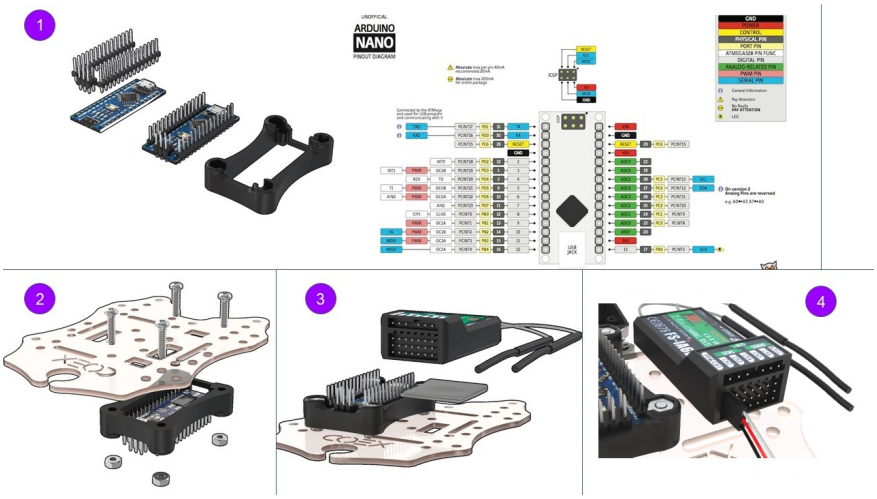
   Use an M3x16 screw and an M3 nut



Arduino and FlySky radio receiver installation

1. Solder Arduino Nano micro-controller pins to its board.
2. Install the micro-controller into a special mount, and attach to the lower deck using M3x16 screws (4 pcs).
3. Using double-sided tape, attach the receiver as shown in the picture.
4. Connect the ribbon cable from the radio receiver to Pixracer as shown in the picture.

```
white -> PPM
red -> 5V
black -> GND
orange, green -> currently not used. They are set to the unused pins of t
```

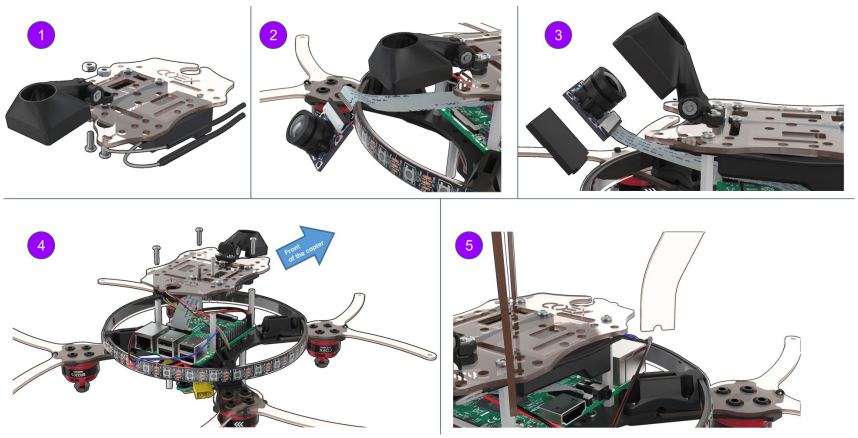


RPi camera installation

1. Attach the mount for the RPi camera assembly to the lower deck with M3x12 screws (2 pcs.)
2. Connect the ribbon cable to the RPi camera.
3. Install the camera into the mount, secure it with M2 self-tappers.
4. Attach Raspberry with 30 mm legs (4 pcs.).

   Attach the lower deck assembly to the rack with M3x8 screws (4 pcs.)

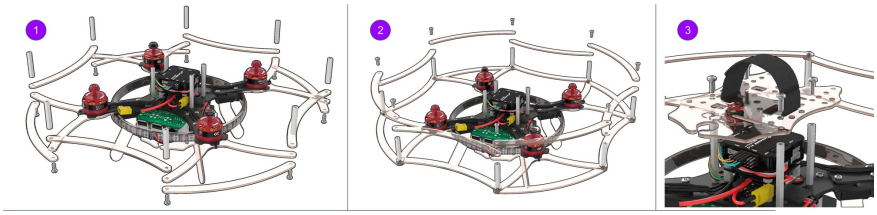
5. Install the legs into the mounts (4 pcs).



Installation of the remaining structural elements

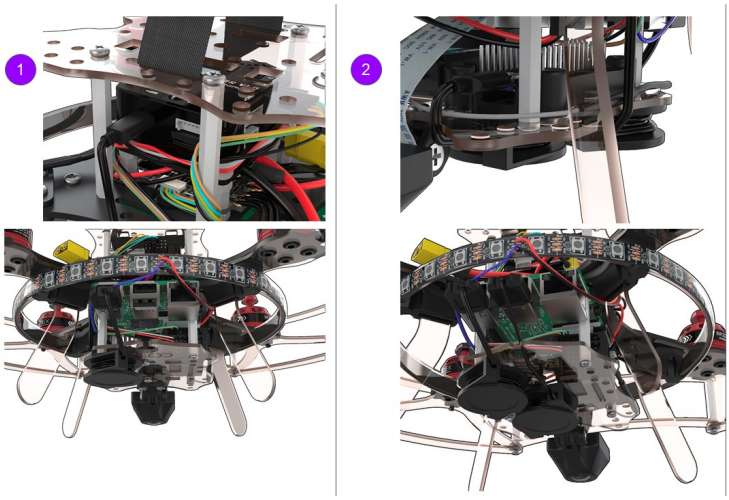
1. Install the bottom guard using M3x12 screws (8 units) and the 30 mm legs (8 pcs).
2. Install the top guard using M3x12 screws (8 pcs).
3. Insert the strap into the upper deck for attaching the battery.

   Secure the upper deck with M3x8 screws (4 pcs.)



USB connectors installation

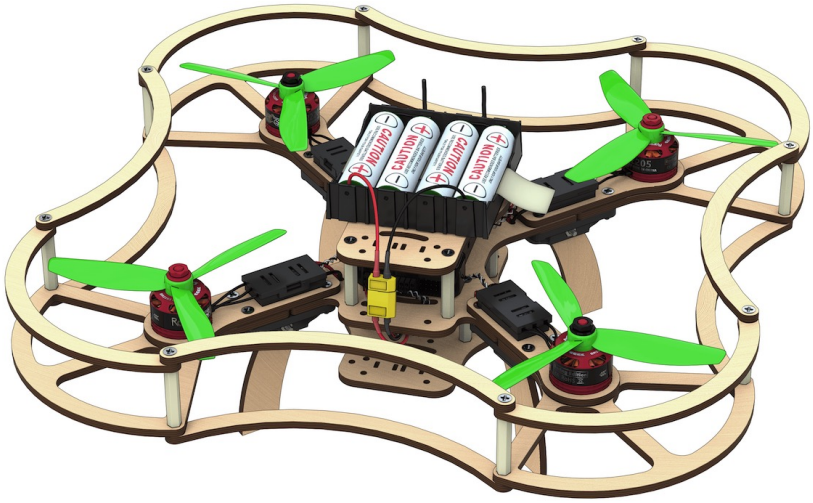
1. Connect Pixracer to Raspberry using the micro USB - USB cable.
2. Connect Arduino to Raspberry using the micro USB - USB cable.



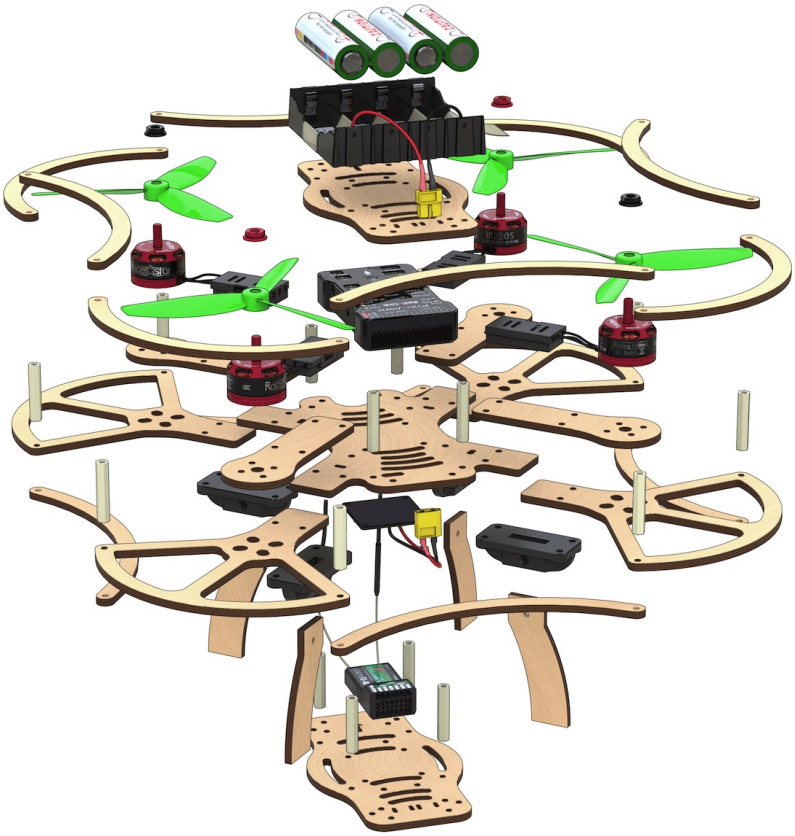
.

Read more about connection in article.

Clover 2 construction kit assembly instruction



The constructor kit contents



- Central frame ×2.
- Additional frame ×4.

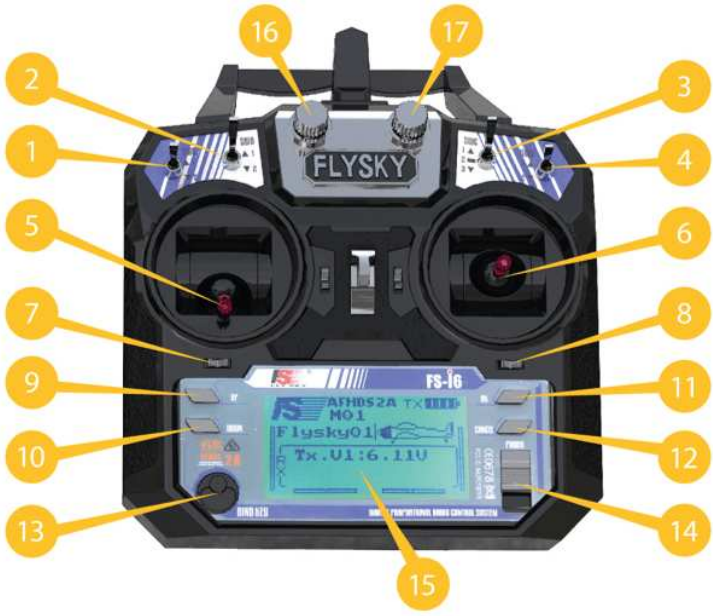
- Motor mount ×8.
- Legs x8.
- Motor mount guard ×8.
- Propeller guard ×16.
- Side guard ×16.
- Dalprop 5045 plastic propeller ×4.
- Racerstar BR2205 2300kV brushless motor ×4.
- Speed controllers ESC, DYS XSD20A ×4.
- Power controller XT60 pin ×1.
- Power connector XT60 socket ×1.
- Three-wire female-female flat cable ×2.
- Multicore silicone insulated copper wire 14AWG (red, black), 50 cm long
- Power distribution board PDB BeeRotor Power Distribution Board V2.0 ×1.
- Li-ion rechargeable battery (battery) 18650 ×8.
- EFEST Luc V4 Li-Ion Charger ×1.
- Protective case for regulators ×4.
- Legs attachment ×8.
- Pixhawk flight controller ×1.
- FlySky i6 radio receiver ×1.
- FlySky i6 radio transmitter ×1.
- EFEST LUC V4 Charger ×1.
- Micro USB to USB Cable ×1
- Battery compartment 18650 Li-ion ×1
- Wire copper multicore silicone insulated cable 18AWG (red, black), 100 cm long
- AA battery ×4
- Jumper, Bind-plug

Fasteners

- 6 mm plastic legs ×28.
- 30 mm plastic legs ×32.
- M3x8 screws ×48.
- M3x12 screws ×24.
- M3x16 screws ×40.
- Plastic nuts ×8.
- Metal nuts ×48.
- Stickers for the battery compartment ×8.
- Thermal contraction tube ⌀15, .50 cm
- Thermal contraction tube ⌀5, 100 cm
- Double-sided 3M adhesive tape ×16.
- Screwdriver ×1 (visualization needed)
- Insulation tape ×1
- Scissors ×1
- Strap for the battery 250 mm ×1

Flysky i6 transmitter

1. Switch A (SwA).
2. Switch B (SwB).
3. Switch (SwC).
4. Switch D (SwD).
5. Left stick.
6. Right stick.
7. Left trimmer.
8. Right trimmer.
9. Up button.
10. Down button.
11. OK button.
12. Cancel button.
13. BIND KEY button.
14. POWER switch.
15. LCD.
16. Handle A (VrA).
17. Handle B (VrB).

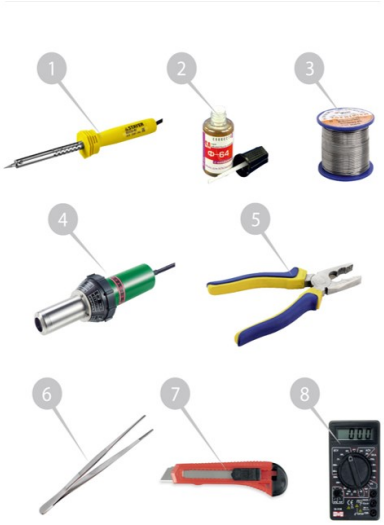


Additional equipment

This equipment is not part of the Clover 2 constructor kit, but it is required for the assembly process

1. Soldering iron
2. Colophony/ Flux (neutral)
3. Solder
4. Hot air gun
5. Pliers
6. Pincers
7. Stationery knife

8. Multimeter

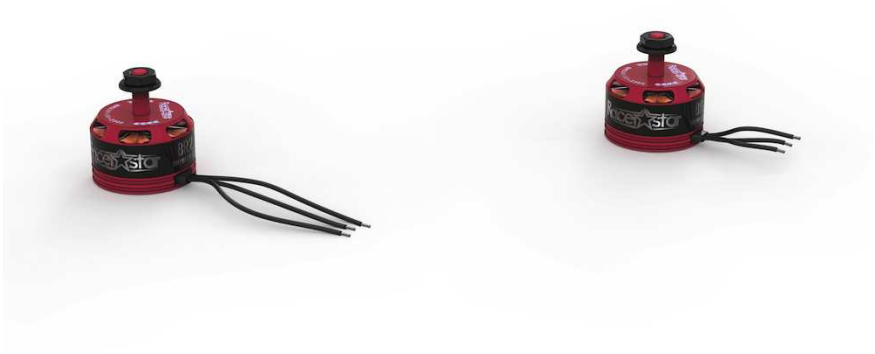


Soldering safety

Assembly order

Installation of motors

- Unpack the motors. Using pliers, shorten the wires on the motors by cutting half their length (leaving about 25 mm).



Strip

- remove 2 mm of insulation from the ends of the wire without damaging the copper strands.

Twist the wires.

Tin wires

- Apply flux to the exposed part of the wire.
- Cover the solder using tweezers.

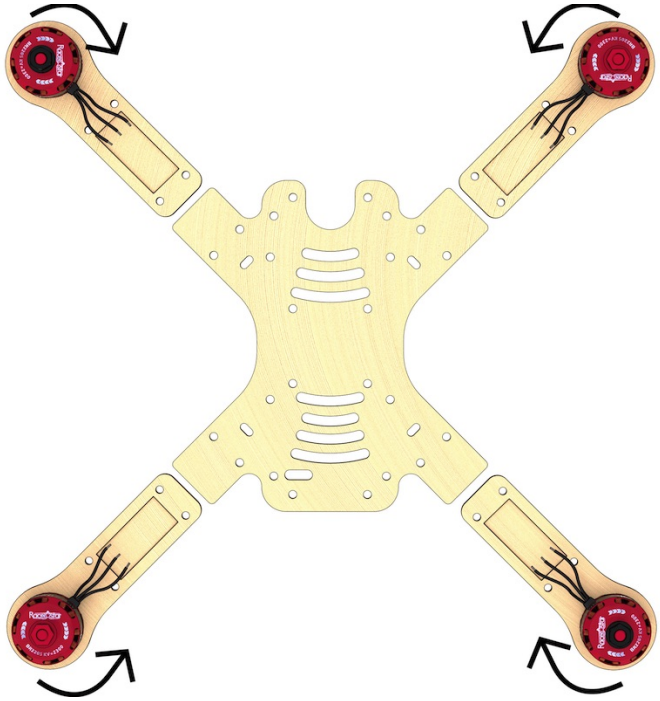


Fix the motor on the mount

- Install the motor on the engraved side of the mount.
- Attach the motors to the mounts with M3x8 screws using a screwdriver.



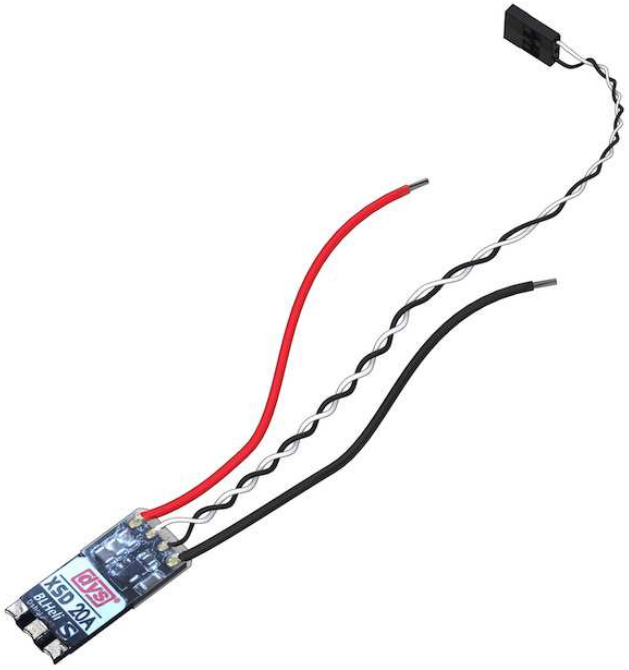
- Mounts with motors should be arranged according to the diagram. The arrows indicate the direction of motor rotation direction.



Tin three contact pads of the speed controller

- Apply flux
- Apply solder

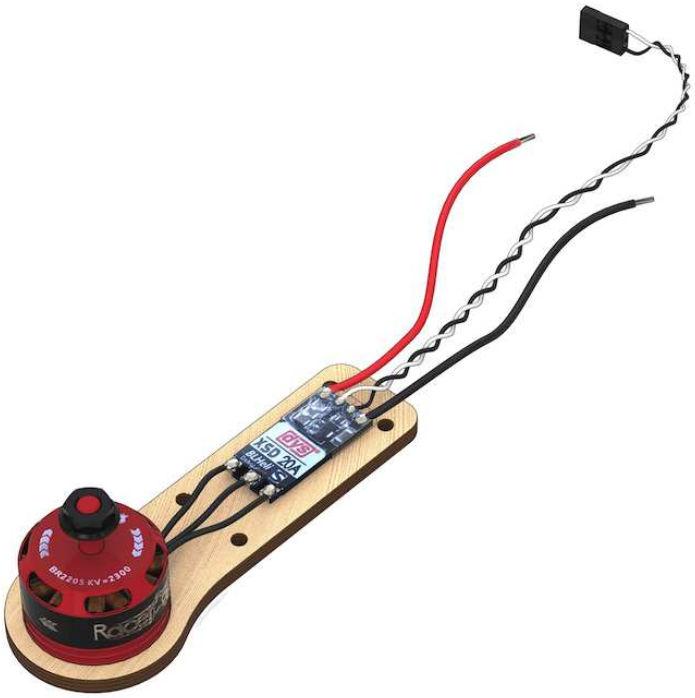
By warming up the contact pads of the controller, the tin will evenly fill the entire pad. To do so, apply heat by holding the soldering iron on the contact pads for 2 sencods (or more if needed).



- Repeat this operation for the remaining three ESC

Solder the wires of the motors to the ESC

Solder the prepared wires of the motors to the pads of the controllers.

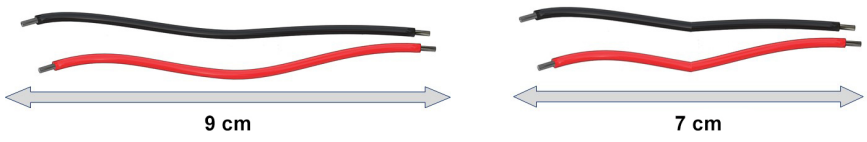


- Repeat this operation for the remaining three ESC

Power connectors installation

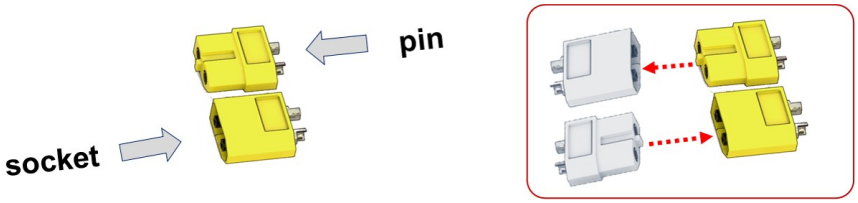
Preparing wires for XT60 power connectors

1. Take a bundle of red and black wires marked 14AWG
2. Cut 4 pieces of wire of the following lengths
3. Length 7 cm (XT60 pin power connector) - 1 red, 1 black
4. Length 9 cm (XT60 socket power connector) - 1 red, 1 black

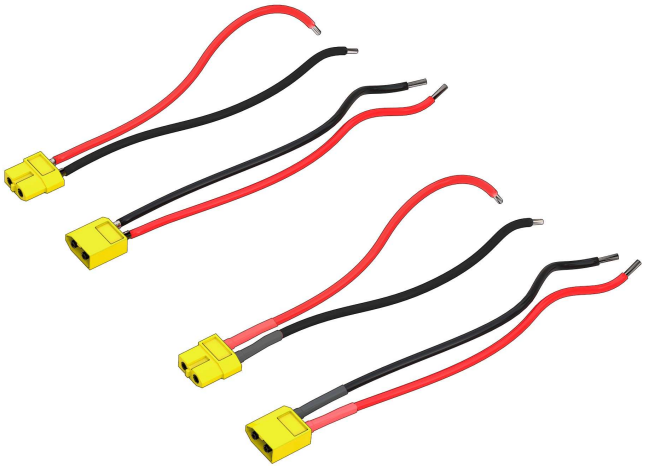


Preparing XT60 pin and XT60 socket high-power connectors

Article about high-power connectors and their designations



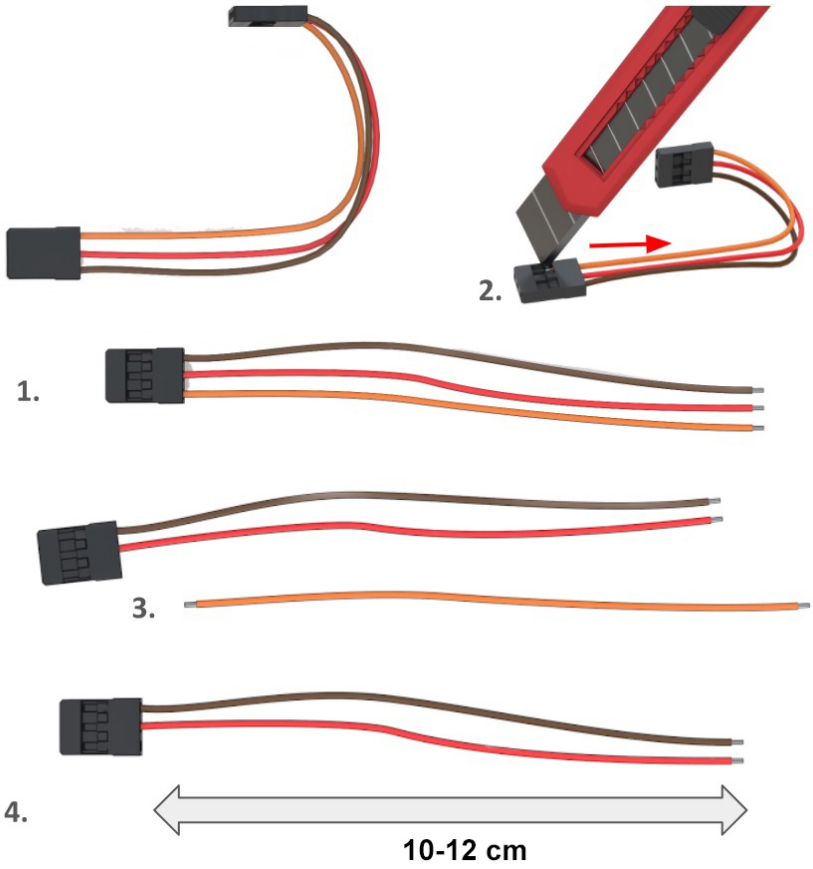
1. Tin two red and black 14AWG 7 cm long power wires for the XT60 pin connector.
2. Tin contact pads of the XT60 pin connector.
3. Solder the black wire to the “-” contact of the connector.
4. Solder the red wire to the “+” contact of the connector.
5. Cut ⌀5 heat-shrink tubing (2 sections × 10 mm).
6. Slip the ⌀5 heat-shrink tubing tube on the wires so that they cover the contact pads of the wires from XT60.
7. Shrink the heat-shrink tubing with a hot air gun.



8. Repeat the procedure for XT60 socket connector.

Preparation of the 5V power connectors for the control circuit

1. Trim/pull out all pins from one of the connectors. Disconnect it.
2. Using an utility knife, pry the retainer off on the remaining connector to release the 3rd wire.
3. Remove the 3rd (orange) wire from the connector, since it is not needed.
4. The length of the remaining black and red wires should be of 10 – 12 cm.

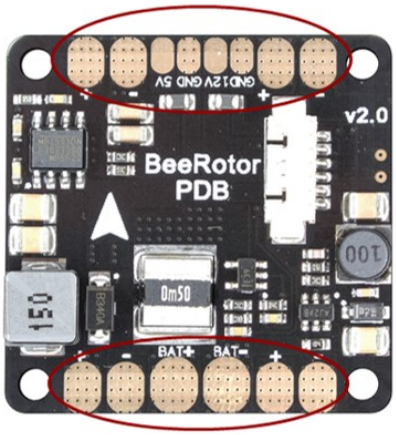


Installation of the power distribution board

Pre-soldering check

Article about continuity test





Check OPEN CONDITION of the following circuits (the multimeter does not beep):

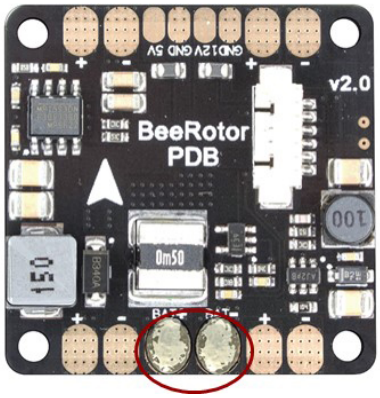
- “BAT+” and “BAT-”
- “12V” and “GND”
- “5V” and “GND”

Check CLOSED CONDITION of the following circuits (the multimeter beeps):

- “BAT-” with every contact marked “-” and “GND”
- “BAT+”, with every contact marked “+”

Tin the contact pads of the power board

1. Tin* the contact pads of the power board.
2. Using a multimeter, check absence of short-circuits on the PCB (check continuity).

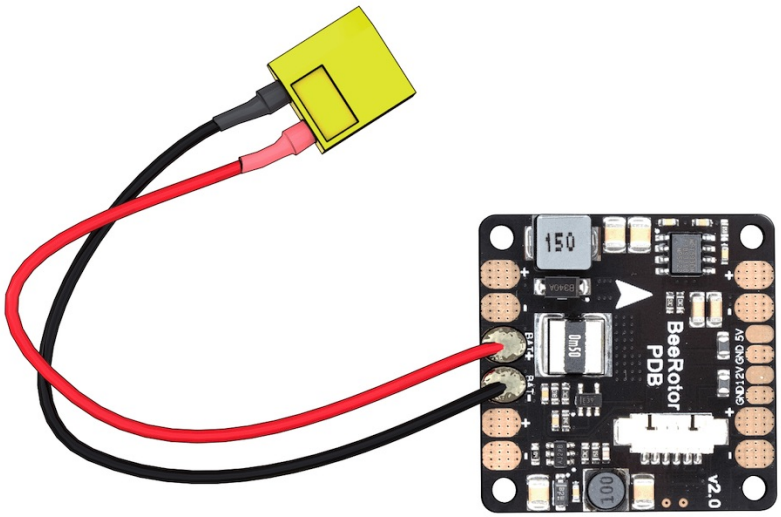


1. Blanch contact pads BAT+ and BAT-
2. Blanch other contact pads

By warming up the contact pads of the controller, the tin will evenly fill the entire pad. To do so, apply heat by holding the soldering iron on the contact pads for 2 sencods (or more if needed).

Soldering the XT60 high power connector

Solder the connector for battery, taking into account the polarity on the contact pads.

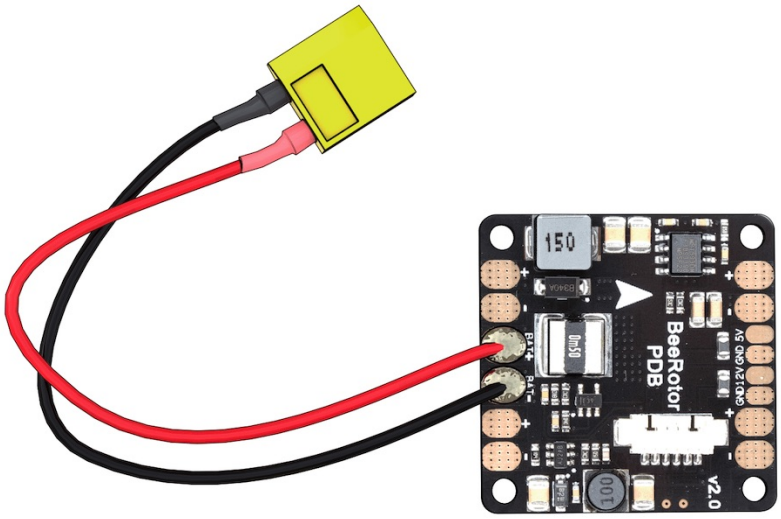


IMPORTANT NOTE about polarity

- the red wire is “+”
- the black wire is “-”

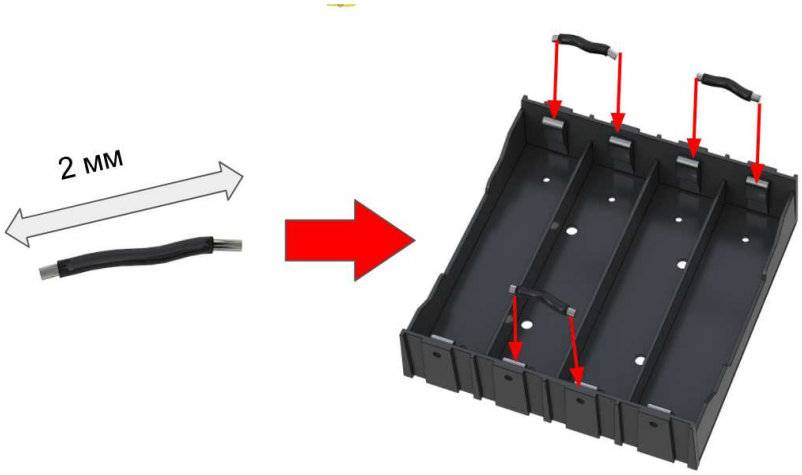
Soldering of the power connector for the 5V control circuit

Solder the 5V connector, taking into account the polarity on the contact pads. (in the picture: the red wire is “+”)



Installation of the battery compartment

Preparation of jumpers (3 pcs.)



- Cut off 2 cm of high-power wire.
- Strip on both ends.
- Tin.
- Make 3 jumpers.
- Solder the jumpers according to the diagram.
- Check for continuity with a multimeter. If necessary, clean with sand paper.

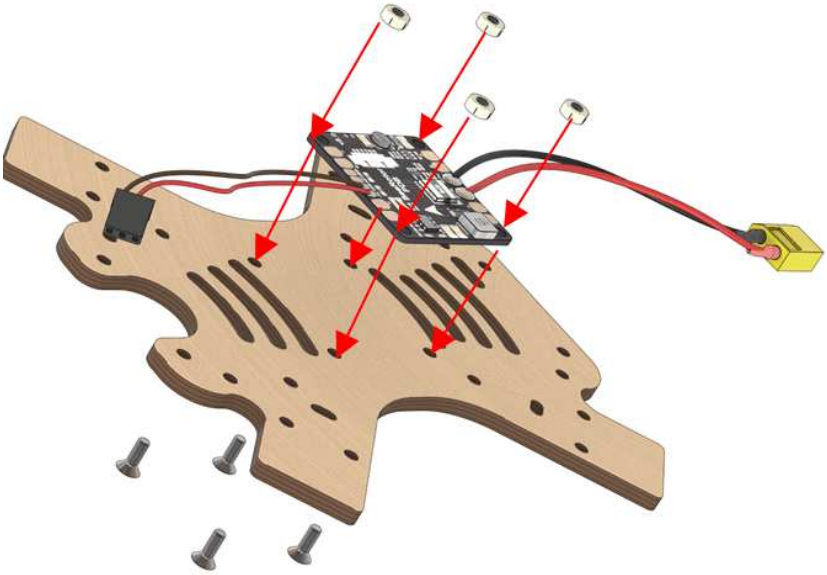
Preparation of the battery compartment



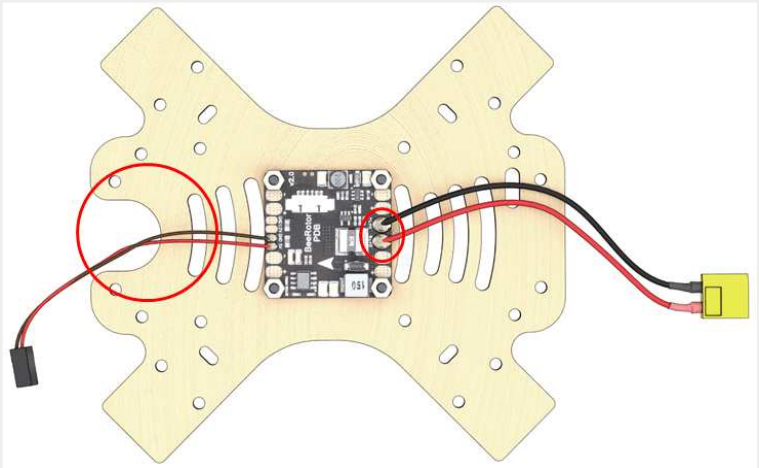
- Conforming to the polarity, glue the sticker with markings inside the battery compartment.
- Stick a strip of adhesive tape to the bottom of the compartment.

Installation of the power distribution board

- Fix the power board to the frame with M3x8 screws and plastic nuts.



The white arrow on the BeeRotor board points towards the fore cutout.

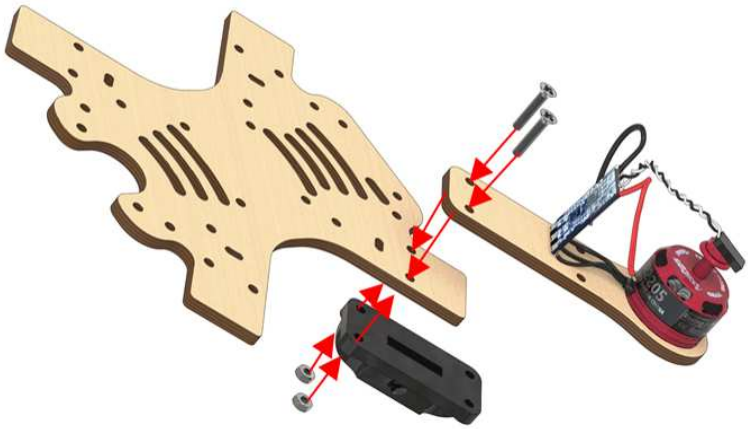


Installation of elements

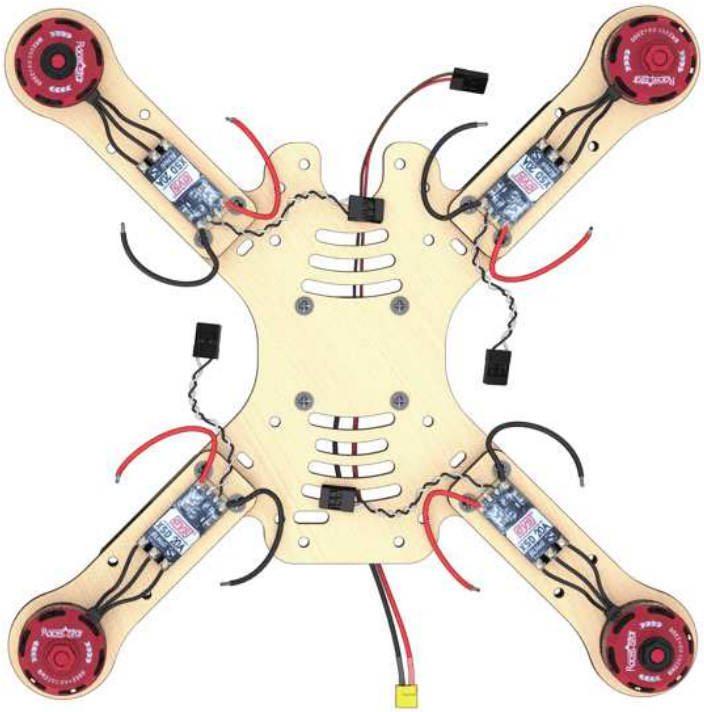
1. Install the nuts into plastic holders.



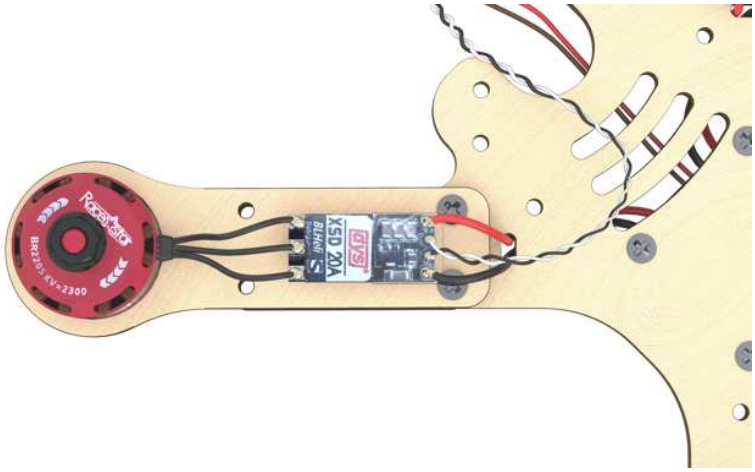
2. Fix the motor mounts to the frame with M3x16 screws.
    - The mounts are installed above the frame.
    - Plastic holders are installed beneath the frame.



3. Arrangement of motors. Check arrangement of the motors (the motors with black nuts should be in the top left and lower right corners).

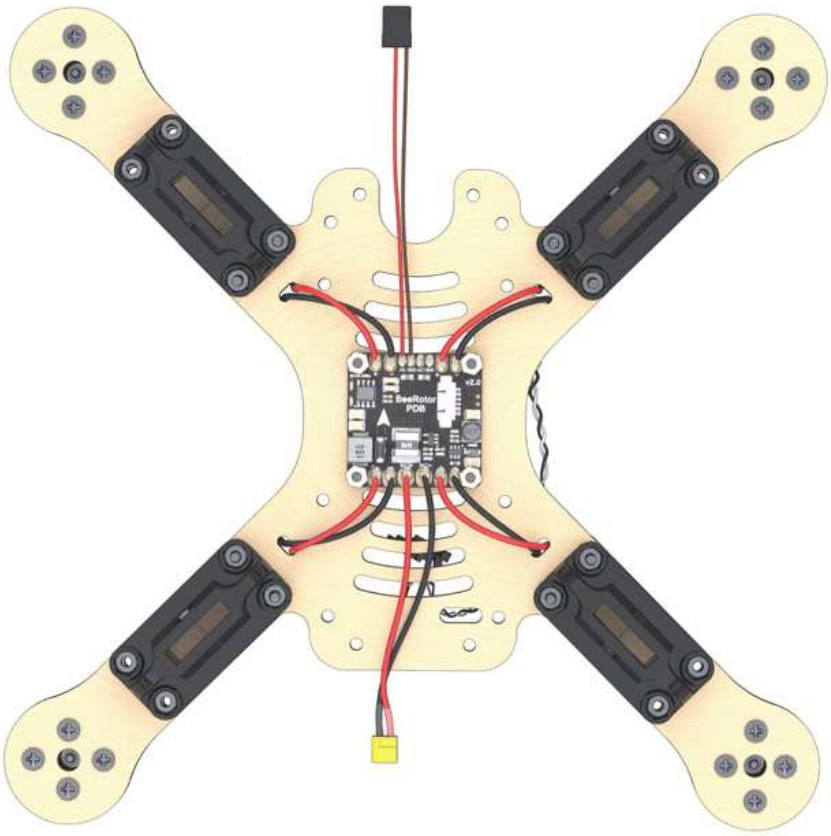


4. Put the power wires of the ESC through the holes.



Soldering the high-power circuit board

Solder the high-power wires of the ESC to the power supply board observing polarity.

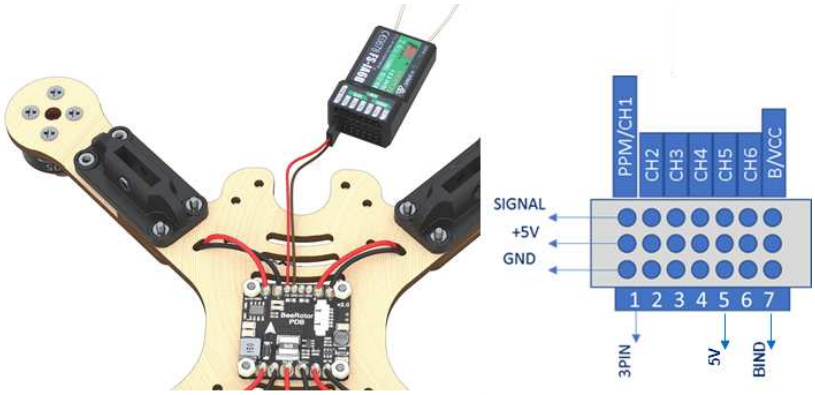


IMPORTANT NOTE about polarity

- the red wire is “+”
- the black wire is “-”

Pairing the receiver and transmitter

1. Connect the radio receiver to the 5V connector. In any connector the GND is in the bottom. In the diagram, the power is labeled 5V



2. Connect the battery. The LED on the radio receiver should be flashing. ! [Connecting the battery]

SAFETY when working with the battery

Safety when working with 18650 Li-ion batteries

- Handle batteries carefully. Avoid falls, bumps, and deformations.
- When connecting (disconnecting) batteries, hold only connectors, never pull on the wires.
- In case of broken connectors, damaged insulation or casing of the battery, don't touch it and immediately inform the instructor.



Enabling the transmitter

1. Insert the jumper into B/VCC of the radio receiver (short "ground" and "signal")
2. On the transmitter, hold down the BIND KEY button.
3. Power up the transmitter (flip the POWER switch, do not release BIND KEY).
4. Connect the battery to the drone.
5. Wait for synchronization.
6. Disconnect the jumper.
7. The LED will remain ON continuously.



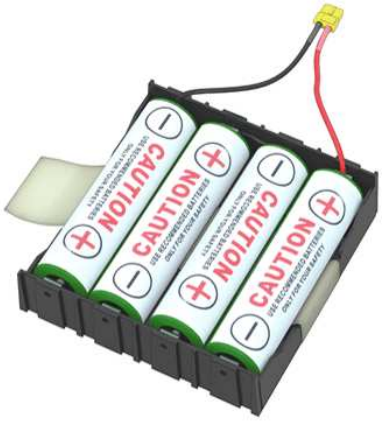
Radio equipment troubleshooting manual

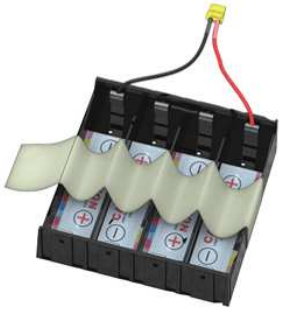
Checking the motors direction of rotation

1. Apply stickers to the 18650 batteries.

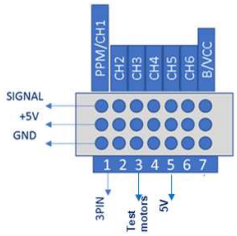
2. Install the 18650 batteries into the compartment observing polarity.

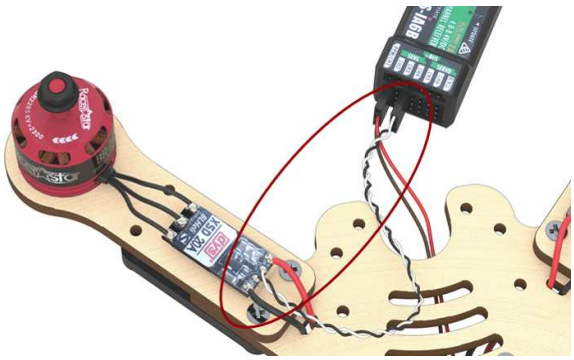






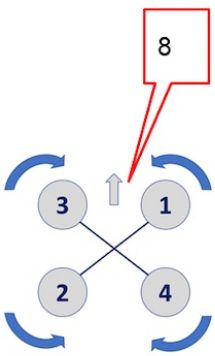
3. Check that the 5V power plug is connected to the receiver according to the circuit diagram.
4. Connect the motor ESC to channel 3, marked as CH3 on the receiver as on the circuit diagram.



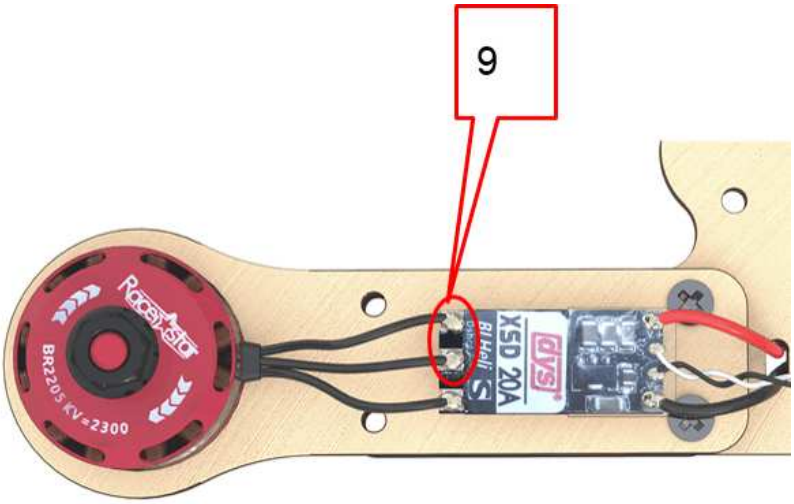


5. Connect external power (battery).
6. Turn the transmitter ON.
7. Using the left stick, set throttle to 10 %.
8. Check the motor direction of rotation according to the scheme.



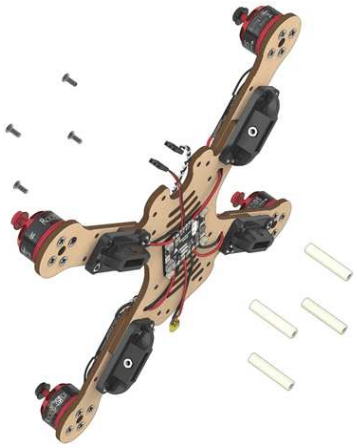


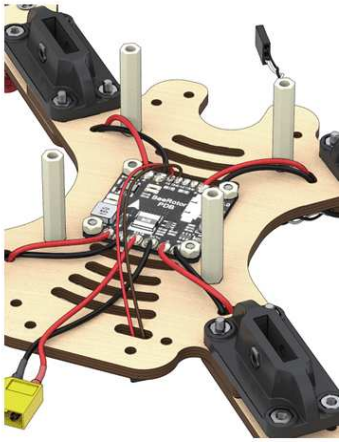
9. If you have to change the rotation direction, swap any two phase wires of the motor (needs resoldering).



Installation of the radio receiver

1. Install the 30 mm plastic legs on the frame with M3x8 screws.
2. Pass the 5V power connector through the slit.

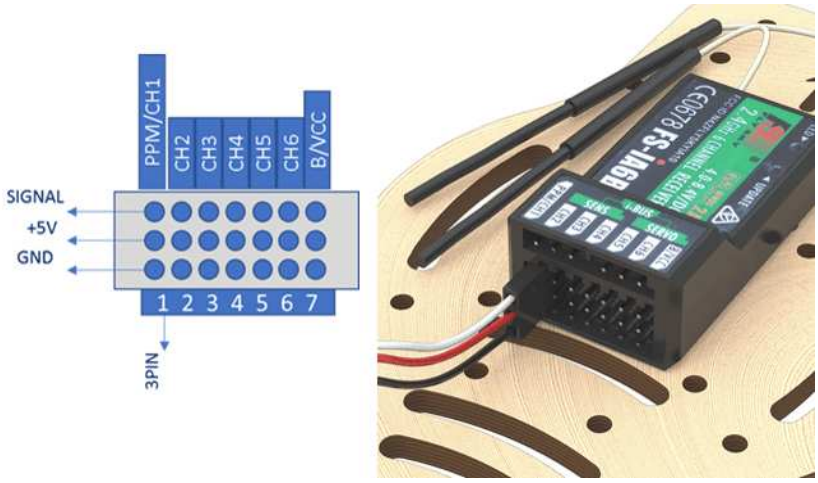




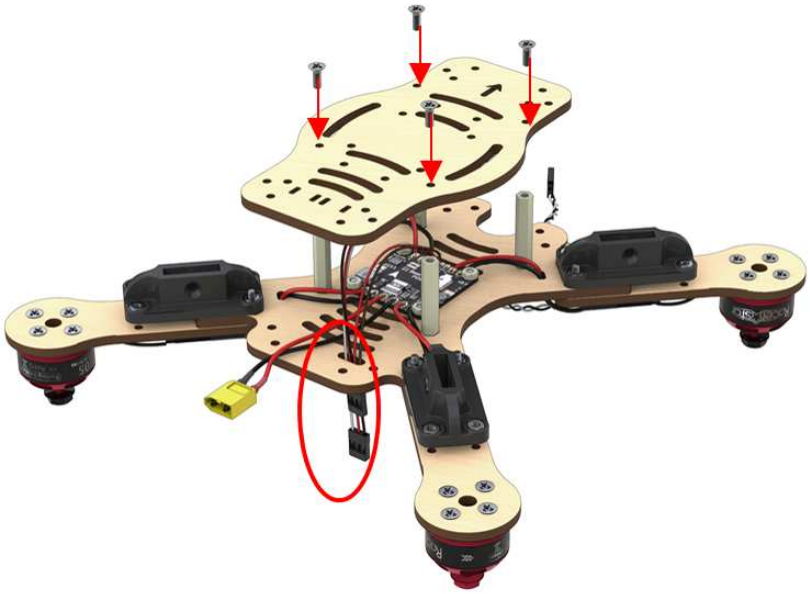
3. Attach the receiver to the bottom of the additional frame using double-sided adhesive tape and following the orientation of the engraved arrow. The antennas are to be pointing forward.



4. Install the 3-wire flat cable into the PPM / CH1 channel.



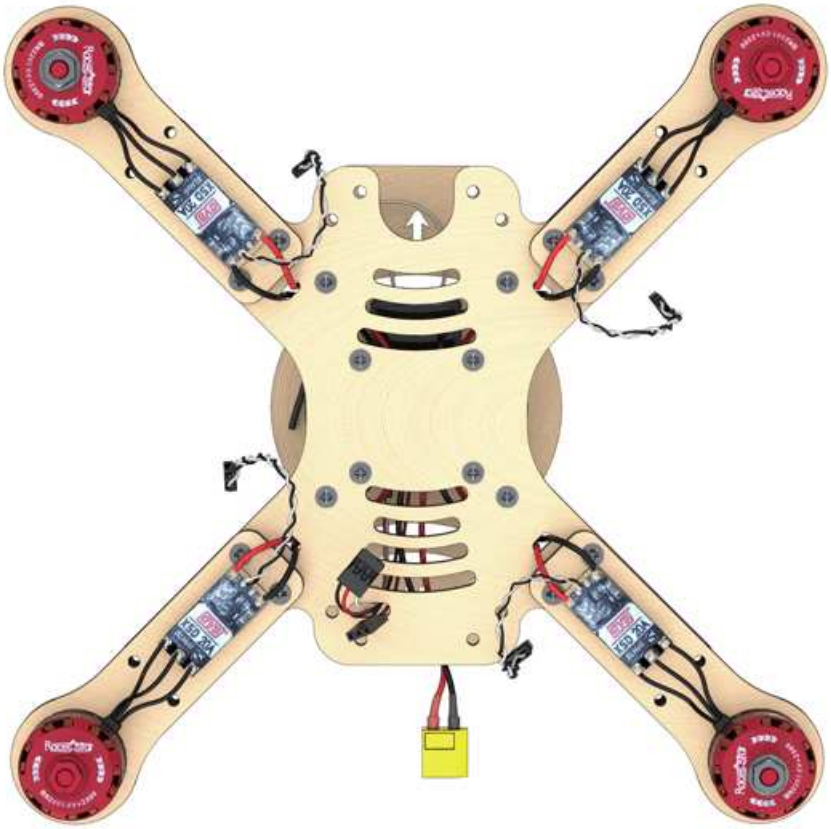
5. Pass them through the slit to the 5 V connector.
6. Screw the bottom an additional frame to the legs on the central frame with M3x8 screws.



The directions of the arrows on the power supply board and the additional frame should coincide

Installation of the flight controller

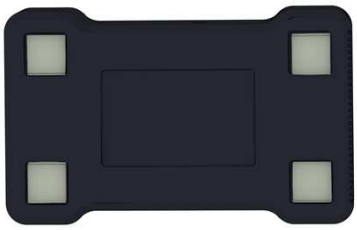
Turn the assembly upside down



Installation of the Pixhawk flight controller

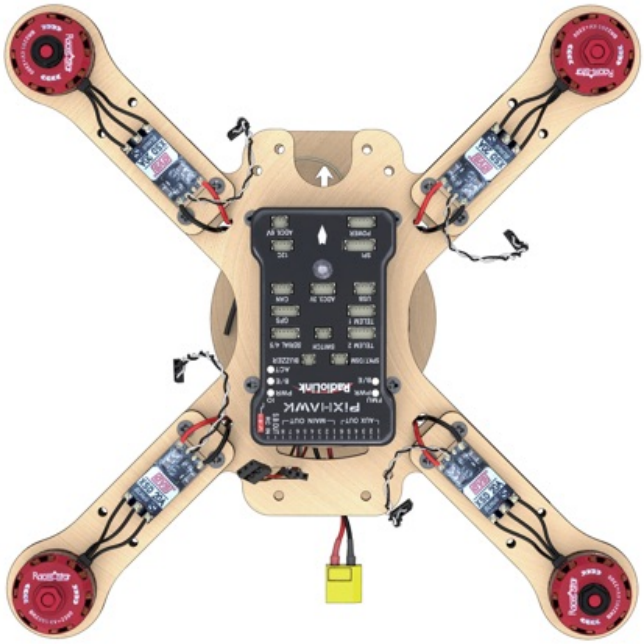
1. Stick the two-sided adhesive tape in the corners of the flight controller.





When the motors rotate, vibrations occur, which affect sensors of the Pixhawk flight controller. To avoid this effect, the number of double-sided tape layers should be increased up to 4 – 5.

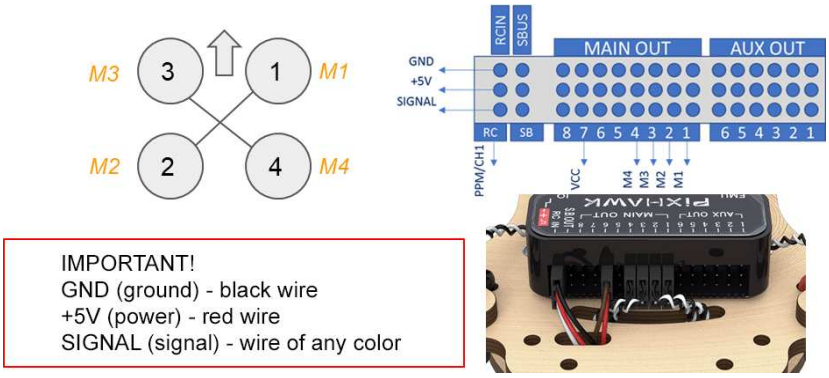
2. Install the flight controller in the center of the frame.



The arrows on the frame and Pixhawk should point in the same direction

Connecting the flight controller according to the circuit diagram

1. Connect PPM (three-wire flat cable) to the RCIN port
2. Motors to MAIN OUT ports 1,2,3,4, according to the circuit diagram
3. Power by PDB (5V/VCC) to any port except for SB (SBUS)



ESC assembly

1. Stick the double-sided adhesive tape to the base of the ESC protective case
   Adhesive tape on the ESC case
2. Put the ESCs into protective cases. Fasten the assembly to the motor mounts of the frame.

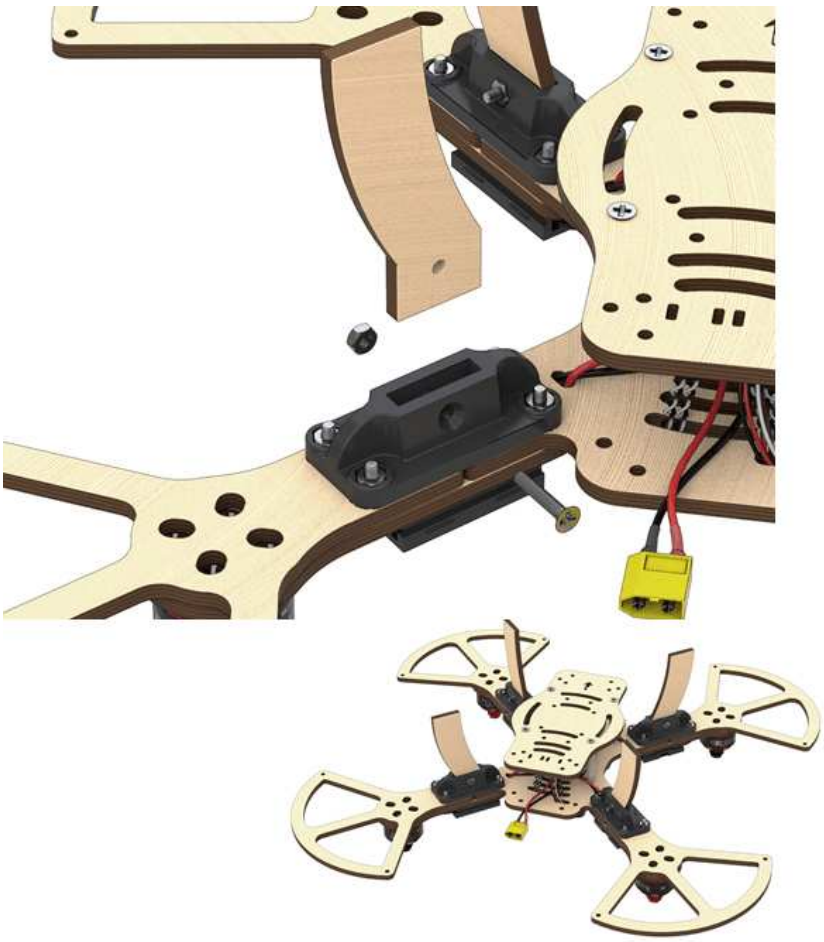


Installation of guard

1. Attach the lower guard with M3x16 screw to the motor mounts of the frame.



2. Attach the feet to the plastic holders with M3x16 screws.



3. Attach the 30 mm long legs to the holes of the lower guard with M3x12 screw.



4. Attach the top guard with M3x12 screws.

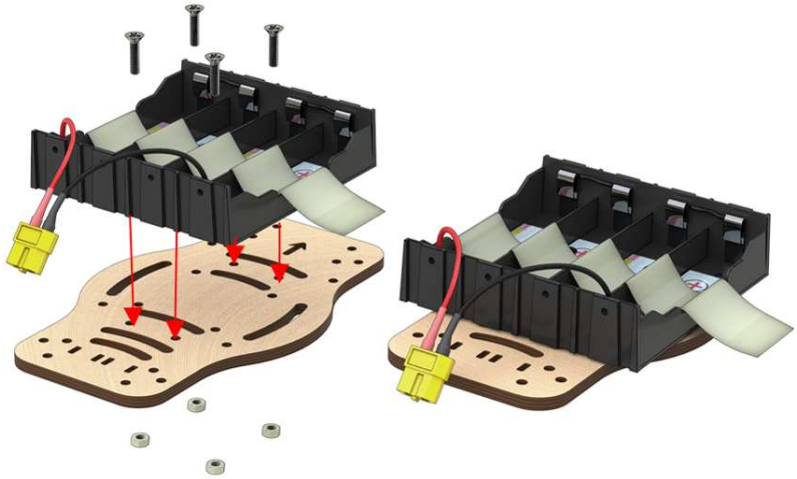


Installation of the battery compartment

Requires the following components:

- M3x12 screws (4 pcs)
- M3 nuts (4 pcs)
- Additional frame (1 pc)
- Battery compartment (1 pc)

- Attach the battery compartment on top of the additional frame with M3x12 screws and nuts.



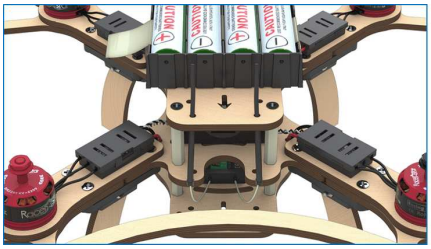
- Attach the top additional frame to the legs with M3x8 screws.

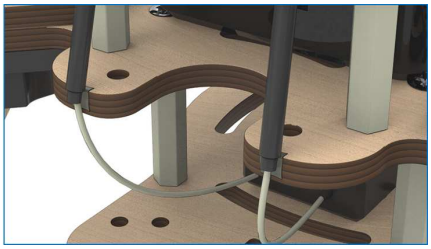


- Install the battery into the battery compartment.

Installation of antennas

1. Attach antennas on double-sided adhesive tape or duct tape, and put the antennae into the front holes of the top additional frame.

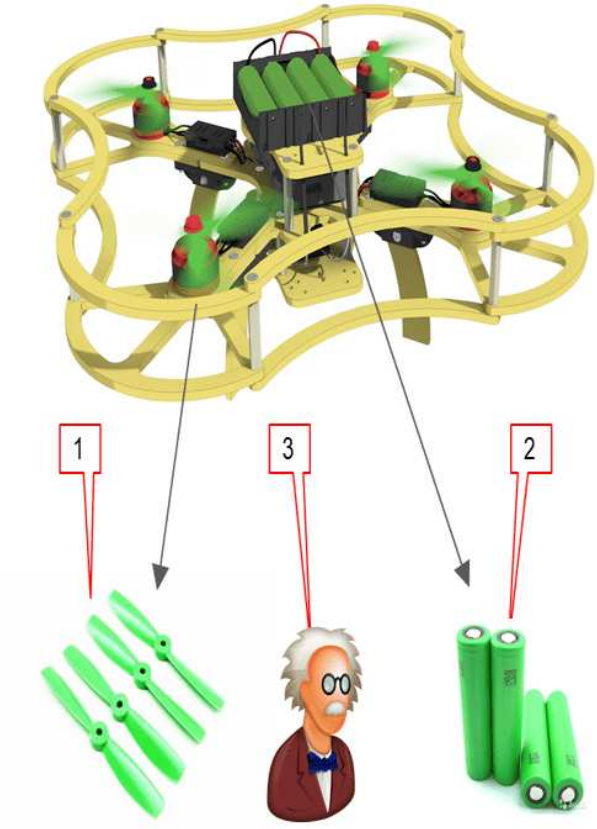




The drone is ready for configuration!

Safety notes for assembly and configuration

1. Remove the propellers.“All ground operations are to be performed with propellers removed. Propellers are to be installed on the motors before the flight only.”
2. Disconnect the battery. Keep the power off. “Assembly, configuration, and maintenance should be performed with power disconnected. Connect power only for testing electronic components of the drone. After testing, power is to be disconnected before other works.”
3. Call for help. “If you experience problem when working with the drone, contact the instructor or the teacher, do not try to solve the problem yourself.”



Security when working with 18650 Li-ion batteries

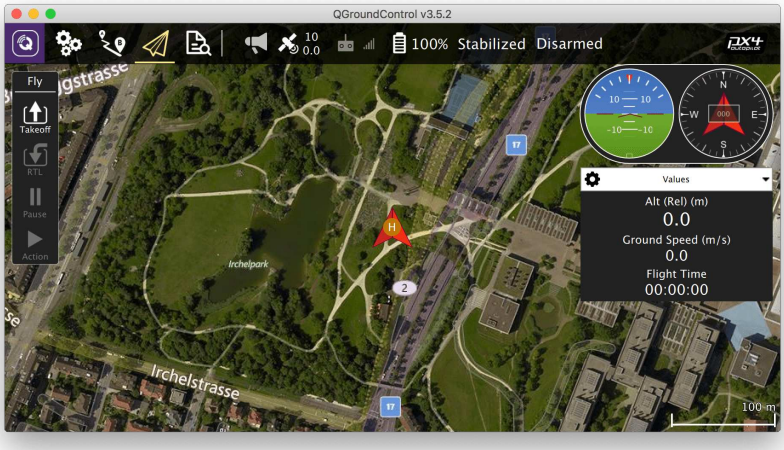
1. Handle batteries carefully. Avoid falls, bumps, and deformations.
2. When connecting (disconnecting) batteries, hold only the connectors, never pull or tug the wires.
3. If you see open connectors, violation of insulation or battery compartment integrity, do not touch it, and immediately inform the instructor or teacher.

See article safety precautions when soldering and during drone flight operation

Initial setup

Installing QGroundControl

QGroundControl is a software package that can be used to flash, configure and calibrate the flight controller.



Download and install the version for your operating system from the official QGroundControl website. When asked, agree to install additional drivers.

Consult the official QGroundControl user guide if anything goes wrong.

Preparing the MicroSD card

Prepare the MicroSD card for your flight controller.



- Put the card into your computer (use an adapter if necessary).

- Format the card to FAT32 filesystem. Right click on the SD card icon in Windows Explorer and select "Format". Use the Disk Utility in macOS.
- Use "Safely Remove Hardware" and unplug the card.
- Put the card into your flight controller.

Flashing the flight controller

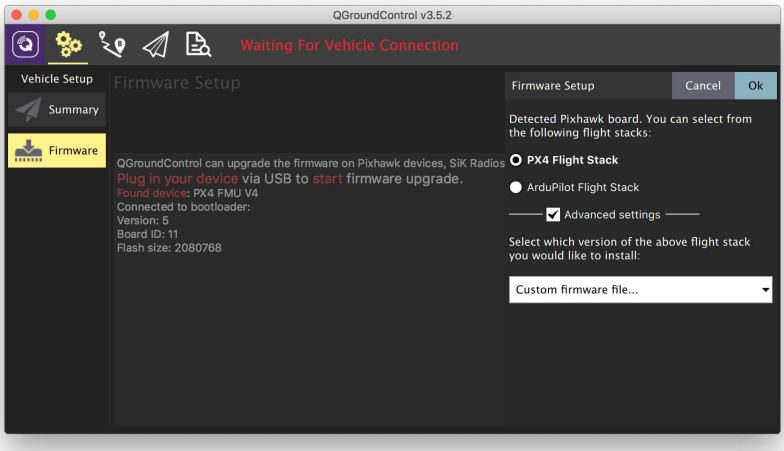
Main article: https://docs.qgroundcontrol.com/en/SetupView/Firmware.html

> Do not connect your flight controller prior to flashing.

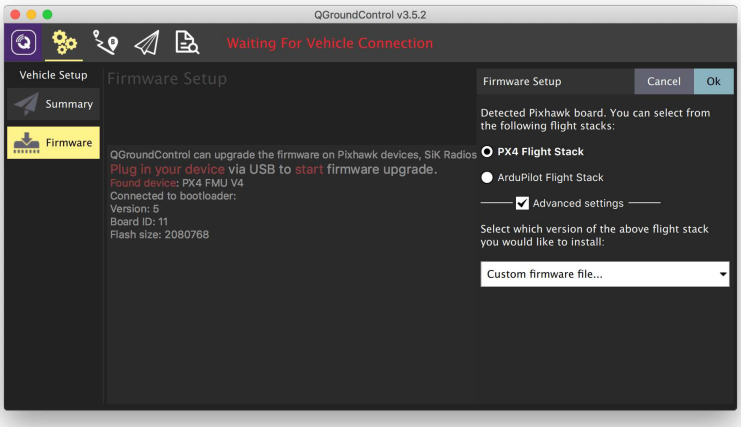
We recommend using the modified version of PX4 with COEX patches for the Clover drone, especially for autonomous flights. Download the latest stable version **from our GitHub**.

To use all the most recent PX4 functions you also can use the latest official firmware version (experimentally).

Flash the flight controller with this firmware:



1. Disconnect the flight controller from computer (if connected).
2. Launch QGroundControl software.
3. Go to *Vehicle Setup* panel (click on the QGroundControl logo in the top-left corner) and select *Firmware* menu.
4. Connect your flight controller to your PC over USB.
5. Select *PX4 Flight Stack* in the right bar appeared.



6. To use **COEX patched firmware**:
    - Check *Advanced Settings* checkbox.
    - Select *Custom firmware file...* from the dropdown list.
    - Press *OK* and select the file that you've downloaded.

    To use the latest **official stable firmware** just press *OK*.

Wait for QGroundControl to finish flashing the flight controller.

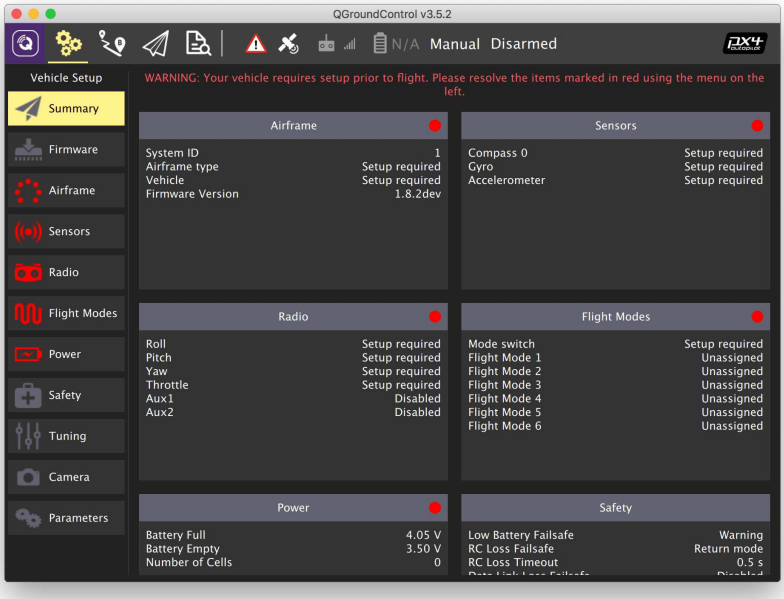
> Do not unplug the flight controller during the flashing process.

Read more about the firmware in the "Pixhawk Firmware" article.

Configuring the flight controller

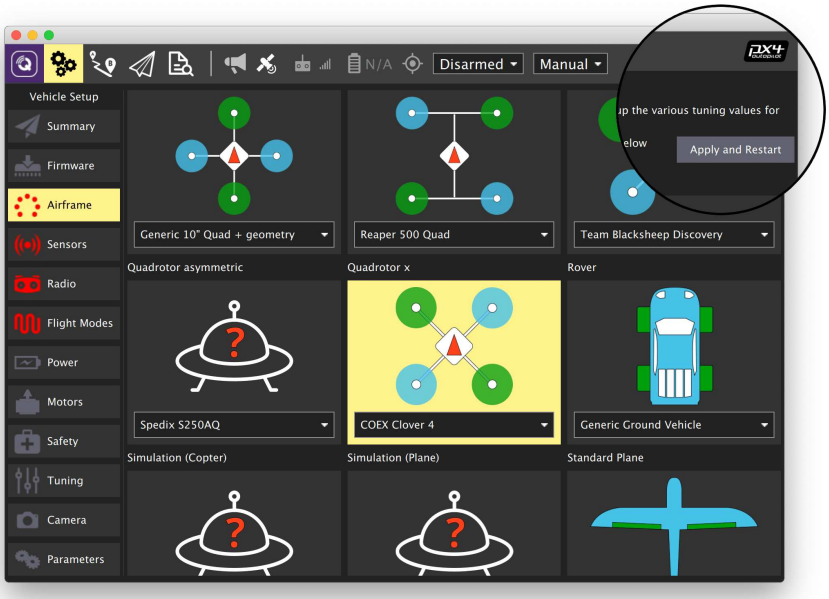
> You can use the Raspberry Pi to access the flight controller over Wi-Fi for the rest of the setup process.

This is how the main QGroundControl settings window will look like:



1. Parameters that require setup: *Airframe*, *Radio*, *Sensors*, *Flight Modes*.
2. Current firmware version.
3. Current flight mode.
4. Error messages.

Selecting the airframe

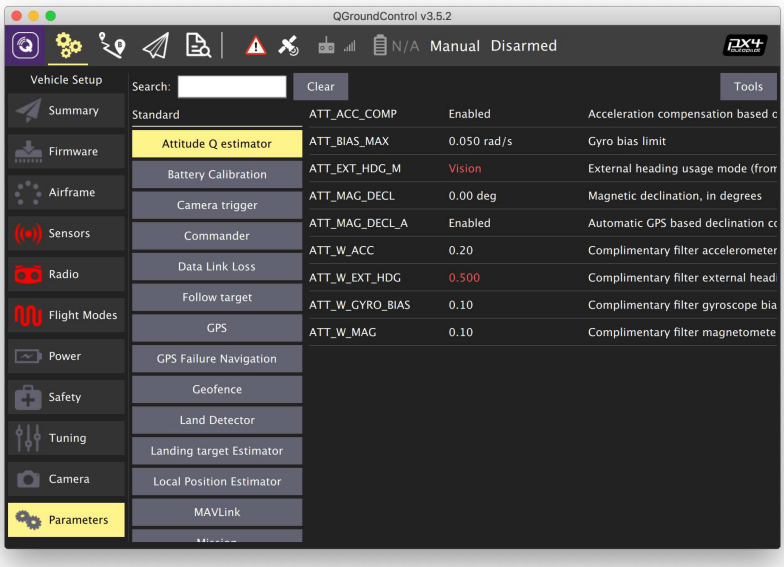


1. Open the *Vehicle Setup* tab.
2. Select the *Airframe* menu.
3. Select the *Quadrotor X* airframe type.
4. For Clover 4 select *COEX Clover 4* from the dropdown menu. Otherwise select *Generic Quadrotor X*.

5. Return to the top of the list and press *Apply and Restart* button, confirm by pressing *Apply*.
6. Wait for the settings to be applied and for the flight controller to restart.

Setting parameters

Open the *Vehicle Setup* tab and select the *Parameters* menu. You can use the *Search* field to find parameters by name. Recommended parameters values are given in the further documentation and also in the parameters summary article.



Press the *Save* button to save the changed value to the flight controller. Changing some parameters require rebooting the flight controller. You can do that by pressing the *Tools* button and selecting the *Reboot vehicle* option.

Configuring PID regulators

> Selecting *COEX Clover 4* frame subtype doesn't require setting PID coefficients.

Averaged PID coefficients for the Clover 4 drone

- `MC_PITCHRATE_P` = 0.087
- `MC_PITCHRATE_I` = 0.037
- `MC_PITCHRATE_D` = 0.0044
- `MC_PITCH_P` = 8.5
- `MC_ROLLRATE_P` = 0.087
- `MC_ROLLRATE_I` = 0.037
- `MC_ROLLRATE_D` = 0.0044
- `MC_ROLL_P` = 8.5
- `MPC_XY_VEL_P` = 0.11
- `MPC_XY_VEL_D` = 0.013
- `MPC_XY_P` = 1.1

- `MPC_Z_VEL_P` = 0.24
- `MPC_Z_P` = 1.2

Averaged PID coefficients for the Clover 3 drone

- `MC_PITCHRATE_P` = 0.145
- `MC_PITCHRATE_I` = 0.050
- `MC_PITCHRATE_D` = 0.0025
- `MC_ROLLRATE_P` = 0.145
- `MC_ROLLRATE_I` = 0.050
- `MC_ROLLRATE_D` = 0.0025

> Note that you should fine-tune the PID parameters for each drone individually. See the PID tuning guide.

Circuit breaker parameters

1. Set `CBRK_USB_CHK` to 197848 to allow flights with the USB cable connected.
2. Disable safety switch check: `CBRK_IO_SAFETY` = 22027.

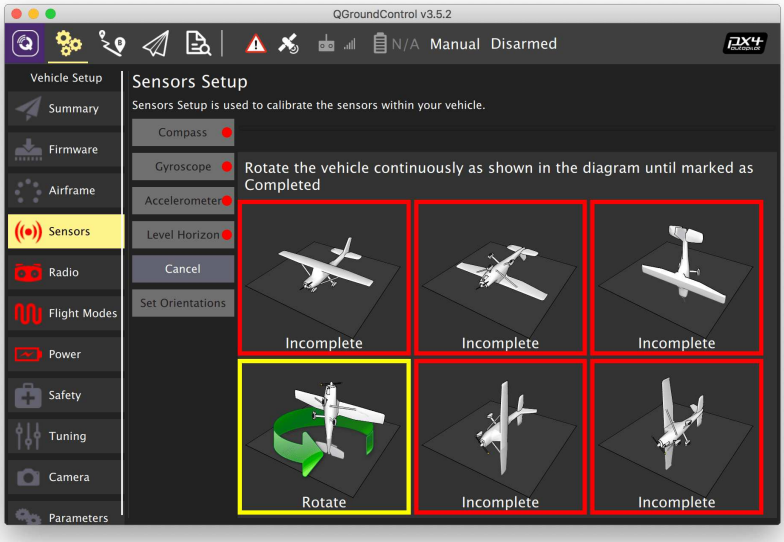
Next: Sensor calibration.

Sensor calibration

In order to perform the sensor calibration, select the *Vehicle Setup* tab and choose the *Sensors* menu.

> If you use the flight controller *COEX Pix*, and it's installed with servo pins faced backwards, all *Autopilot Orientation* columns must specify `ROTATION_ROLL_180_YAW_90`, otherwise the flight controller will not correctly perceive the tilt and rotation of the copter.

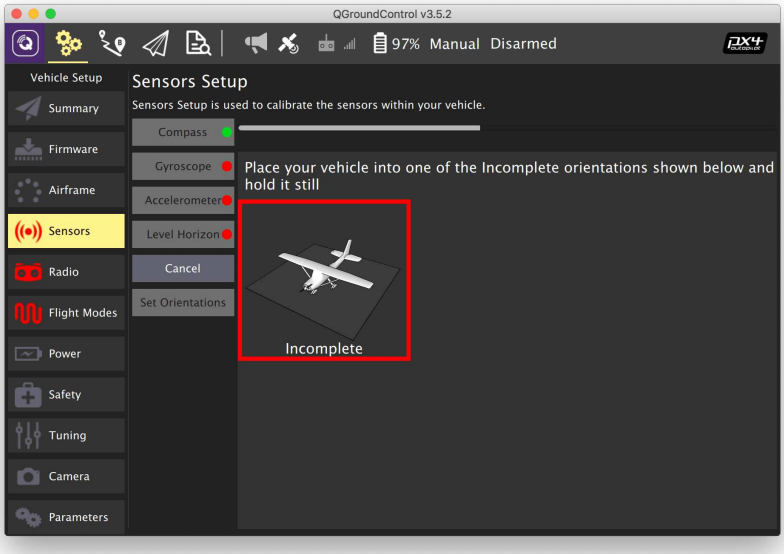
Compass



1. Select the *Compass* submenu
2. Choose the flight controller orientation (*ROTATION_NONE* if the arrow on the flight controller is aligned with the arrow on the frame).
3. Press *OK*.
4. Put the drone in one of the orientations marked by the red outline and wait for the appropriate outline to turn yellow.
5. Spin the drone as required until the outline turns green. Do this for all orientations.

Read more in the PX4 docs: https://docs.px4.io/master/en/config/compass.html.

Gyroscope

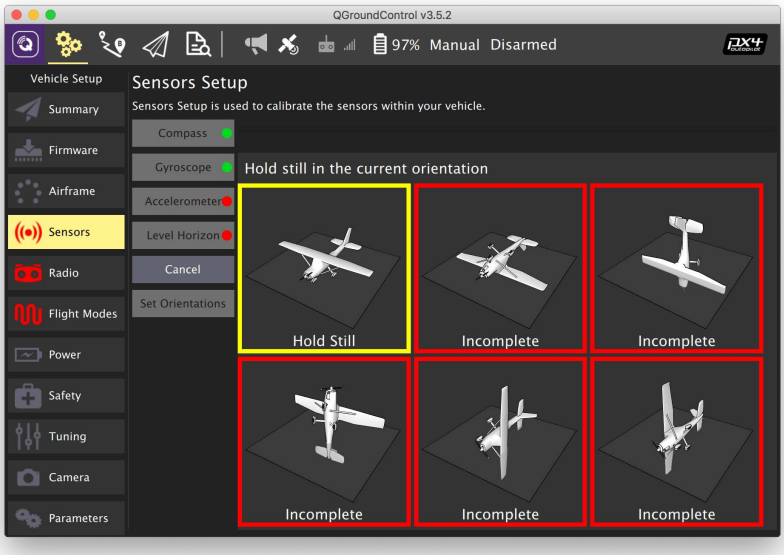


1. Select the *Gyroscope* submenu.
2. Place the drone on a flat, horizontal surface.
3. Press *OK*.
4. Wait for the calibration to finish.s

> The drone should stay completely still during the calibration.

Read more in the PX4 docs: https://docs.px4.io/master/en/config/gyroscope.html.

Accelerometer

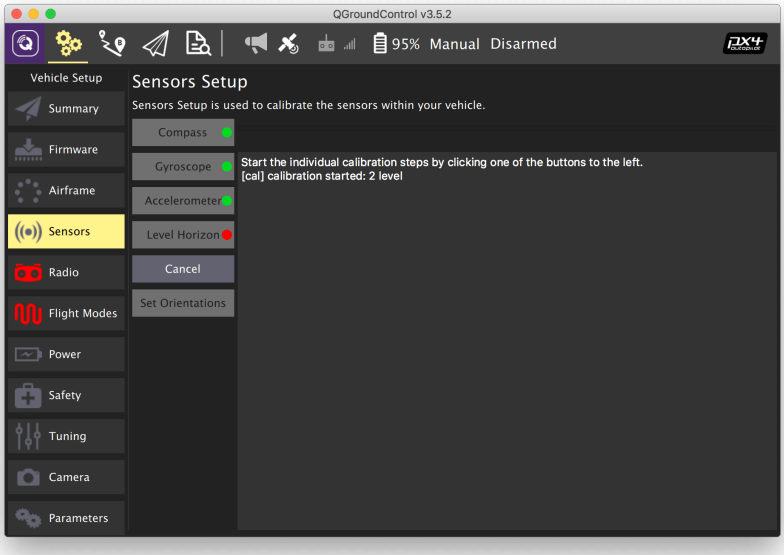


1. Select the *Accelerometer* submenu.
2. Choose the flight controller orientation (*ROTATION_NONE* if the arrow on the flight controller is aligned with the arrow on the frame).

3. Put the drone in one of the orientations marked by the red outline and wait for the appropriate outline to turn yellow.
4. Hold the drone in this orientation until the outline turns green. Do this for all orientations.

Read more in the PX4 docs:
https://docs.px4.io/master/en/config/accelerometer.html.

Level horizon

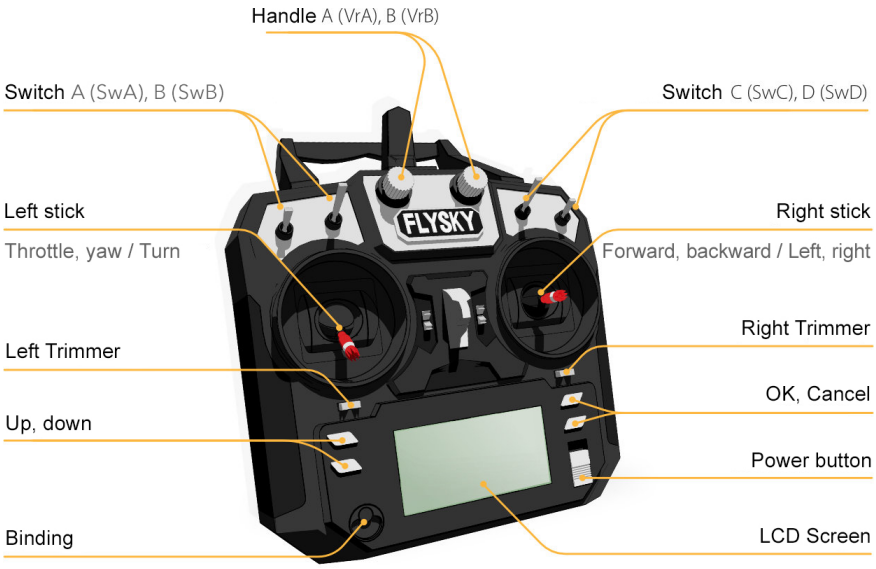


1. Select the *Level Horizon* submenu.
2. Choose the flight controller orientation (*ROTATION_NONE* if the arrow on the flight controller is aligned with the arrow on the frame).
3. Place the drone on a flat, horizontal surface.
4. Press *OK*.
5. Wait for the calibration to finish.

Read more in the PX4 docs:
https://docs.px4.io/master/en/config/level_horizon_calibration.html.

Next: RC setup.

RC setup



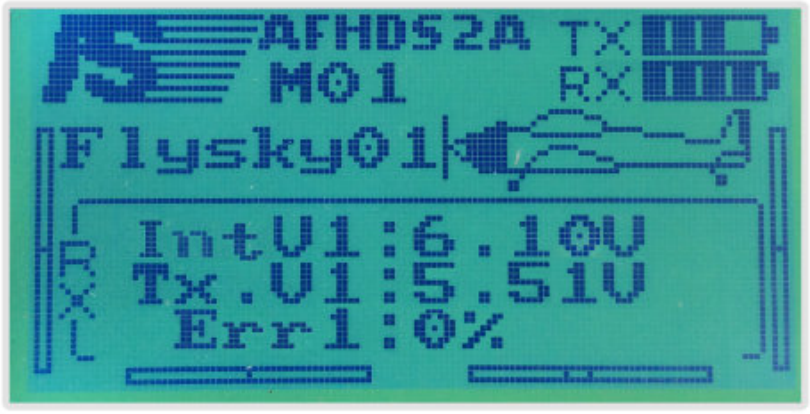
Before connecting and calibrating the RC, make sure that:

- There is no battery connected to the drone.
- The propellers are not mounted.

Connecting the RC transmitter

1. In QGroundControl software, go the *Vehicle Setup* panel and choose the *Radio* menu.
2. Power on the transmitter by sliding the **POWER** slider up.
3. Make sure the transmitter-receiver link is working.

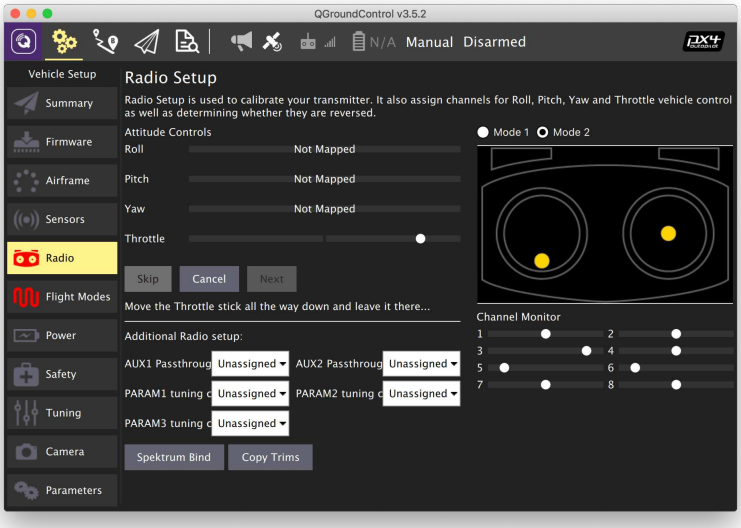
   The transmitter LCD screen should display the connection:



   The LED on the receiver should glow steadily. Read the radio troubleshooting article if the link does not work.

Transmitter calibration

1. Press the *Calibrate* button.
2. Set the *Throttle*, *Yaw*, *Pitch*, and *Roll* trims to 0
   - Trims are small constant offsets applied to a control in order to make your drone fly correctly.
   - Move the trimming slider to the center using trimming buttons until you hear a long beep. Do this for each axis.
3. Press *OK* in QGroundControl.



4. Place the left stick (throttle) in the bottom position and press *Next*.
5. Place the sticks in positions requested by QGroundControl.
6. When you get the *"Move all transmitter switches and/or dials back and forth to their extreme positions"* instruction, move all switches and dials to their extreme positions.
7. Press *Next*.
8. When you get the *"All settings have been captured. Click Next to write the new parameters to your board"*, press *Next*.

Further reading: https://docs.qgroundcontrol.com/en/SetupView/Radio.html

Next: Flight modes.

Using Flysky FS-A8S

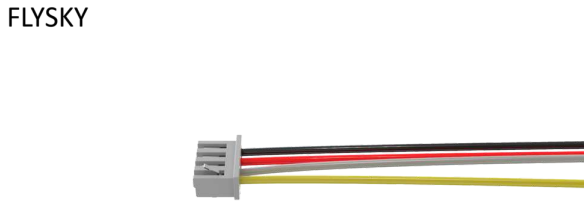
The Flysky FS-A8S receiver is compatible with the Flysky FS-i6 and FS-i6x transmitters. The receiver can output both analog PPM and digital S.Bus/i-Bus signals to the flight controller.

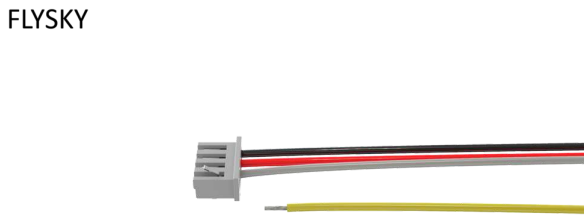
S.Bus is the preferred protocol for the receiver.

Making a cable

> You don't need to follow these steps if you already have the right cable; read on to learn how to bind your transmitter.

1. Gently remove the yellow wire from the receiver connector. Use sharp tweezers to lift up the plastic wire lock:



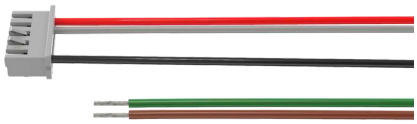


2. [Pixracer only] Remove the green and brown wires from the 5-pin connector:

PIXRACER



PIXRACER



3. [COEX Pix only] Remove the green wire (or blue if the green one is not present) from the 4-pin connector:

COEX PIX

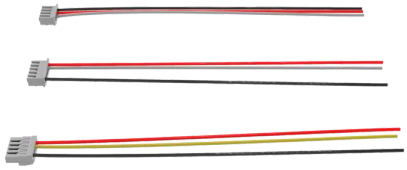


COEX PIX



4. Use side cutters to cut the Dupont connectors:





5. Strip and tin 5-7 mm of wire from each side:



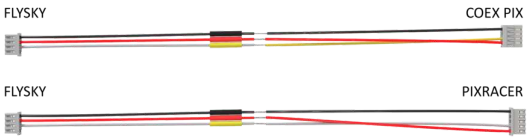
6. Put heat shrinking tubes on the wires:

FLYSKY

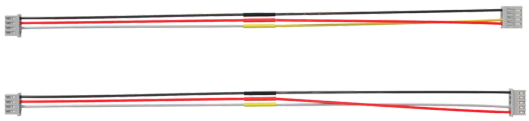


7. Solder the following wires:

   - black wire from the receiver connector to the black wire from the flight controller connector;
   - red wire from the receiver connector to the red wire from the flight controller connector;
   - white wire from the receiver connector to the white (if you're using Pixracer) or yellow (if you're using COEX Pix) wire from the flight controller connector:



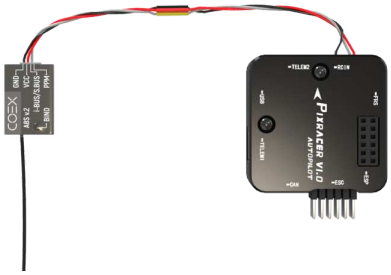
8. Put the heat shrinking tubes on the solder joints and heat them:

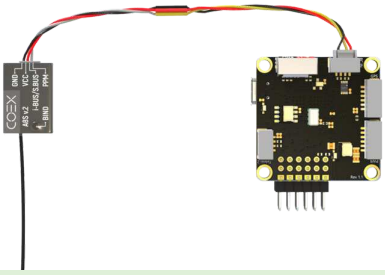


9. Twist your new cable:

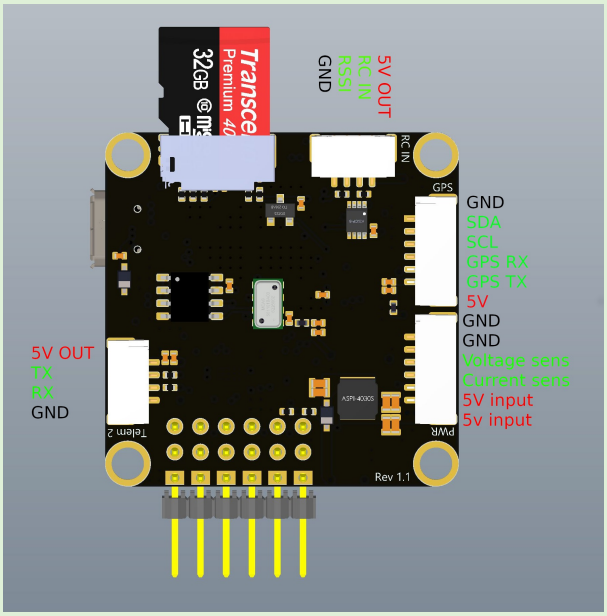


Connect your receiver to the RC IN port on your flight controller:





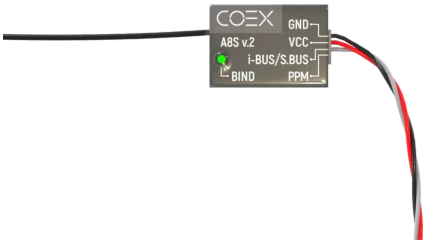
Double check that you're using the RC IN port on the COEX Pix:



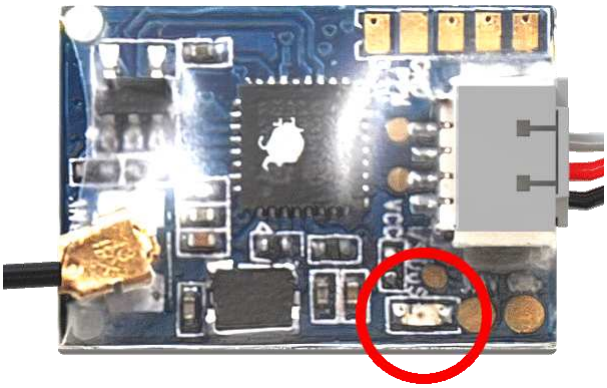
Binding your transmitter

Do the following to bind your transmitter to the FS-A8S receiver:

1. Make sure your flight controller is powered off.
2. Hold the **BIND** button on the receiver:



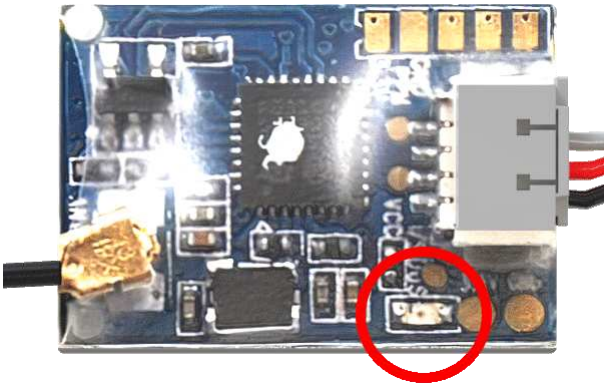
3. Turn on the flight controller. The LED light on the receiver should blink fast, about 3 times per second.



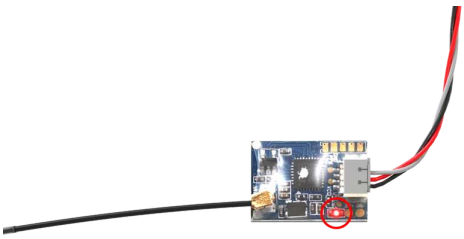
4. Hold the **BIND KEY** on your transmitter and power it on. You should see a message saying **RX Binding...**



5. The LED light on the receiver should start blinking slowly, about once per second.



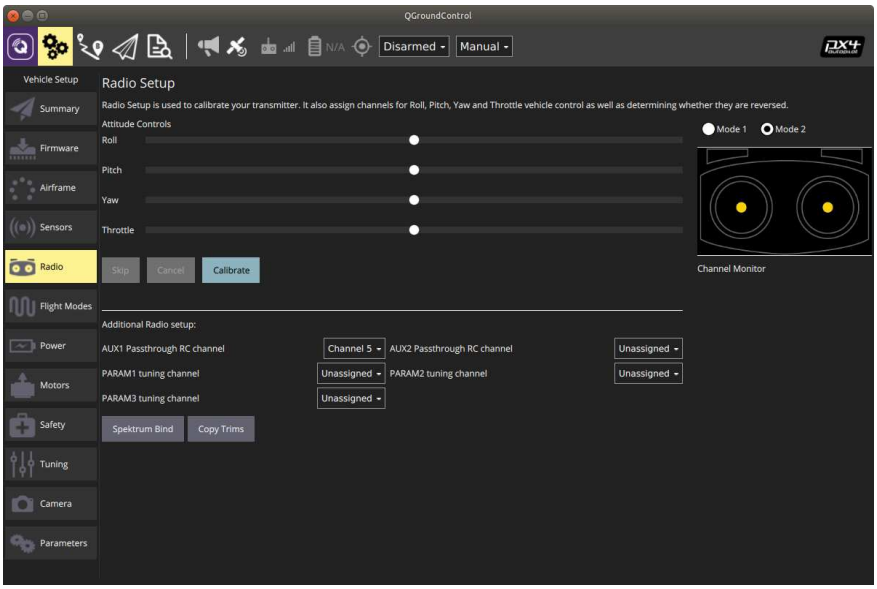
6. Turn your transmitter off and on again. The LED light on the receiver should glow steadily.



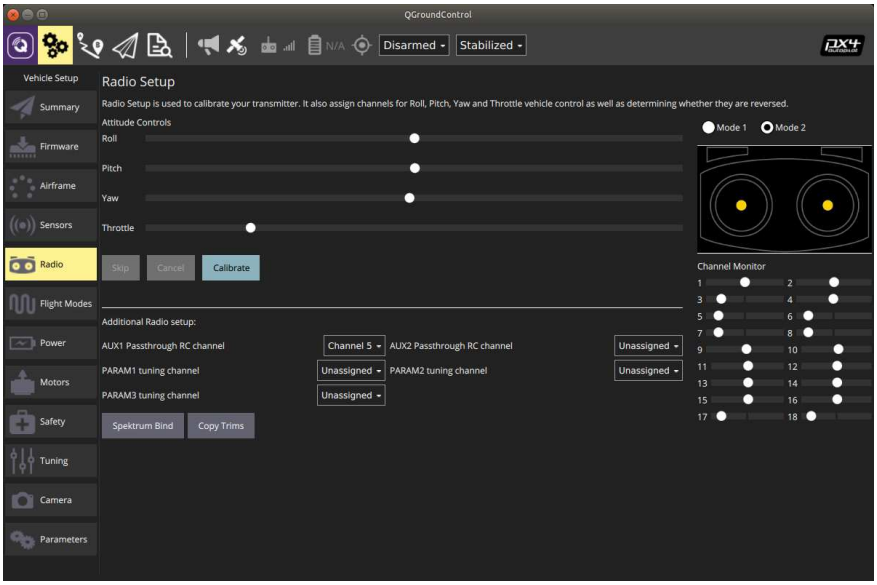
> This receiver does not send any telemetry data back to the transmitter. Your transmitter will not display any data like RSSI and drone battery level on its screen. In fact, there will be no indication that the transmitter is connected to the receiver. This is not a malfunction, the controls will still work.

Changing the receiver mode (S.Bus/i-Bus)

Connect your flight controller to your computer and open QGroundControl. In it, open the Radio tab:



If it shows zero channels under the transmitter image, hold the **BIND** key on the receiver for 2 seconds. You should then see 18 channels appear under the image:


QGroundControl
Disarmed
Stabilized
Vehicle Setup
Summary
Firmware
Airframe
Sensors
Radio
Flight Modes
Power
Motors
Safety
Tuning
Camera
Parameters
Radio Setup
Radio Setup is used to calibrate your transmitter. It also assign channels for Roll, Pitch, Yaw and Throttle vehicle control as well as determining whether they are reversed.
Attitude Controls
Roll
Pitch
Yaw
Throttle
Skip
Cancel
Calibrate
Mode 1
Mode 2
Channel Monitor
Additional Radio setup:
AUX1 Passthrough RC channel
Channel 5
AUX2 Passthrough RC channel
Unassigned
PARAM1 tuning channel
Unassigned
PARAM2 tuning channel
Unassigned
PARAM3 tuning channel
Unassigned
Spektrum Bind
Copy Trims

Flight modes

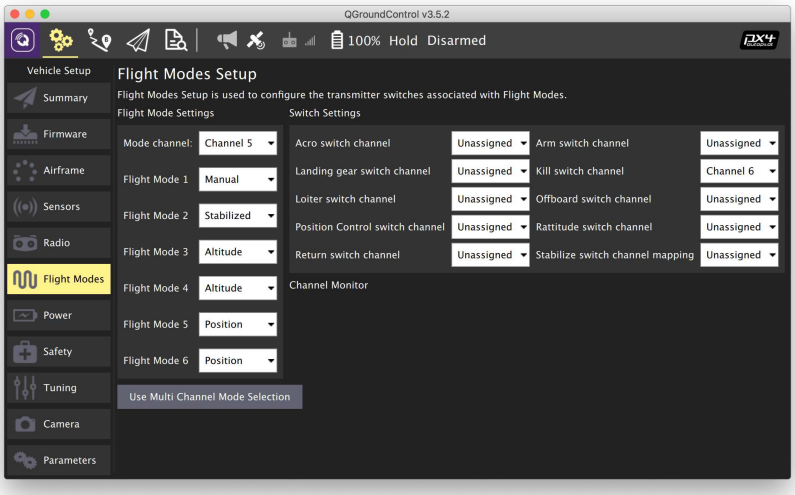
PX4 **mode** determines how the vehicle should react to commands and RC signals. Mode changing is usually mapped to one of the RC transmitter sticks.

In order to configure flight modes:

1. Open the *Vehicle Setup* panel in QGroundControl.
2. Select the *Flight Modes* menu.
3. Set the *Mode Channel* to the SwC switch (*Channel 6*).
4. Optionally, set the *Emergency Kill Switch Channel* to SwA switch (*Channel 5*).
5. Set desired flight modes.

   The following flight modes are recommended:

   - Flight Mode 1: *Stabilized*.
   - Flight Mode 4: *Altitude*.
   - Flight Mode 6: *Position*.
6. Check mode switching by changing the switch position.
7. Choose SwA (Channel 5) as emergency motor stop (*Kill switch*).



Flight modes description

Manual control

In manual mode the pilot controls the drone directly. GPS, computer vision data, and barometer are not used. Flying in these modes requires good drone piloting skills.

- **STABILIZED/MANUAL** — the mode with stabilized horizontal orientation. Allows control of the throttle, the copter pitch and roll, and the yaw rate.

- **ACRO** — control of throttle and the copter's pitch rate, roll rate, and yaw rate. Used by drone racers and in 3D piloting stunt shows.
- **RATTITUDE** — in the center, the right stick is similar to STABILIZED, at the edges, it passes to the ACRO mode.

Assisted flight modes

- **ALTCTL** (ALTITUDE) — control of the altitude rate, pitch, roll and yaw angular velocity. Requires a barometer or another altitude source.
- **POSCTL** (POSITION) — control of the altitude rate, forward/backward and right/left speed, and yaw angular velocity. It is the easiest flying mode. The barometer, GPS, computer vision, and other sensors are used.

Auto flight modes

In autonomous flight modes the quadcopter ignores the control signals from the transmitter and uses a program to fly.

- **OFFBOARD** mode uses an external computer (like a Raspberry Pi). This mode is used in Clover for autonomous flights.
- **AUTO.MISSION** – PX4 uses the mission pre-loaded into the drone (the mission is uploaded using ground control station over MAVLink). This mode is commonly used to move in a pre-planned path using GPS as a position source, for example, in photogrammetry.
- **AUTO.RTL** – the copter automatically returns to the takeoff (launch) point.
- **AUTO.LAND** – the copter lands at the current position.

Additional information: https://dev.px4.io/en/concept/flight_modes.html.

Next: Power setup.

Power setup

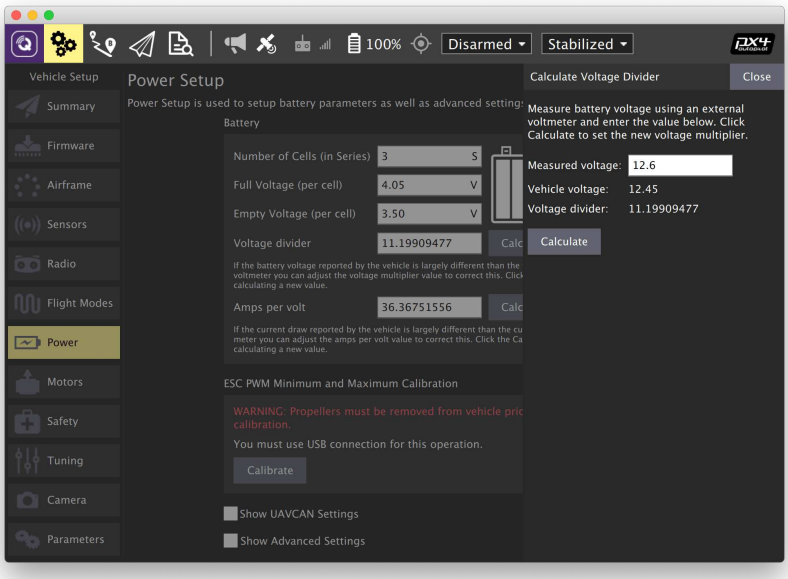
Open the *Vehicle Setup* tab and select the *Power* menu.

Calibrating the power sensor

> Power sensor calibration should be done with the battery pack connected to the drone.

1. In QGroundControl software, go the *Vehicle Setup* panel and choose the *Power* menu.
2. Set the *Number of cells* parameter according to the number of cells in your battery (*3* for the Clover 4 drone).
3. Calculate the voltage divider:
   - Measure voltage across the battery (you may use a battery voltage tester for that).
   - Press the *Calculate* button next to the *Voltage divider* label.
   - Put the battery voltage into the prompt and click *Calculate*.
   - Press *Close* to save the calculated value.

If there is no voltage indicator or manual calibration is not possible, set the average value of the voltage divider for the Clover 4 kit (*Voltage divider* = 11).

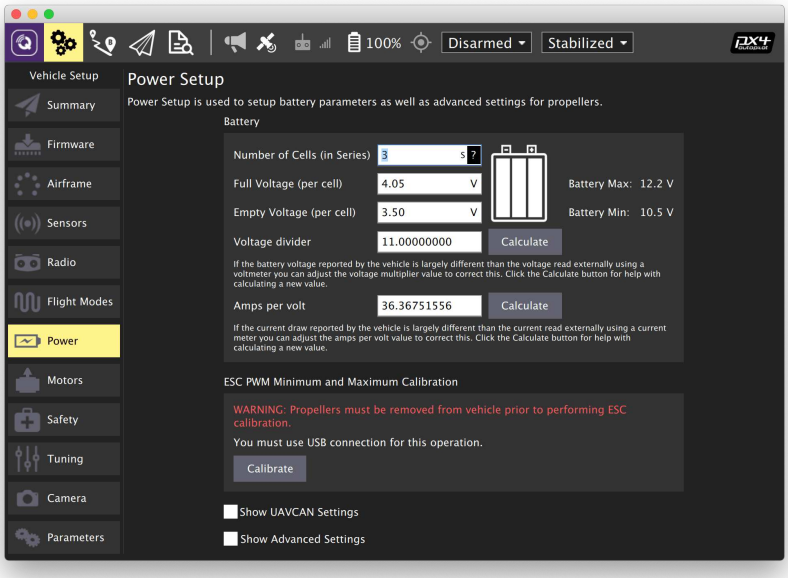


Further reading: https://docs.qgroundcontrol.com/en/SetupView/Power.html.

ESC calibration

> **Never attempt ESC calibration with propellers on!** In some cases the motors will start spinning with maximum speed.

1. Make sure the battery is disconnected and the propellers are not mounted.
2. Press *Calibrate*.
3. Connect the battery when prompted.
4. Wait for the *Calibration complete*.



Further reading:
https://docs.px4.io/master/en/advanced_config/esc_calibration.html.

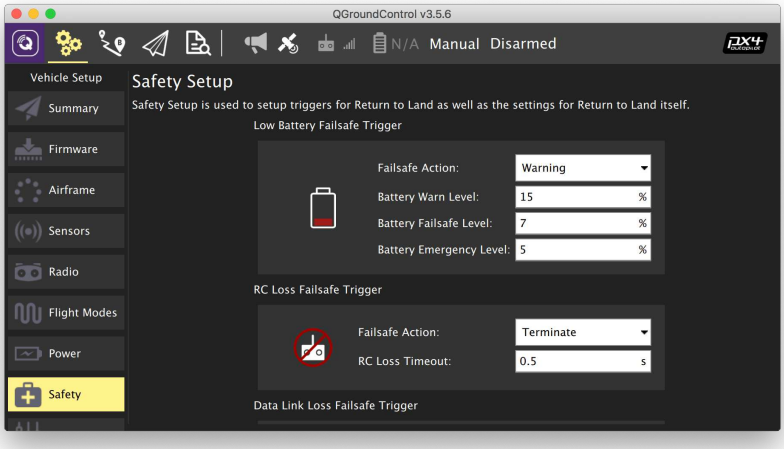
Next: Failsafe configuration

Failsafe configuration

Main article is available at https://docs.px4.io/master/en/config/safety.html.

The *Safety* panel allows you to configure actions that should be performed when a failsafe is triggered. You should at the very least configure the RC Loss failsafe, which is triggered when the RC transmitter link is lost:

1. In QGroundControl software, go to the *Vehicle Setup* panel and choose the *Safety* menu.
2. Select one of the following actions in the *RC Loss Failsafe Trigger* option:
   - *Land mode* – transition to automatic land mode;
   - *Terminate* – set all outputs to their failsafe values.
3. Set the timeout value before RC Loss triggers in the *RC Loss Timeout* field. We recommend setting it to 2 s.



Flight

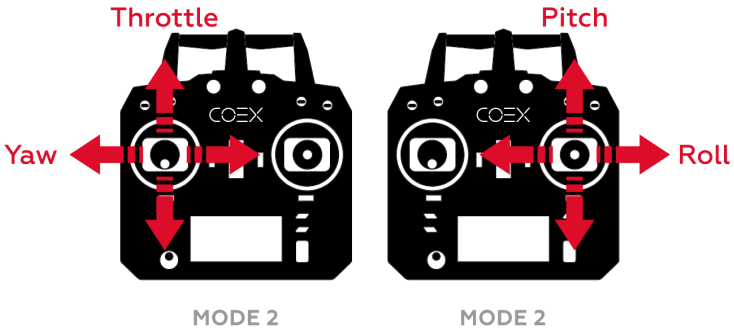
> See also official PX4 flying guide: https://docs.px4.io/master/en/flying/.

This section explains the basics of manual controlling the quadcopter in different modes using radio remote control (for autonomous flying see "Programming") section.

Main features of radio remote control

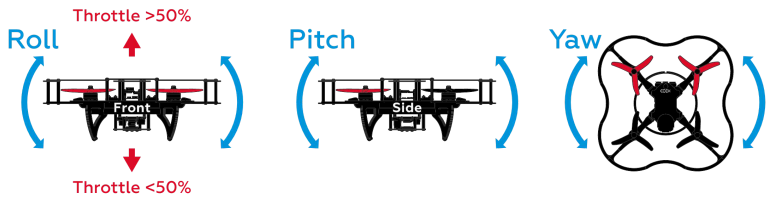
Before you can launch your drone, you need to understand how the radio remote control works.

The drone is controlled using two sticks on the remote control. By default, the left stick controls throttle and yaw, and the right stick controls roll and pitch. These terms are used for all aircraft, from airplanes to quadcopters.



- Throttle – is responsible for rotation speed of the motors.
- Yaw – is responsible for rotation around the vertical axis (Z), clockwise (when tilted to the right) and counterclockwise (when tilted to the left).
- Pitch – is responsible for tilting or moving forward / backward.
- Roll – is responsible for tilting or moving left / right.

These descriptions assume the aircraft is turned with its back to the pilot.



Flight Modes

Manual flight using the PX4 flight controller can be performed in different flight modes. They determine the radio controller stick assignments and other flight characteristics. For the complete list of flight modes, see the article "Flight modes".

The main manual modes are described below.

STABILIZED - horizontal angle stabilization mode. In this mode, the aircraft will hold the horizon if not controlled. Functions of sticks:

- Throttle – the average speed of rotation of the motors.
- Yaw – angular velocity around the vertical axis.
- Pitch – the angle of inclination around the transverse axis (forward / backward).
- Roll – the angle of inclination around the longitudinal axis (left / right).

POSCTL - position holding mode (requires positioning system enabled). Functions of sticks:

- Throttle – vertical flight speed.
- Yaw – angular velocity around the vertical axis.
- Pitch – linear speed of the drone (forward / backward).
- Roll – linear speed of the drone (left / right).

ACRO - controlling the average rotational speed of the motors and angular speeds of the drone. This mode is the most difficult to fly and is most often used by drone racers and 3D piloting shows to perform tricks. Functions of sticks:

- Throttle – the average speed of rotation of the motors.
- Yaw – angular velocity around the vertical axis.
- Pitch – angular velocity around the transverse axis (forward / backward).
- Roll – angular velocity around the longitudinal axis (left / right).

> Other flight controllers may have different names for similar flight modes.

Preparing to fly

Installing propellers and batteries

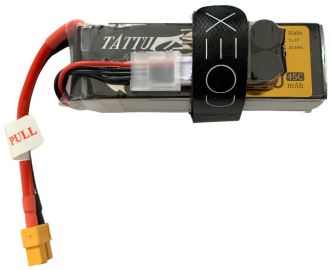
1. Install the battery strap.



2. Set the propellers according to the motor direction pattern.



3. Attach the buzzer and install the battery.





Setting the buzzer

In order not to over-discharge or damage the battery, it is recommended to use a voltage indicator (*buzzer*).

To configure the buzzer, connect it to the balance connector of your battery. By pressing the button, change the minimum voltage on the cells. The optimal value for the minimum voltage is *3.5-3.6 V*.





Flight readiness states

Before starting the flight, the aircraft must be in the *Armed* state.

- *Armed* state – motors rotate according to throttle stick position, copter is ready to fly.
- *Disarmed* state – motors do not rotate, copter does not respond to throttle stick.

By default, the aircraft is in the *Disarmed* state and switches to it automatically if you do not take off for a long time.

There are several ways to change the copter's state to *Armed*:

- Using the stick – move the left stick down to the right and wait a couple of seconds.



- Using the toggle switch – the Armed / Disarmed states can be set to one of the toggle switches. For more information on setting up, see the article on flight modes.
- With QGC – you can arm your drone programmatically. To do this, click on the *Disarmed* label in the header and select another state.
- In the user program – the copter can switch to *Armed* state if the `auto_arm=True` argument is specified in the navigation command, such as `navigate`, `set_position`, etc.

Kill switch

When the *Kill Switch* is activated, no control signals are sent to the motors and the motors stop rotating. This function is used in extreme cases, for example, if you lose control of the aircraft.

> Be careful, *Kill Switch* does not put the copter into *Disarmed* state!

Before disabling the *Kill Switch*, make sure the throttle stick is its down position and the aircraft is in *Disarmed* state. If the throttle stick is not in the lower position, when the *Kill Switch* is turned off, a signal corresponding to the stick position will be sent to the motors, which will lead your copter to jerk.

Next: Drone control exercises.

Drone control exercises

The following are the recommended exercises for novice pilots. Repeat each exercise as many times as necessary until you feel confident in it.

> In case there is a person nearby who can control a copter, use trainer mode.

> The first flights are strongly recommended to be performed behind a protective grid. In the absence of such, the flight area must be at least 6x6 m.

Turning on/off motors, changing flight modes

> For convenience, connect to the aircraft using QGC over Wi-Fi and turn on the sound. This will allow you to monitor the change in flight modes. If you cannot connect via Wi-Fi, connect via USB to check flight modes.

Be sure to set the flight mode to one of the toggle switches. To do this, switch the toggle to different positions and make sure that the mode change.

It is recommended to configure *Kill Switch*. To check it, follow these steps:

- Turn on *Kill Switch*, make sure QGC has a notification.
- Put the aircraft in *Armed* state and then enable *Kill Switch*. Make sure the motors stop. Then switch the *Kill Switch* to its original position. If the aircraft haven't automatically entered the *Disarmed* state due to inactivity, the motors will start rotating again.

> Set the aircraft to *Armed* state on the flight zone only.

Make sure modes switching is assigned to toggle switch that is convenient for you. Otherwise, change it according to the article on setting flight modes. Repeat the above steps several times in order to remember which toggle switches are responsible for what.

Working with throttle

The first step is to feel the responsiveness of the copter to the movement of the throttle stick and learn how to control it. Each drone has slightly different power reserves and therefore lifts off the ground at different stick positions.

For this exercise, only the throttle stick should be used. It is recommended not to use the rest of the sticks during the exercise.

The main tasks of the exercise:

1. Drift of the copter on the ground without taking off the ground.

Preflight checks

Do the following before takeoff:

1. Check the integrity of the aircraft and the propellers are clear to rotate.
2. Make sure the aircraft is with its back toward you.
3. Turn on the aircraft by connecting the battery.
4. Move back to a safe distance. It is recommended to maintain a minimum distance of 4-5 m to the aircraft.
5. Make sure the aircraft is in *Stabilized* mode.

Do not try to lift the copter off the ground right away, find the lowest possible stick position for the copter to start drifting on the ground. Failure to do so may result in damage or injury.

> If you lose control of the aircraft, you must immediately turn on *Kill Switch*.

Exercise №1. Slowly lift the throttle stick up until the aircraft starts to move. At this point, it will begin to slowly drift on the ground. Leave the throttle stick in this position and wait a couple of seconds, then move the throttle stick to its original position to land the aircraft. After landing the aircraft, turn off the motors by switching to *Disarmed* state. Repeat the exercise 5–10 times to get better feel for the copter's throttle stick response.

Exercise №2. Slowly lift the throttle stick up until the aircraft starts to lift slightly off the ground. Leave the throttle stick in this position and wait a couple of seconds, then land the aircraft as in Exercise №1. Repeat the exercise 5–10 times.

Exercise №3. Raise the throttle stick until the aircraft starts to drift on the ground, wait a second and continue increasing the throttle until the aircraft starts to lift off the ground, wait a second and land the aircraft. To consolidate, repeat the exercises 5–10 times, increasing the number of repetitions if necessary.

Working with roll and pitch

After mastering the throttle control of the copter, it is necessary to learn how to control its horizontal position. The right stick on the remote control is responsible for this.

Manipulating these axes is intuitive:

- Stick tilted forward (up) - aircraft moves forward.
- Stick tilted back (down) - aircraft moves backward.
- Stick tilted to the right - aircraft moves to the right.
- Stick tilted to the left - aircraft moves to the left.

The more the stick is tilted to the side, the more the aircraft will tilt to the side and the faster it moves.

The main tasks of the exercise:

1. Flying along the X axis, forward / backward.
2. Flying along the Y axis, left / right.
3. Stabilization of the copter in one place.
4. Flying in a square clockwise and counterclockwise.

> Always stay behind the aircraft with the rear facing towards you, otherwise you may lose control over it by mixing sides.

As with throttle control, perform the following steps before flying.

> If the aircraft is spinning strongly around its axis, land it and recalibrate the magnetometer and gyroscope.

Exercise №1. Similar to throttle exercises, raise the throttle stick until the aircraft starts to drift on the ground or bounce a little, then release the throttle stick, leaving it in that position, and raise the pitch stick, first up for a second, then down. The copter will gradually move away from you and then towards you. Repeat the exercise 5–10 times until you feel the copter's responsiveness to the stick movement.

Exercise №2. Raise the throttle stick until the aircraft starts to drift, then leave it and move the roll stick first to the right for a second, then to the left. The aircraft will gradually move first to the right and then to the left. Repeat the exercise 5–10 times until you feel the copter's responsiveness to the stick movement.

Exercise №3. Raise the throttle stick until the aircraft starts to drift, then leave it. Combine the first and second exercises and try to stabilize the aircraft at one point, compensating for its drift with the stick. Hold the aircraft for 50–60 seconds.

Exercise №4. Raise the throttle stick until the aircraft starts to drift, then leave it. When you feel the copter's responsiveness to stick changes, make a "square" shape with a side of 1 m, first clockwise and then counterclockwise. Perform the figures 2–3 times.

Air cushion and control in it

The concept of *air cushion* is very important for all flying vehicles. The air cushion itself is a zone of increased pressure created by the air being forced through the propellers. This area is characterized by turbulence and air currents affecting the flight of the copter.

Pilots try to avoid flying in an air cushion, but there is a stable area at the boundary where the aircraft can hover at minimum throttle. In this case, it feels like the copter has "sat down" on an air cushion.

The main feature and advantage of such a flight is that the copter will not change altitude with one throttle value.

Main tasks:

1. Stabilization of the copter in one place.
2. Flying in a square.
3. Flying in a circle.

Similarly to the previous exercises, perform the following steps before take off.

Exercise №1. Raise the throttle stick until the copter flies over the air cushion and is above it (height from floor ~ 25-30 cm, for Clover 4 copter). The aircraft should not climb up or fall down, the flight altitude should stabilize. As in the previous exercise, adjust the X and Y position of the aircraft using the roll and pitch sticks. As a result, the copter should hover at one point with slight wiggle to the sides. Hold the aircraft for 30–40 seconds.

Exercise №2. Raise the aircraft on the air cushion and stabilize it at one point. Next, fly over a square with a side of 1 m, first clockwise, then counterclockwise. Repeat the path 2–3 times in each direction.

Exercise №3. Raise the aircraft on the air cushion and stabilize it at one point. Try to fly a circle with the copter around 1 m in diameter, clockwise and counterclockwise. Repeat the path 2–3 times in each direction.

Working with yaw

In the visual control of multicopter devices, yaw does not play as important role as with fixed wing vehicles, since the copter can move in any direction regardless of where it is directed.

> The term *yaw* refers to the rotation of the aircraft around the vertical axis.

Main tasks:

1. Rotate the copter, orienting the rear of the copter towards you.
2. Turning around the copter, orienting the rear part towards you.

It is recommended that you find plenty of free space for the exercises presented.

Similarly to previous exercises, perform preflight checks before takeoff.

Exercise №1. Raise the aircraft on the air cushion and stabilize it at one point. Fly a circle around you with the copter, at a distance of 2–3 m, while rotating it so that the back of the copter is always directed towards you. Do the exercise clockwise and counterclockwise. Repeat the exercise 4–5 times.

Exercise №2. Raise the aircraft on the air cushion and stabilize it at one point. Walk around the aircraft while turning it so that the rear is facing you. Walk around the aircraft clockwise and counterclockwise. Repeat the exercise 4–5 times.

> Additional exercises are much more difficult than usual and are not required. Only proceed with them if you are already confidently flying the copter.

Additional exercise №1. Raise the aircraft on the air cushion and stabilize it at one point. Face the aircraft with its front facing you and try to fly it backwards.

Additional exercise №2. Raise the aircraft on the air cushion and stabilize it at one point. Fly so that the front of the aircraft is always facing the direction of the aircraft.

Free flight

If you can complete each of the exercises described above, chances are you already know how to freely take off and fly the aircraft. Some exercises will be presented below to consolidate the acquired skills.

Exercises:

- Flying in a vertical square.
- Flying along the sides of the cube.
- Flying in a vertical circle.
- Flight of the eight.
- Ascent of the copter in a spiral.

Strengthen the acquired skills as many times as necessary for you.

Raspberry Pi

Raspberry Pi is a single-board computer that fits in the palm, created on the basis of ARM mobile microprocessor. It features low energy consumption, and it can even run on solar panels. A Raspberry Pi is included in the kits for programmable quadcopters "Clover".



Technical specifications:

- Weight is 45 grams.
- Clock rate is 1.2 GHz.
- Graphics core in the Broadcom BCM2837 processor.
- RAM is 1 GB.
- Four USB 2.0 ports.
- An HDMI port.

Raspberry Pi is connected to the flight controller in the Clover kit and is used as a companion computer. It can be used to connect to the drone over Wi-Fi, perform autonomous flights, access peripherals and much more.

Next: Raspberry Pi image

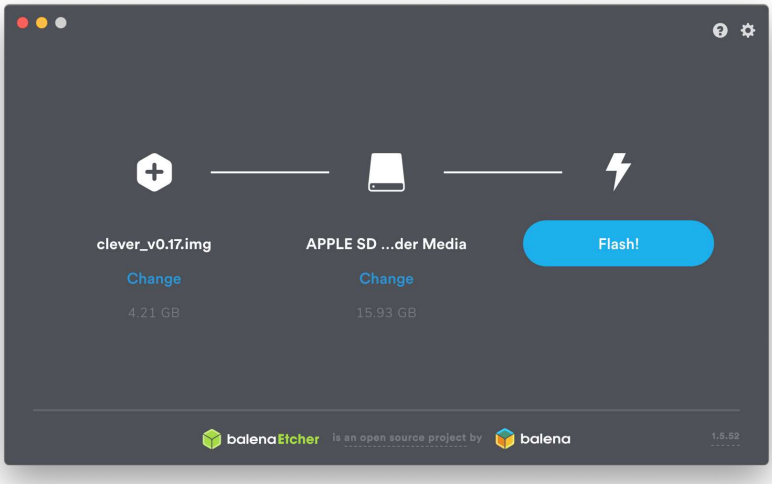
Raspberry Pi image

The RPi image for Clover contains all the necessary software for working with Clover and programming autonomous flights. The Clover platform is based on Raspbian OS and robotics framework ROS. The source code of the image builder and all the additional packages is available on GitHub.

Usage

> Starting from version v0.22, the image is based on ROS Noetic and using Python 3. If you want to use ROS Melodic and Python 2, use version v0.21.2.

1. Download the latest stable release of the image – **download**.
2. Download and install Etcher, the software for flashing images (available for Windows/Linux/macOS).
3. Put the MicroSD-card into your computer (use an adapter if necessary).
4. Flash the downloaded image to the card using Etcher.
5. Put the card into the Raspberry Pi.



After flashing the image on the MicroSD-card, you can connect to the Clover over Wi-Fi, use wireless connection in QGroundControl, gain access to the Raspberry over SSH and use all the other features.

Next: Connecting over Wi-Fi.

Connecting to Clover via Wi-Fi

> The following applies to image version **0.20** and up. See previous version of the article for older images.

RPi image provides a pre-configured Wi-Fi access point with SSID `clover-xxxx`, where `xxxx` are four random numbers that are assigned when your Raspberry Pi is run for the first time.

Connect to this Wi-Fi using the password `cloverwifi`.

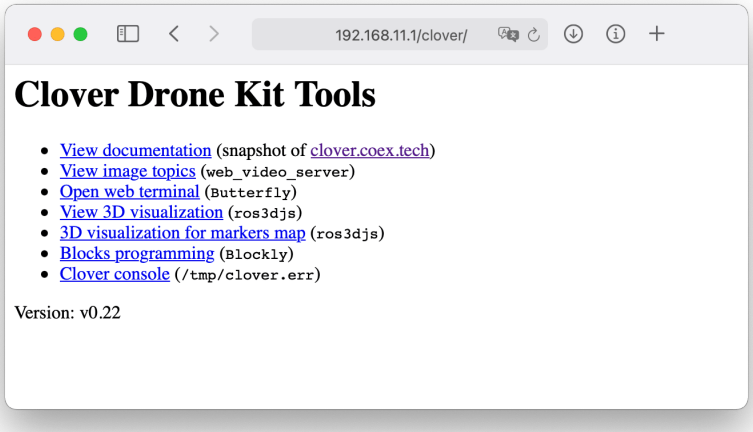




To edit Wi-Fi settings, or to obtain more detailed information about the network device on Raspberry Pi, read this article.

Web interface

After connecting to Clover Wi-Fi, open http://192.168.11.1 in you web browser. It contains all the basic web tools of Clover: viewing image topics, web terminal (Butterfly), and the full copy of this documentation.



Next: Connecting Raspberry Pi to the flight controller.

Connecting Raspberry Pi to the flight controller

In order to program autonomous flights, work with Pixhawk or Pixracer over Wi-Fi, use controller app and access other functions you need to connect your Raspberry Pi to the flight controller.

USB connection

USB connection is the preferred way to connect to the flight controller.



1. Connect your FCU to the Raspberry Pi using a microUSB to USB cable.
2. Connect to the Raspberry Pi over SSH.
3. Make sure that the connection is working properly by running the following command on the Raspberry Pi:

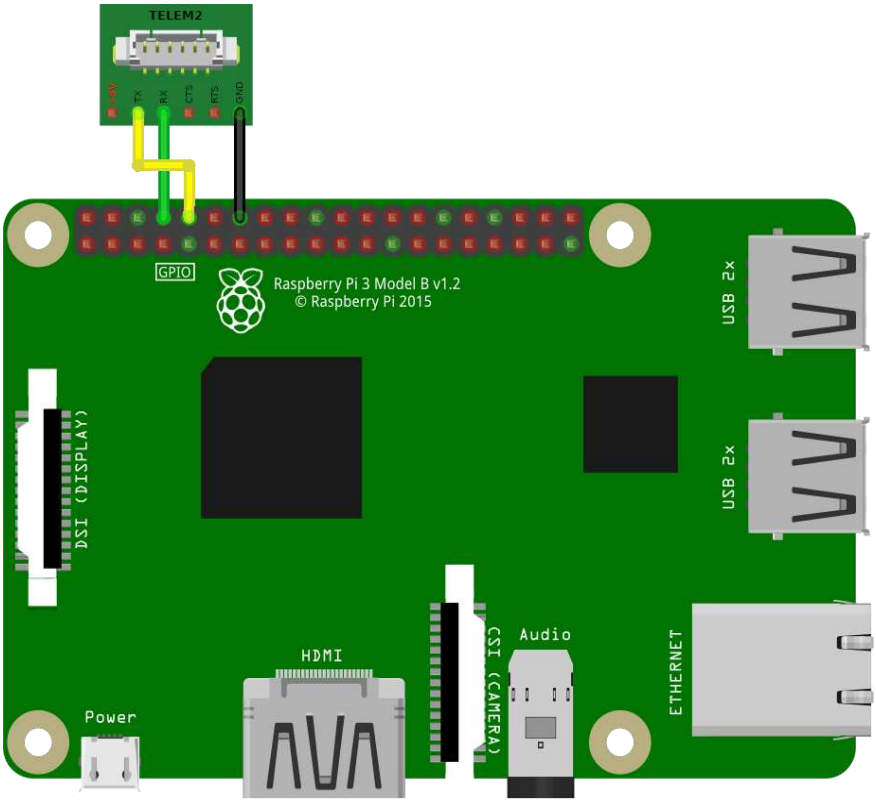
   ```
   rostopic echo /mavros/state
   ```

   The `connected` field should have the `True` value.

You need to set the `CBRK_USB_CHK` parameter to 197848 for the USB connection to work.

UART connection

UART connection is another way for the Raspberry Pi and FCU to communicate.



If the pin marked GND is occupied, you can use any other ground pin (look at the pinout for reference).

1. Connect the TELEM 2 port on the flight controller using a UART cable to the Raspberry Pi pins following this instruction: the black cable (*GND*) to Ground, the green cable (*UART_RX*) to *GPIO14*, the yellow cable (*UART_TX*) to *GPIO15*. Do not connect the red cable (*5V*).
2. In PX4 of version v1.9.0 or higher, set parameter values: `MAV_1_CONFIG` to TELEM 2, `SER_TEL2_BAUND` to 921600 8N1. In PX4 of version prior to v1.9.0 the parameter `SYS_COMPANION` should be set to `Companion Link (921600 baud, 8N1)`, to set it correctly use the old version of QGC v3.3.1.
3. Connect to the Raspberry Pi over SSH.
4. Check the presence of the parameters `enable_uart=1` and `dtoverlay=pi3-disable-bt` in the file `/boot/config.txt` by running the following command on the Raspberry Pi:

   ```
   cat /boot/config.txt | grep -E "^enable_uart=.|^dtoverlay=pi3-disable-bt"
   ```

   If the parameters in the file are different or missing, then edit the file and restart the Raspberry Pi.

5. Change the connection type from `usb` to `uart` in the Clover' launch file `~/catkin_ws/src/clover/clover/launch/clover.launch`:

```
<arg name="fcu_conn" default="uart"/>
```

If you change the launch file, you need to restart the `clover' service:

```
sudo systemctl restart clover
```

6. Make sure that the connection is working properly by running the following command:

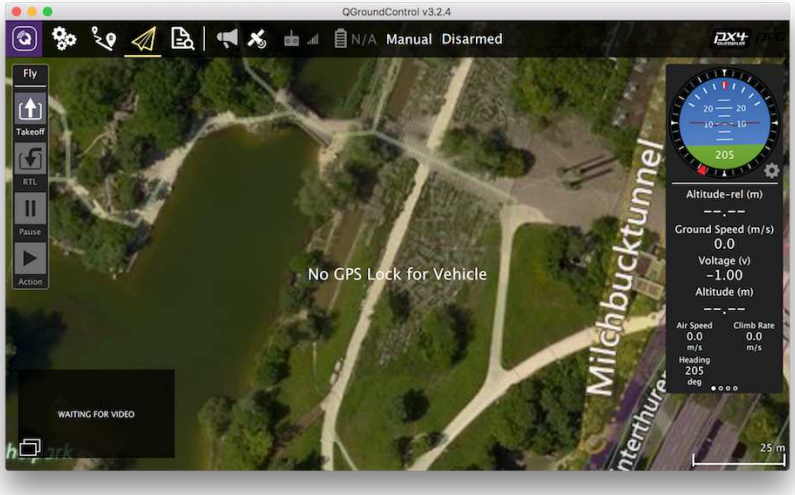
```
rostopic echo -n1 /mavros/state
```

The `connected` field should have the `True` value.

Read more in the PX4 docs: https://docs.px4.io/main/en/peripherals/serial_configuration.html.

Next: Using QGroundControl over Wi-Fi

Using QGroundControl via Wi-Fi



You can monitor, control, calibrate and configure the flight controller of the quadcopter using QGroundControl via Wi-Fi. This requires connecting to Wi-Fi of the `clover-xxxx` network.

After that, in the Clover launch-file `/home/pi/catkin_ws/src/clover/clover/launch/clover.launch`, choose one of the preconfigured bridge modes.

After editing the launch-file, restart the `clover` service:

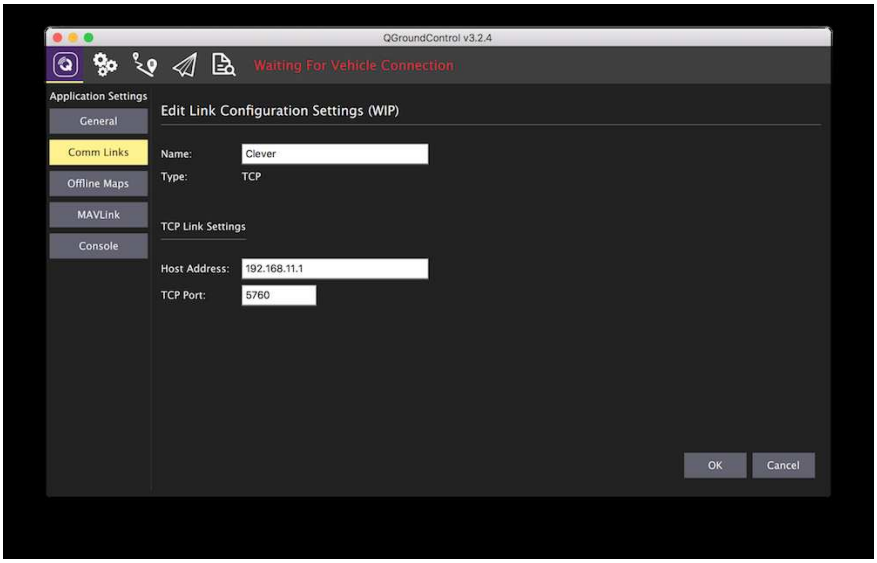
```
sudo systemctl restart clover
```

TCP bridge

Change parameter `gcs_bridge` in the launch file:

```
<arg name="gcs_bridge" default="tcp"/>
```

Then in the QGroundControl program, choose Application Settings > Comm Links > Add. Create a connection with the following settings:



Then choose the created connection from the list of connections, and click "Connect".

UDP bridge (with automated connection)

Change parameter `gcs_bridge` in the launch file:

```
<arg name="gcs_bridge" default="udp-b"/>
```

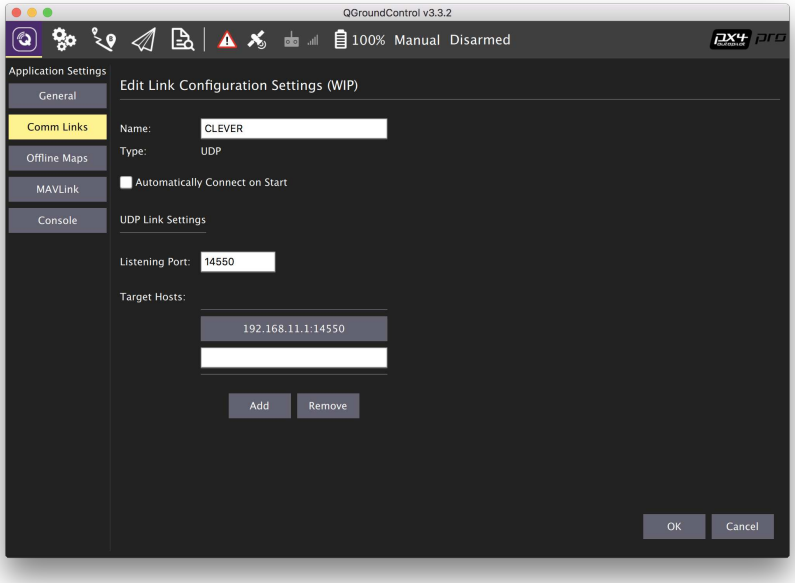
After opening the QGroundControl application, the connection should be established automatically.

UDP bridge (without automated connection)

Change parameter `gcs_bridge` in the launch file:

```
<arg name="gcs_bridge" default="udp-b"/>
```

Then in the QGroundControl program, choose Application Settings > Comm Links > Add. Create a connection with the following settings:



Then choose the created connection from the list of connections, and click "Connect".

UDP broadcast bridge

> The feature of the UDP broadcast bridge is the ability to view drone telemetry simultaneously from multiple devices (e.g., a phone and a PC). It is also well suited for devices networking using a router.

Change parameter `gcs_bridge` in the launch file:

```
<arg name="gcs_bridge" default="udp-b"/>
```

After opening the QGroundControl application, the connection should be established automatically.

Next: Remote access using SSH

SSH access to Raspberry Pi

RPi image is configured to be accessed via SSH for editing files, loading data and running programs.

For the SSH access, it is necessary to connect to Raspberry Pi over Wi-Fi (connection via an Ethernet cable is also possible).

In Linux or macOS, run the command prompt, and execute command:

```
ssh pi@192.168.11.1
```

Password: `raspberry`.

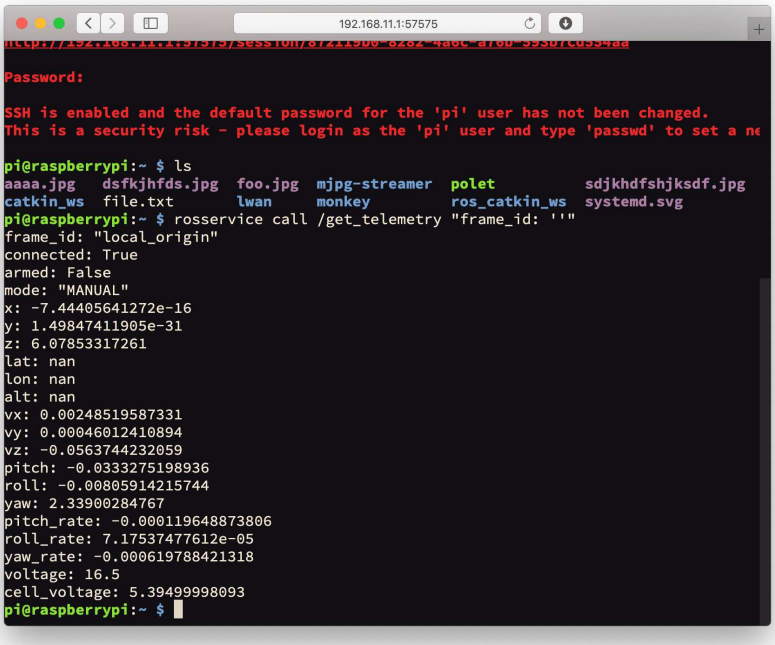
For SSH access from Windows, you may use PuTTY. You can also gain SSH access from your smartphone using the Termius app.

> To avoid entering the password each time you connect via SSH, see the article on using SSH keys.

Read more: https://www.raspberrypi.org/documentation/remote-access/ssh/README.md

Web access

Starting with version 0.11.4 of the image, access to the shell is also available via a web browser (using Butterfly). To gain access, open web page http://192.168.11.1, and select link *Open web terminal*:



Next: Command-line interface

Command line interface

The Raspberry Pi OS, Raspbian, uses CLI as its primary user interface (which is common for Linux-based operating systems). You can use a secure shell connection to access the command line.

Basic commands

> Double-tapping the `Tab ↹` key autocompletes the command or its argument. This is known as "tab completion".

Show the contents of the current directory:

```
ls
```

Change current (working) directory:

```
cd catkin_ws/src/clover/clover/launch
```

Go one directory level up:

```
cd ..
```

Print path to the current directory:

```
pwd
```

Go to the user's home directory:

```
# all three commands are equivalent, where the tilde character (~) is an abbrev
# path entry to the home directory, and the $HOME variable stores this path
cd
cd ~
cd $HOME
```

Print contents of the `file.py` file:

```
cat file.py
```

Run `file.py` as a Python script:

```
python3 file.py
```

Reboot Raspberry Pi:

```
sudo reboot
```

You can terminate currently running (foreground) program by pressing `Ctrl` + `C`.

Read more about the Linux command line in the Raspberry Pi documentation: https://www.raspberrypi.org/documentation/linux/usage/commands.md.

Editing files

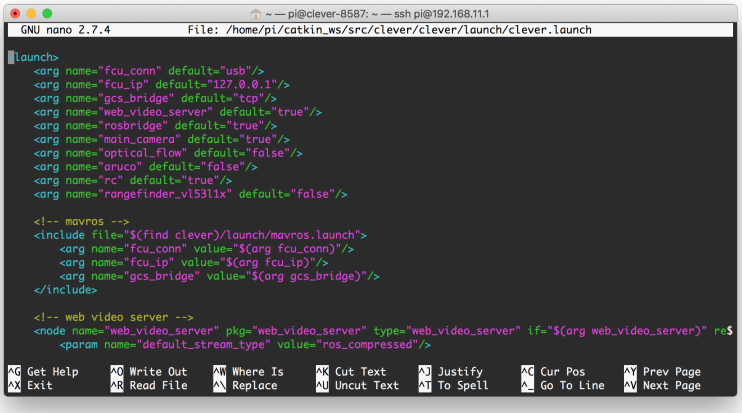
You can use **nano** to edit files on the Raspberry Pi. It is one of the more user-friendly console-based text editor.

1. Use the following command to edit or create a file:

   ```
   nano path/to/file
   ```

   For example:

   ```
   nano ~/catkin_ws/src/clover/clover/launch/clover.launch
   ```



2. Edit the file.
3. Press `Ctrl` + `X`, `Y`, `Enter` to save your file and exit.
4. Restart the `clover` service if you've changed .launch files:

   ```
   sudo systemctl restart clover
   ```

You may also use other editors like **vim** if you prefer.

Resetting changes

For resetting all the changes in Clover package related files (`launch`-files), use git:

```
cd ~/catkin_ws/src/clover
git checkout .
sudo systemctl restart clover
```

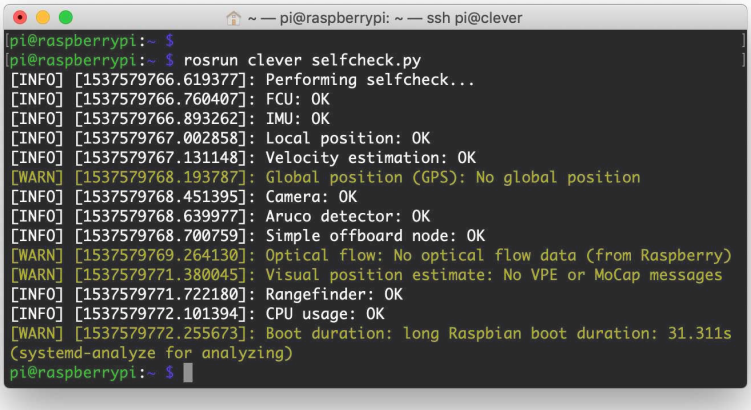
Automatic check

The RPi image contains a tool for automatic checking the correctness of all the settings and subsystems of the drone – **selfcheck.py**.

It is generally a good idea to perform this check before flight, especially before an autonomous one.

In order to run it, enter the following command in the Raspberry Pi console:

```
rosrun clover selfcheck.py
```



Description of some checks:

- FCU – checks for proper connection with the flight controller;
- IMU – checks whether the data from from IMU is sane;
- Local position – checks presence of local position data;
- Velocity estimation – checks whether drone velocity estimation is sane(**autonomous flight is not to be performed if this check fails!**);
- Global position (GPS) — checks for presence of global position data (GPS module is required for this check);
- Camera — checks for proper operation of the Raspberry camera.
- ArUco — checks whether ArUco detection is working
- VPE — checks whether VPE data is published.
- Rangefinder — checks whether rangefinder data is published.
- RPi health – checks the onboard computer status.
- CPU usage – checks the CPU load of the onboard computer.

> Pay attention on the checks marked with *WARN* sign. If necessary, contact Copter Express technical support.

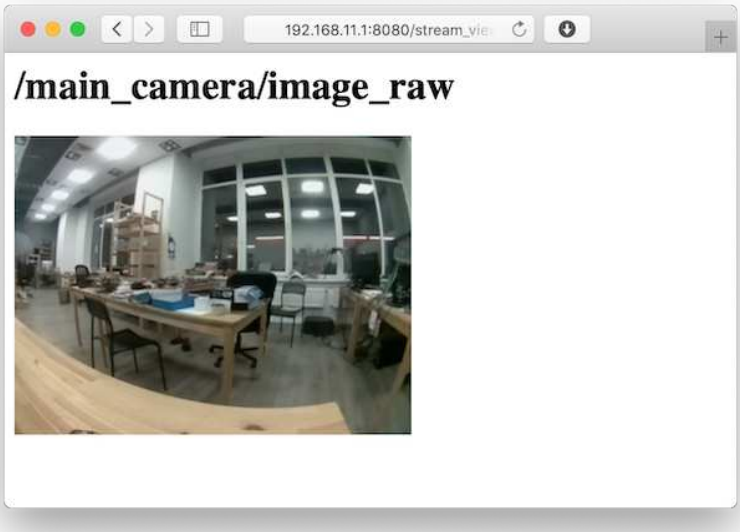
Viewing images from cameras

To view images from cameras (or other ROS topics), you can use rviz, rqt, or watch them in a browser using web_video_server.

See read more about using rqt.

Viewing in a browser

To view a video-stream, you have to connect to Wi-Fi network of the Clover (`clover-xxxx`), navigate to page http://192.168.11.1:8080/, and choose the topic.



If the image is transmitted too slow, you can speed it up by changing GET parameter `quality` (from 1 to 100), which is responsible for video-stream compression, for example:

http://192.168.11.1:8080/stream_viewer?topic=/main_camera/image_raw&quality=1

At the URL above, a stream from the main camera will be available in the minimum possible quality.

Parameters `width` , `height` , etc. re also available. Read more about `web_video_server` : http://wiki.ros.org/web_video_server.

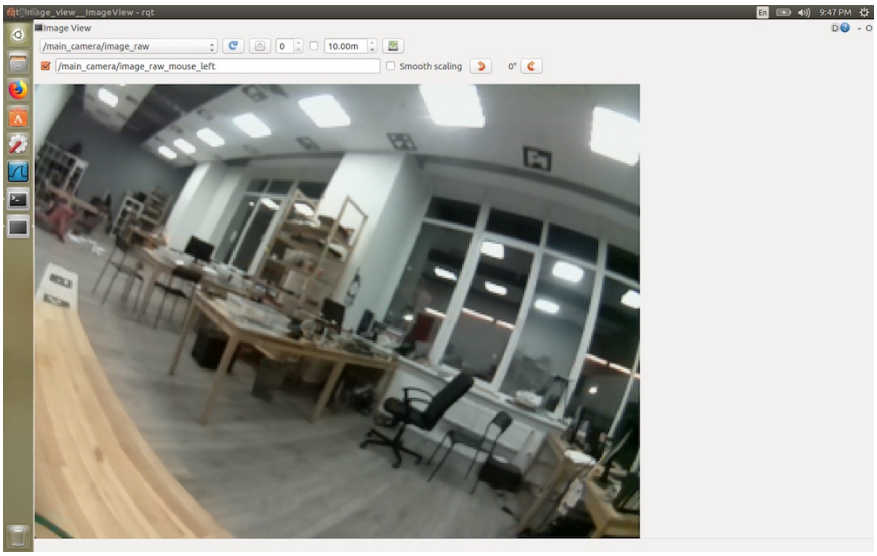
Browse with rqt_image_view

To browse images with the rqt tools the user needs a computer with Ubuntu 20.04 and ROS Noetic.

Connect to the Clover Wi-Fi network an run `rqt_image_view` with its IP-address:

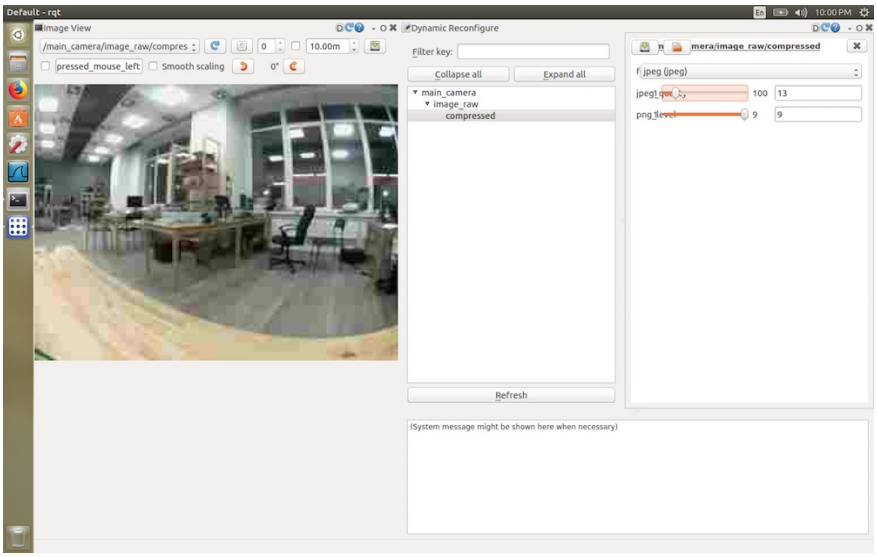
```
ROS_MASTER_URI=http://192.168.11.1:11311 rqt_image_view
```

Choose a topic for browsing, for example `/main_camera/image_raw` :



To reduce network load and reduce latency, use a compressed version of the topic – `/main_camera/image_raw/compressed` .

To change the compression settings use the rqt-plugin Dynamic Reconfigure:



Refer to more about rviz and rqt.

Programming



The Clover platform allows a Raspberry Pi computer to be used for programming autonomous flights. The flight program is typically written using the Python programming language. The program may receive telemetry data (which includes battery data, attitude, position, and other parameters) and send commands like: fly to a point in space, set attitude, set angular rates, and others.

The platform utilizes the ROS framework, which allows the user program to communicate with the Clover services that are running as a `clover` systemd daemon. The MAVROS package is used to interact with the flight controller.

PX4 uses OFFBOARD mode for autonomous flights. The Clover API can be used to transition the drone to this flight mode automatically. If you need to interrupt the autonomous flight, use your flight mode stick on your RC controller to transition to any other flight mode.

Positioning system

A drone has to use a positioning system to be able to hover still or to fly from point to point. The system should compute the drone position and feed this data into the flight controller. Clover allows using multiple positioning systems, such as optical flow (requires a camera and a rangefinder), fiducial markers (requires a camera and markers), GPS and others.

Optical flow

Optical flow is used to compute shifts between consecutive frames and to use this data to compute the drone shifting in space.

Read more in the Optical Flow article.

ArUco markers

Fiducial markers allow the drone to compute its position relative to these markers. This data may then be transferred to the flight controller.

Read more about ArUco markers in our articles about them.

GPS (outdoor flight)

GPS allows you to specify global Earth coordinates (latitude and longitude). The `navigate_global` function takes these as parameters instead of the usual cartesian coordinates.

Read more in the GPS connection article.

Autonomous flight

> For studying Python programming language, see tutorial.

After you've configured your positioning system, you can start writing programs for autonomous flights. Use the SSH connection to the Raspberry Pi to run your scripts.

Before the first flight it's recommended to check the Clover's configuration with selfcheck.py utility:

```
rosrun clover selfcheck.py
```

In order to run a Python script use the `python3` command:

```
python3 flight.py
```

Below is a complete flight program that performs a takeoff, flies forward and lands:

```
import rospy
from clover import srv
from std_srvs.srv import Trigger

rospy.init_node('flight')

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)

# Takeoff and hover 1 m above the ground
navigate(x=0, y=0, z=1, frame_id='body', auto_arm=True)

# Wait for 3 seconds
rospy.sleep(3)

# Fly forward 1 m
navigate(x=1, y=0, z=0, frame_id='body')

# Wait for 3 seconds
rospy.sleep(3)

# Perform landing
land()
```

> The `navigate` function call is not blocking; that is, the program will continue executing the next commands before the drone arrives at the set point. Look at the `navigate_wait` snippet for a blocking function.

Note that only the first `navigate` call has its `auto_arm` parameter set to `True`. This parameter arms the drone and transitions it to the OFFBOARD flight mode.

The `frame_id` parameter specifies which frame of reference will be used for the target point:

- `body` is rigidly bound to the drone body;
- `navigate_target` has its origin at the last target point for `navigate`;
- `map` is the drone's local frame;
- `aruco_map` is bound to the ArUco marker map;
- `aruco_N` is bound to the marker with ID=N.

Read more in the coordinate systems article.

You can also use the "Autonomous flight" article as an API reference.

> Clover supports blocks-based programming as well.

Additional periphery

The Clover platform also exposes APIs for interacting with other peripherals. Read more in the following articles:

- LED strip;
- laser rangefinder;
- GPIO;
- ultrasonic rangefinder;
- camera.

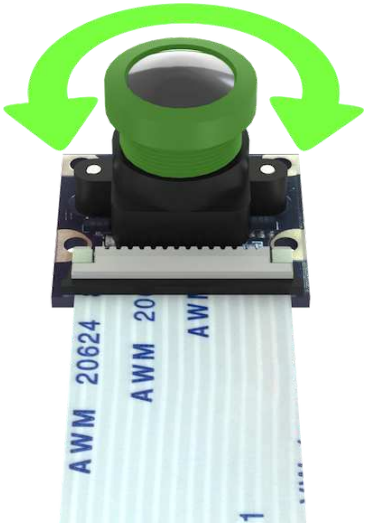
Camera setup

> The following applies to image version **0.20** and up. See previous version of the article for older images.

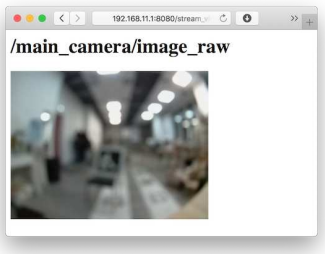
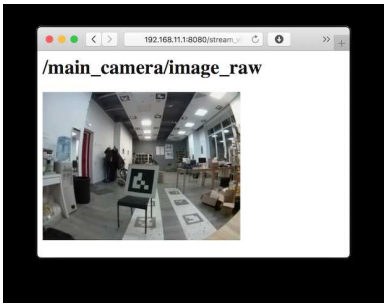
Computer vision modules (like ArUco markers and Optical Flow) require adjusting the camera focus and set up camera position and orientation relative to the drone body. Optional camera calibration can improve their quality of performance.

Focusing the camera lens

In order to focus the camera lens, do the following:



1. Open the live camera stream in your browser using web_video_server.
2. Rotate the lens to adjust the image. Make sure the objects that are 2-3 m from the camera are in focus.

Unfocused image	Focused image
	

Setting the camera position

Position and orientation of the main camera is set in the `~/catkin_ws/src/clover/clover/launch/main_camera.launch` file:

```
<arg name="direction_z" default="down"/> <!-- direction the camera points: down
<arg name="direction_y" default="backward"/> <!-- direction the camera cable po
```

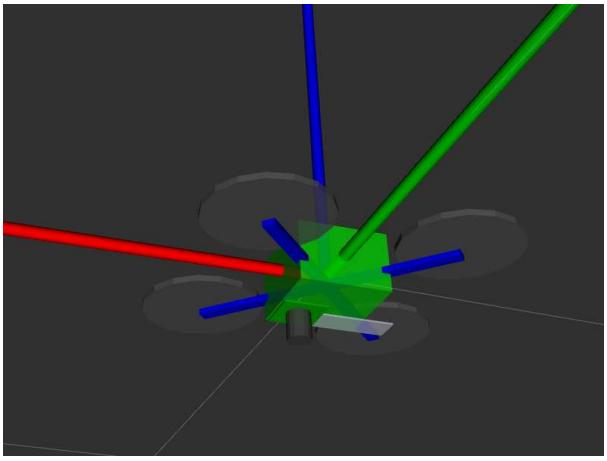
To set the orientation, define:

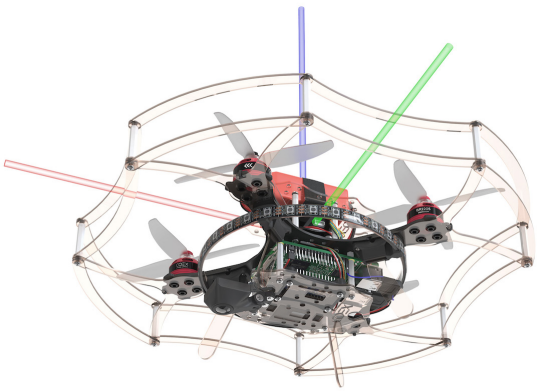
- direction the camera lens points `direction_z` : `down` or `up` ;
- direction the camera cable points `direction_y` : `backward` or `forward` .

Examples

Camera faces downward, cable goes backward

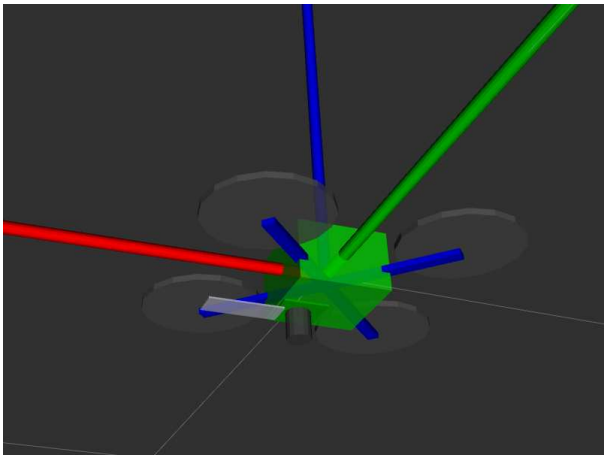
```
<arg name="direction_z" default="down"/>
<arg name="direction_y" default="backward"/>
```

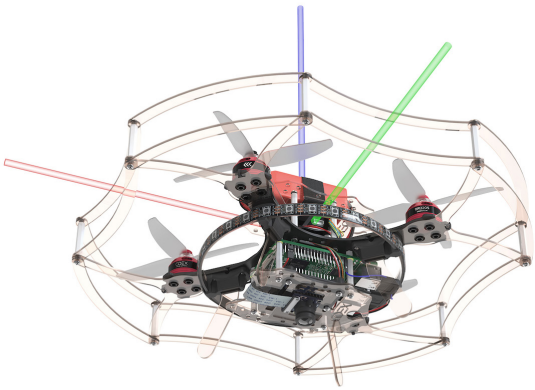




Camera faces downward, cable goes forward

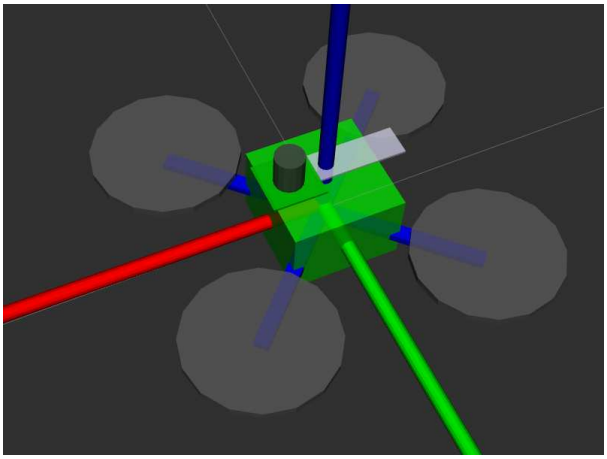
```
<arg name="direction_z" default="down"/>
<arg name="direction_y" default="forward"/>
```

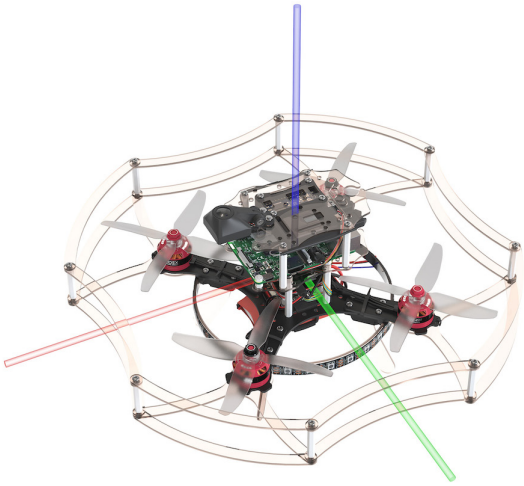




Camera faces upward, cable goes backward

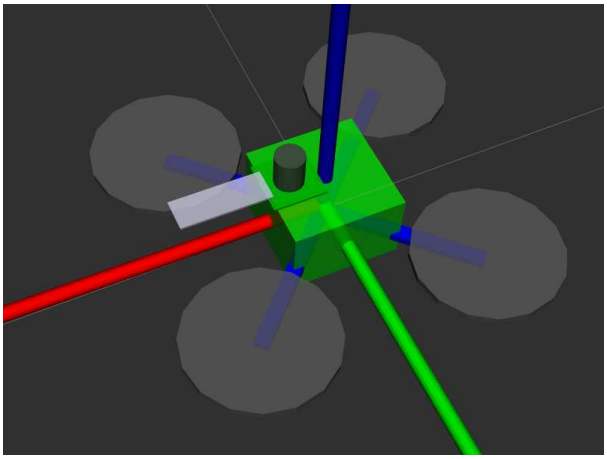
```
<arg name="direction_z" default="up"/>
<arg name="direction_y" default="backward"/>
```

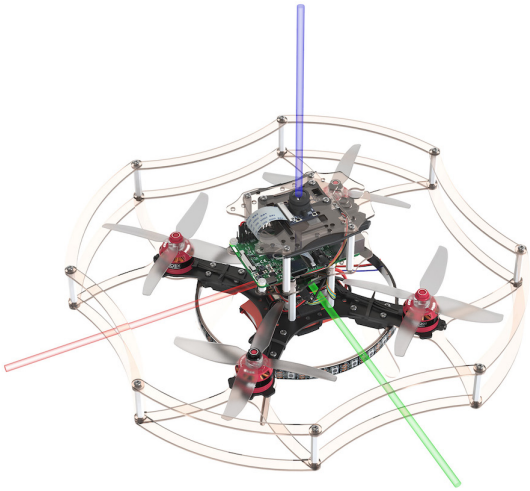




Camera faces upward, cable goes forward

```
<arg name="direction_z" default="up"/>
<arg name="direction_y" default="forward"/>
```





> The `selfcheck.py` utility will describe your current camera setup in a human-readable fashion. Be sure to check whether this description corresponds to your actual camera position.

Custom camera position

It's possible to set arbitrary camera position and orientation. In order to do that uncomment node, marked as `Template for custom camera orientation`:

```
<!-- Template for custom camera orientation -->
<!-- Camera position and orientation are represented by base_link -> main_camer
<!-- static_transform_publisher arguments: x y z yaw pitch roll frame_id child_
<node pkg="tf2_ros" type="static_transform_publisher" name="main_camera_frame"
```

This line describes how the camera is positioned relative to the drone body. Technically, it creates a static transform between the `base_link` frame (which corresponds to the flight controller housing) and the camera (`main_camera_optical`) in the following format:

```
shift_x shift_y shift_z yaw_angle pitch_angle roll_angle
```

Camera frame (that is, frame of reference) is aligned as follows:

- **x** points to the right side of the image;
- **y** points to the bottom of the image;
- **z** points away from the camera matrix plane.

Shifts are set in meters, angles are in radians. You can check the transform for correctness using rviz.

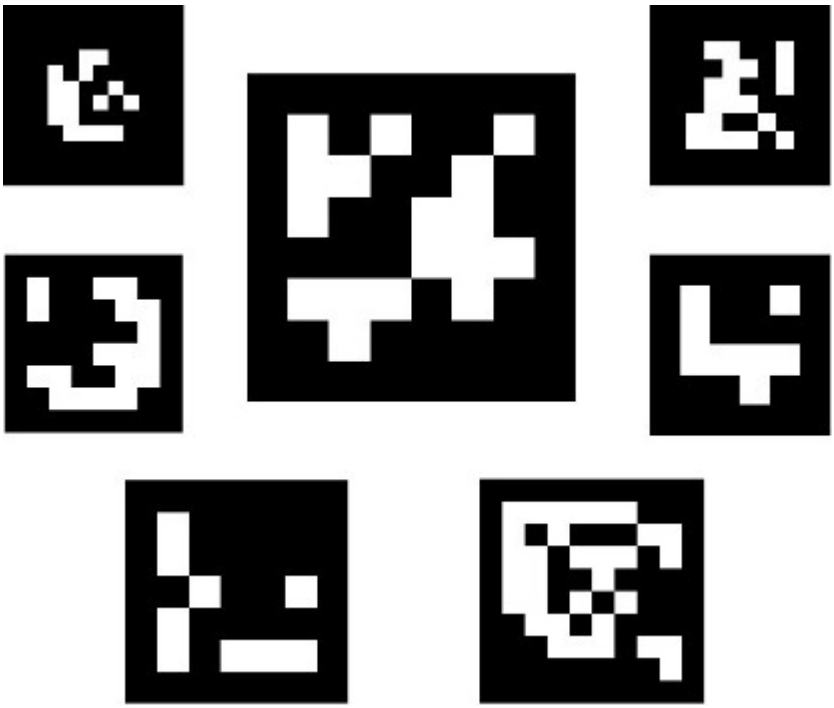
Calibration

To improve the quality of computer vision related algorithms it's recommended to perform camera calibration, which is described in the appropriate article.

ArUco markers

ArUco markers are commonly used for vision-based position estimation.

Examples of ArUco markers:



> Use the most matte paper for printing visual markers. Glossy paper may glitter in the light, severely deteriorating the quality of recognition.

For rapid generation of markers for printing, you may use an online tool: http://chev.me/arucogen/.

Clover Raspberry Pi image contains a pre-installed `aruco_pose` ROS package, which can be used for marker detection.

Modes of operation

There are several preconfigured modes of operation for ArUco markers on the Clover drone:

- single marker detection and navigation;
- map-based navigation.

> Additional documentation for the `aruco_pose` ROS package is available on GitHub.

ArUco marker detection

> The following applies to image versions **0.22** and up. Older documentation is still available for for version **0.20**.

> Marker detection requires the camera module to be correctly plugged in and configured.

`aruco_detect` module detects ArUco markers and publishes their positions in ROS topics and as TF frames.

This is useful in conjunction with other positioning systems, such as GPS, Optical Flow, PX4Flow, visual odometry, ultrasonic (Marvelmind) or UWB-based (Pozyx) localization.

Using this module along with map-based navigation is also possible.

Setup

Set the `aruco` argument in `~/catkin_ws/src/clover/clover/launch/clover.launch` to `true`:

```
<arg name="aruco" default="true"/>
```

For enabling detection set the `aruco_detect` argument in `~/catkin_ws/src/clover/clover/launch/aruco.launch` to `true`:

```
<arg name="aruco_detect" default="true"/>
```

For the module to work correctly the following arguments should also be set:

```
<arg name="placement" default="floor"/> <!-- markers' placement, explained belo
<arg name="length" default="0.33"/>     <!-- length of a single marker, in mete
```

`placement` argument should be set to:

- `floor` if *all* markers are on the ground;
- `ceiling` if *all* markers are on the ceiling;
- an empty string otherwise.

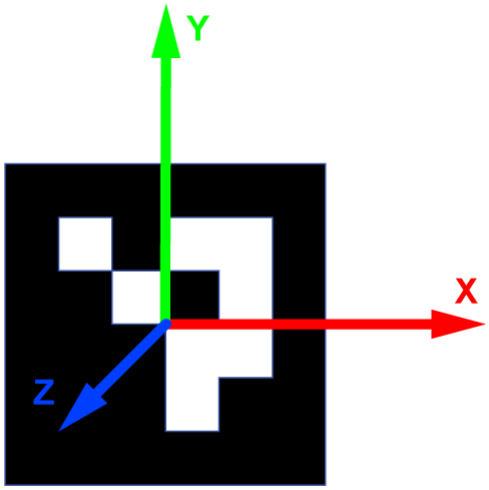
You may specify length for each marker individually by using the `length_override` parameter of the node `aruco_detect`:

```
<param name="length_override/3" value="0.1"/>    <!-- marker with id=3 has a si
<param name="length_override/17" value="0.25"/>  <!-- marker with id=17 has a s
```

Coordinate system

Each marker has its own coordinate systems. It is aligned as follows:

- the **x** axis points to the right side of the marker;
- the **y** axis points to the top side of the marker;
- the **z** axis points outwards from the plane of the marker



Working with detected markers

Navigation within the marker-based TF frames is possible with `simple_offboard` node.

Sample code to fly to a point 1 metre above marker with id 5:

```
navigate(frame_id='aruco_5', x=0, y=0, z=1)
```

Sample code to fly to a point 1 metre to the left and 2 metres above marker with id 7:

```
navigate(frame_id='aruco_7', x=-1, y=0, z=2)
```

Note that if the required marker isn't detected for 0.5 seconds after the `navigate` command, the command will be ignored.

These frames may also be used in other services that accept TF frames (like `get_telemetry`). The following code will get the drone's position relative to the marker with id 3:

```
telem = get_telemetry(frame_id='aruco_3')
```

Note that if the required marker isn't detected for 0.5 seconds, the `telem.x`, `telem.y`, `telem.z`, `telem.yaw` fields will contain `NaN`.

Handling marker detection in Python

The following snippet shows how to read the `aruco_detect/markers` topic in Python:

```
import rospy
from aruco_pose.msg import MarkerArray
rospy.init_node('my_node')

# ...

def markers_callback(msg):
    print('Detected markers:'):
    for marker in msg.markers:
        print('Marker: %s' % marker)

# Create a Subscription object. Each time a message is posted in aruco_detect/m
rospy.Subscriber('aruco_detect/markers', MarkerArray, markers_callback)

# ...

rospy.spin()
```

Each message contains the marker ID, its corner points on the image and its position relative to the camera.

Suggested reading: map-based navigation

Map-based navigation with ArUco markers

> The following applies to image versions **0.22** and up. Older documentation is still available for for version **0.20**.

> Marker detection requires the camera module to be correctly plugged in and configured.

> We recommend using our custom PX4 firmware.

`aruco_map` module detects whole ArUco-based maps. Map-based navigation is possible using vision position estimate (VPE).

Configuration

Set the `aruco` argument in `~/catkin_ws/src/clover/clover/launch/clover.launch` to `true`:

```
<arg name="aruco" default="true"/>
```

In order to enable map detection set `aruco_map` and `aruco_detect` arguments to `true` in `~/catkin_ws/src/clover/clover/launch/aruco.launch`:

```
<arg name="aruco_detect" default="true"/>
<arg name="aruco_map" default="true"/>
```

Set `aruco_vpe` to `true` to publish detected camera position to the flight controller as VPE data:

```
<arg name="aruco_vpe" default="true"/>
```

Marker map definition

Map is defined in a text file; each line has the following format:

```
marker_id marker_size x y z z_angle y_angle x_angle
```

`N_angle` is the angle of rotation along the `N` axis in radians.

Map files are located at the `~/catkin_ws/src/clover/aruco_pose/map` directory. Map file name is defined in the `map` argument:

```
<arg name="map" default="map.txt"/>
```

Some map examples are provided in the directory.

Grid maps may be generated using the `genmap.py` script:

```
rosrun aruco_pose genmap.py length x y dist_x dist_y first -o test_map.txt
```

`length` is the size of each marker, `x` is the marker count along the *x* axis, `y` is the marker count along the *y* axis, `dist_x` is the distance between the centers of adjacent markers along the *x* axis, `dist_y` is the distance between the centers of the *y* axis, `first` is the ID of the first marker (top left marker, unless `--bottom-left` is specified), `test_map.txt` is the name of the generated map file. The optional `--bottom-left` parameter changes the numbering of markers, making the bottom left marker the first one.

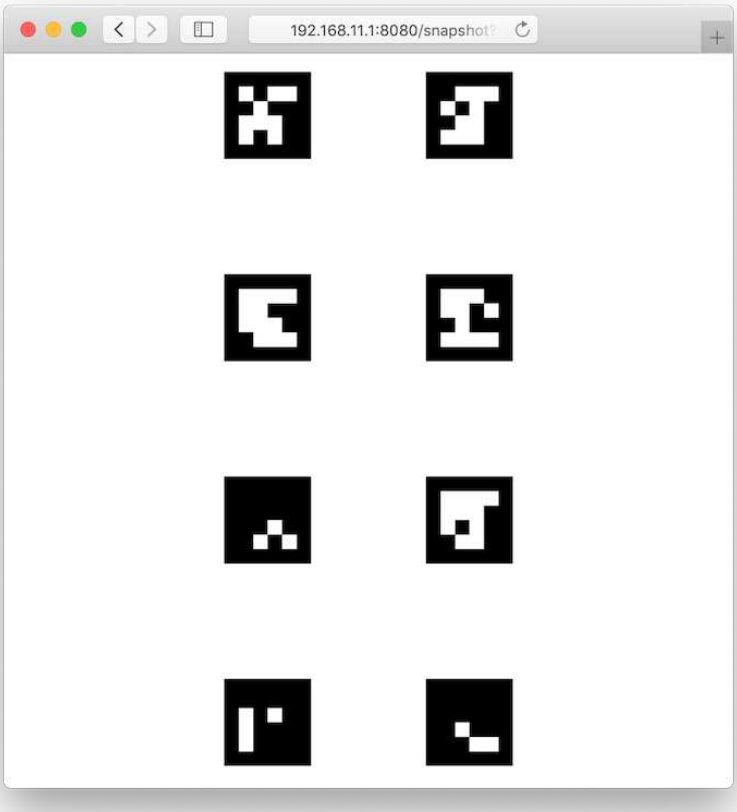
Usage example:

```
rosrun aruco_pose genmap.py 0.33 2 4 1 1 0 -o test_map.txt
```

Additional information on the utility can be obtained using `-h` key: `rosrun aruco_pose genmap.py -h`.

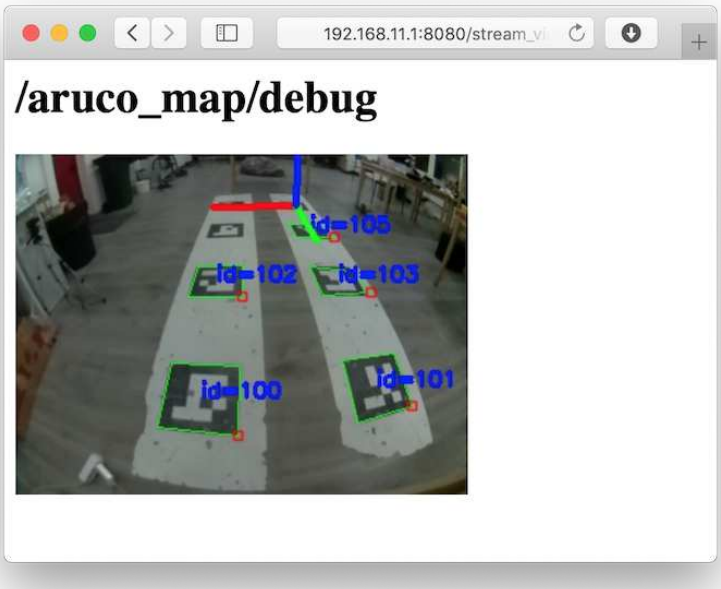
Checking the map

The currently active map is posted in the `/aruco_map/image` ROS topic. It can be viewed using web_video_server by opening the following link: http://192.168.11.1:8080/snapshot?topic=/aruco_map/image



Current estimated pose (relative to the detected map) is published in the `aruco_map/pose` ROS topic. If the VPE is disabled, the `aruco_map` TF frame is created; otherwise, the `aruco_map_detected` frame is created instead. Visualization for the current map is also posted in the `aruco_map/visualization` ROS topic; it may be visualized in rviz.

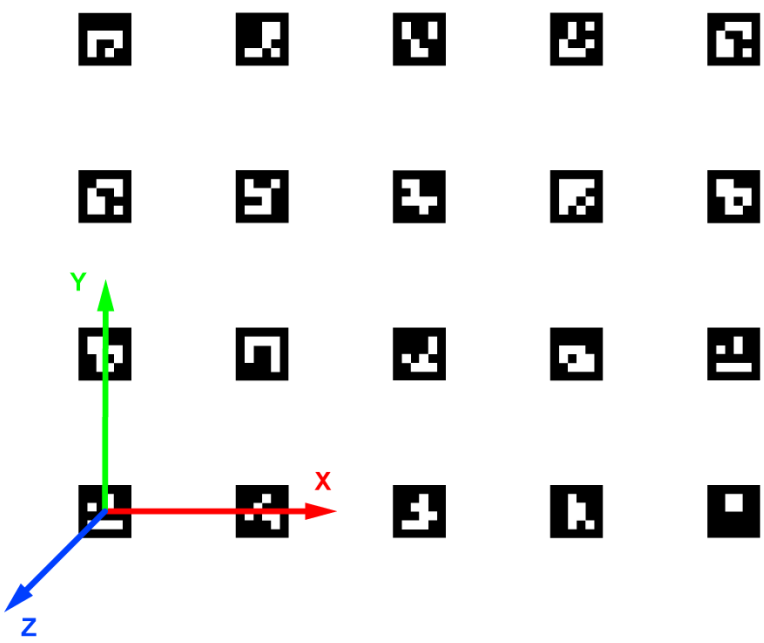
An easy to understand detected map visualization is posted in the `aruco_map/debug` ROS topic (live view is available on http://192.168.11.1:8080/stream_viewer?topic=/aruco_map/debug):



Coordinate system

The marker map adheres to the ROS coordinate system convention, using the ENU coordinate system:

- the **x** axis points to the right side of the map;
- the **y** axis points to the top side of the map;
- the **z** axis points outwards from the plane of the marker



VPE setup

In order to enable vision position estimation you should use the following PX4 parameters.

If you're using **LPE** (`SYS_MC_EST_GROUP` parameter is set to `local_position_estimator,attitude_estimator_q`):

- `LPE_FUSION` should have `vision position` and `land detector` flags set. We suggest unsetting the `baro` flag for indoor flights.
- External heading (yaw) weight: `ATT_W_EXT_HDG` = 0.5.
- External heading (yaw) mode: `ATT_EXT_HDG_M` = 1 (`Vision`).
- Vision position standard deviations: `LPE_VIS_XY` = 0.1 m, `LPE_VIS_Z` = 0.1 m.
- `LPE_VIS_DELAY` = 0 sec.

If you're using **EKF2** estimator (`SYS_MC_EST_GROUP` parameter is set to `ekf2`), make sure the following is set:

- `EKF2_AID_MASK` should have `vision position fusion` and `vision yaw fusion` flags set.
- Vision angle observations noise: `EKF2_EVA_NOISE` = 0.1 rad.
- Vision position observations noise: `EKF2_EVP_NOISE` = 0.1 m.
- `EKF2_EV_DELAY` = 0.

> We recommend using **LPE** for marker-based navigation.

You may use the `selfcheck.py` utility to check your settings.

> In order to use LPE with the Pixhawk v1 hardware you should download the `px4fmu-v2_lpe.px4` firmware

Flight

If the setup is done correctly, the drone will hold its position in `POSCTL` and `OFFBOARD` flight modes automatically.

You will also be able to use `navigate` , `set_position` and `set_velocity` ROS services for autonomous flights. In order to fly to a specific coordinate within the ArUco map you should use the `aruco_map` frame:

```
# Takeoff should be performed in the "body" frame; "aruco_map" frame will appea
navigate(x=0, y=0, z=2, frame_id='body', speed=0.5, auto_arm=True) # Takeoff ar

time.sleep(5)

# Fly to the (2, 2) point on the marker field while being 2 metres above it
navigate(x=2, y=2, z=2, speed=1, frame_id='aruco_map')
```

Using a specific marker frame

Starting with the image version 0.18, the drone also can fly relative to a marker in the map, even if it is not currently visible. Like with single-marker navigation, this works by setting the frame_id parameter to `aruco_ID`, where ID is the desired marker number.

The following code will move the drone to the point 1 meter above the center of marker 5:

```
navigate(frame_id='aruco_5', x=0, y=0, z=1)
```

Additional settings

If the drone's position is not stable when VPE is used, try increasing the *P* term in the velocity PID regulator: increase the `MPC_XY_VEL_P` and `MPC_Z_VEL_P` parameters.

If the drone's altitude is not stable, try increasing the `MPC_Z_VEL_P` parameter and adjusting hover thrust via `MPC_THR_HOVER`.

Placing markers on the ceiling



In order to navigate using markers on the ceiling, mount the onboard camera so that it points up and adjust the camera frame accordingly.

You should also set the `placement` parameter to `ceiling` in `~/catkin_ws/src/clover/clover/launch/aruco.launch`:

```
<arg name="placement" default="ceiling"/>
```

With such a camera orientation the Optical Flow technology cannot work, so it should be disabled in the `~/catkin_ws/src/clover/clover/launch/clover.launch` file:

```
<arg name="optical_flow" default="false"/>
```

Such setup will flip the `aruco_map` frame (making its **z** axis point downward). Thus, in order to fly 2 metres below ceiling, the `z` argument for the `navigate` service should be set to 2:

```
navigate(x=1, y=1.1, z=2, speed=0.5, frame_id='aruco_map')
```

Use of Optical Flow

Running the "Optical Flow" function offers the possibility of POSCTL flight mode, and autonomous flight operating on a camera pointed downwards that detects changes of ground texture.

Enabling

> For Optical Flow to work it's required that the laser rangefinder is connected and configured.

Enable Optical Flow in the file `~/catkin_ws/src/clover/clover/launch/clover.launch`:

```
<arg name="optical_flow" default="true"/>
```

Optical Flow publishes data in `/mavros/px4flow/raw/send` topic. In the topic `/optical_flow/debug` is also published a visualization, that can be viewed with web_video_server.

> Correct connection and setup of the camera module is needed for proper functioning.

Setup of the flight controller

> Suggested parameters are applied automatically in our custom PX4 firmware.

When using **EKF2** (parameter `SYS_MC_EST_GROUP` = `ekf2`):

- `EKF2_AID_MASK` – flag 'use optical flow' is on.
- `EKF2_OF_DELAY` – 0.
- `EKF2_OF_QMIN` – 10.
- `EKF2_OF_N_MIN` – 0.05.
- `EKF2_OF_N_MAX` - 0.2.
- `SENS_FLOW_ROT` – No rotation.
- `SENS_FLOW_MAXHGT` – 4.0 (for the rangefinder VL53L1X)
- `SENS_FLOW_MINHGT` – 0.0 (for the rangefinder VL53L1X)
- Optional: `EKF2_HGT_MODE` – range sensor (cf. rangefinder setup).

When using **LPE** (parameter `SYS_MC_EST_GROUP` = `local_position_estimator, attitude_estimator_q`):

- `LPE_FUSION` – flags 'fuse optical flow' and 'flow gyro compensation' are on.
- `LPE_FLW_QMIN` – 10.
- `LPE_FLW_SCALE` – 1.0.

- `LPE_FLW_R` – 0.2.
- `LPE_FLW_RR` – 0.0.
- `SENS_FLOW_ROT` – No rotation.
- `SENS_FLOW_MAXHGT` – 4.0 (for the rangefinder VL53L1X)
- `SENS_FLOW_MINHGT` – 0.0 (for the rangefinder VL53L1X)
- Optional: `LPE_FUSION` – flag 'pub agl as lpos down' is on (see rangefinder setup.

The `selfcheck.py` utility will help you verify that all settings are correctly set.

POSCTL flight

Setup POSCTL to be one of PX4 flight modes and then select POSCTL.

Autonomous flight

The module simple_offboard enables autonomous flight.

Example of take off and leveling at 1.5m above the ground:

```
navigate(z=1.5, frame_id='body', auto_arm=True)
```

Flying forward for 1m:

```
navigate(x=1.5, frame_id='body')
```

Navigation using ArUco-markers and using VPE are available when using Optical Flow.

Additional settings

For additional background on PID tuning, see the PID tuning guide.

If the copter has an unstable position, try to increase the *P* coefficient of speed PID controller - parameters are `MPC_XY_VEL_P` and `MPC_Z_VEL_P`.

If the copter has an unstable height, try increasing `MPC_Z_VEL_P` coefficient or getting better hover throttle - `MPC_THR_HOVER`.

If the copter is consistently yawing, try:

- recalibrate gyroscopes;
- recalibrate magnetometer;
- different values for `EKF2_MAG_TYPE` parameter, that indicates how data from the magnetometer is used in EKF2;
- changing values of `EKF2_MAG_NOISE`, `EKF2_GYR_NOISE`, `EKF2_GYR_B_NOISE` parameters.

> For better results perform gyro calibration directly before taking off, using appropriate snippet.

If the copter's height is deviating, try:

- increasing the value of `MPC_Z_VEL_P` coefficient;
- change the value of `MPC_THR_HOVER` parameter;
- add `MPC_ALT_MODE` = 2 (Terrain following).

When using Optical Flow, the maximal horizontal speed is further limited. This is an indirect influence of the parameter `SENS_FLOW_MAXR` (maximal reliable "angular speed" of the optical flow). In normal flight mode, control loops will be adjusted so that Optical Flow values do not exceed 50% of this parameter.

Errors

If errors like `EKF INTERNAL CHECKS` occur, try to restart EKF2. To do so, enter in the MAVLink-console:

```
ekf2 stop
ekf2 start
```

Autonomous flight

> The following applies to image versions **0.24** and up. Older documentation is available for for version **0.23**.

The `simple_offboard` module of the `clover` package is intended for simplified programming of the autonomous drone flight (`OFFBOARD` flight mode). It allows setting the desired flight tasks, and automatically transforms coordinates between frames.

`simple_offboard` is a high level system for interacting with the flight controller. For a more low level system, see mavros.

Main services are `get_telemetry` (receive telemetry data), `navigate` (fly to a given point along a straight line), `navigate_global` (fly to a point specified as latitude and longitude along a straight line), `land` (switch to landing mode).

Python usage

You need to create proxies for services before calling them. Use the following template for your programs:

```
import rospy
from clover import srv
from std_srvs.srv import Trigger

rospy.init_node('flight') # 'flight' is name of your ROS node

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_altitude = rospy.ServiceProxy('set_altitude', srv.SetAltitude)
set_yaw = rospy.ServiceProxy('set_yaw', srv.SetYaw)
set_yaw_rate = rospy.ServiceProxy('set_yaw_rate', srv.SetYawRate)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)
```

Unused proxy functions may be removed from the code.

API description

> Omitted numeric parameters are set to 0.

get_telemetry

Obtains complete telemetry of the drone.

Parameters:

- `frame_id` – frame for values `x`, `y`, `z`, `vx`, `vy`, `vz`. Example: `map`, `body`, `aruco_map`. Default value: `map`.

Response format:

- `frame_id` — frame;
- `connected` – whether there is a connection to FCU;
- `armed` - drone arming state (armed if true);
- `mode` – current flight mode;
- `x, y, z` — local position of the drone *(m)*;
- `lat, lon` – drone latitude and longitude *(degrees)*, requires GPS module;
- `alt` – altitude in the global coordinate system (according to WGS-84 standard, not AMSL!), requires GPS module;
- `vx, vy, vz` – drone velocity *(m/s)*;
- `roll` – roll angle *(radians)*;
- `pitch` – pitch angle *(radians)*;
- `yaw` — yaw angle *(radians)*;
- `roll_rate` – angular roll velocity *(rad/s)*;
- `pitch_rate` — angular pitch velocity *(rad/s)*;
- `yaw_rate` – angular yaw velocity *(rad/s)*;
- `voltage` – total battery voltage *(V)*;
- `cell_voltage` – battery cell voltage *(V)*.

> Fields that are unavailable for any reason will contain the `NaN` value.

Displaying drone coordinates `x`, `y` and `z` in the local system of coordinates:

```
telemetry = get_telemetry()
print(telemetry.x, telemetry.y, telemetry.z)
```

Displaying drone altitude relative to the ArUco map:

```
telemetry = get_telemetry(frame_id='aruco_map')
print(telemetry.z)
```

Checking global position availability:

```
import math
if not math.isnan(get_telemetry().lat):
    print('Global position is available')
else:
    print('No global position')
```

Output of current telemetry (command line):

```
rosservice call /get_telemetry "{frame_id: ''}"
```

navigate

Fly to the designated point in a straight line.

Parameters:

- `x`, `y`, `z` — coordinates *(m)*;
- `yaw` — yaw angle *(radians)*;
- `speed` – flight speed (setpoint speed) *(m/s)*;
- `auto_arm` – switch the drone to `OFFBOARD` mode and arm automatically (**the drone will take off**);
- `frame_id` – coordinate system for values `x`, `y`, `z` and `yaw`. Example: `map`, `body`, `aruco_map`. Default value: `map`.

> If you don't want to change your current yaw set the `yaw` parameter to `NaN` (angular velocity by default is 0).

Ascending to the altitude of 1.5 m with the climb rate of 0.5 m/s:

```
navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
```

Flying in a straight line to point 5:0 (altitude 2) in the local system of coordinates at the speed of 0.8 m/s (yaw is set to 0):

```
navigate(x=5, y=0, z=3, speed=0.8)
```

Flying to point 5:0 without changing the yaw angle:

```
navigate(x=5, y=0, z=3, speed=0.8, yaw=float('nan'))
```

Flying 3 m to the right from the drone:

```
navigate(x=0, y=-3, z=0, speed=1, frame_id='body')
```

Flying 2 m to the left from the last navigation target:

```
navigate(x=0, y=2, z=0, speed=1, frame_id='navigate_target')
```

Turn 90 degrees clockwise:

```
navigate(yaw=math.radians(-90), frame_id='body')
```

Flying to point 3:2 (with the altitude of 2 m) in the ArUco map coordinate system with the speed of 1 m/s:

```
navigate(x=3, y=2, z=2, speed=1, frame_id='aruco_map')
```

Ascending to the altitude of 2 m (command line):

```
rosservice call /navigate "{x: 0.0, y: 0.0, z: 2, yaw: 0.0, speed: 0.5, frame_
```

> Consider using the `navigate_target` frame instead of `body` for missions that primarily use relative movements forward/back/left/right. This negates inaccuracies in relative point calculations.

navigate_global

Flying in a straight line to a point in the global coordinate system (latitude/longitude).

Parameters:

- `lat`, `lon` — latitude and longitude *(degrees)*;
- `z` — altitude *(m)*;
- `yaw` — yaw angle *(radians)*;
- `speed` – flight speed (setpoint speed) *(m/s)*;
- `auto_arm` – switch the drone to `OFFBOARD` and arm automatically (**the drone will take off**);
- `frame_id` – coordinate system for `z` and `yaw` (Default value: `map`).

> If you don't want to change your current yaw set the `yaw` parameter to `NaN` (angular velocity by default is 0).

Flying to a global point at the speed of 5 m/s, while maintaining current altitude (`yaw` will be set to 0, the drone will face East):

```
navigate_global(lat=55.707033, lon=37.725010, z=0, speed=5, frame_id='body')
```

Flying to a global point without changing the yaw angle:

```
navigate_global(lat=55.707033, lon=37.725010, z=0, speed=5, yaw=float('nan'), f
```

Flying to a global point (command line):

```
rosservice call /navigate_global "{lat: 55.707033, lon: 37.725010, z: 0.0, yaw
```

set_altitude

Change the desired flight altitude. The service is used to set the altitude and its coordinate system independently, after calling `navigate` or `set_position`.

Parameters:

- `z` – flight altitude *(m)*;
- `frame_id` – coordinate system for computing the altitude.

Set the desired altitude to 2 m relative to the floor:

```
set_altitude(z=2, frame_id='terrain')
```

Set the desired altitude to 1 m relative to the ArUco map:

```
set_altitude(z=1, frame_id='aruco_map')
```

set_yaw

Change the desired yaw angle (and its coordinate system), keeping the previous command in effect.

Parameters:

- `yaw` — yaw angle *(radians)*;
- `frame_id` – coordinate system for computing the yaw.

Rotate by 90 degrees clockwise (the previous command continues):

```
set_yaw(yaw=math.radians(-90), frame_id='body')
```

Set the desired yaw angle to zero relative to the ArUco map:

```
set_yaw(yaw=0, frame_id='aruco_map')
```

Stop yaw rotation (caused by `set_yaw_rate` call):

```
set_yaw(yaw=float('nan'))
```

set_yaw_rate

The the desired angular yaw velocity, keeping the previous command in effect.

Parameters:

- `yaw_rate` – angular yaw velocity *(rad/s)*;

The positive direction of `yaw_rate` rotation (when viewed from the top) is counterclockwise.

Start yaw rotation at 0.5 rad/s (the previous command continues):

```
set_yaw_rate(yaw_rate=0.5)
```

set_position

Set the setpoint for position and yaw. This service may be used to specify the continuous flow of target points, for example, for flying along complex trajectories (circular, arcuate, etc.).

> Use the `navigate` higher-level service to fly to a point in a straight line or to perform takeoff.

Parameters:

- `x`, `y`, `z` — point coordinates *(m)*;
- `yaw` — yaw angle *(radians)*;
- `auto_arm` – switch the drone to `OFFBOARD` and arm automatically (**the drone will take off**);
- `frame_id` – coordinate system for `x`, `y`, `z` and `yaw` parameters (Default value: `map`).

Hovering on the spot:

```
set_position(frame_id='body')
```

Assigning the target point 3 m above the current position:

```
set_position(x=0, y=0, z=3, frame_id='body')
```

Assigning the target point 1 m ahead of the current position:

```
set_position(x=1, y=0, z=0, frame_id='body')
```

set_velocity

Set speed and yaw setpoints.

- `vx`, `vy`, `vz` – flight speed *(m/s)*;
- `yaw` — yaw angle *(radians)*;
- `auto_arm` – switch the drone to `OFFBOARD` and arm automatically (**the drone will take off**);
- `frame_id` – coordinate system for `vx`, `vy`, `vz` and `yaw` (Default value: `map`).

> Parameter `frame_id` specifies only the orientation of the resulting velocity vector, but not its length.

Flying forward (relative to the drone) at the speed of 1 m/s:

```
set_velocity(vx=1, vy=0.0, vz=0, frame_id='body')
```

set_attitude

Set roll, pitch, yaw and throttle level (similar to the `STABILIZED` mode). This service may be used for lower level control of the drone behavior, or controlling the drone when no reliable data on its position is available.

Parameters:

- `roll`, `pitch`, `yaw` – requested roll, pitch, and yaw angle *(radians)*;
- `thrust` — throttle level, ranges from 0 (no throttle, propellers are stopped) to 1 (full throttle).
- `auto_arm` – switch the drone to `OFFBOARD` mode and arm automatically (**the drone will take off**);

- `frame_id` – coordinate system for `yaw` (Default value: `map`).

set_rates

Set roll, pitch, and yaw rates and the throttle level (similar to the `ACRO` mode). This is the lowest drone control level (excluding direct control of motor rotation speed). This service may be used to automatically perform aerobatic tricks (e.g., flips).

Parameters:

- `roll_rate`, `pitch_rate`, `yaw_rate` – pitch, roll, and yaw rates *(rad/s)*;
- `thrust` — throttle level, ranges from 0 (no throttle, propellers are stopped) to 1 (full throttle).
- `auto_arm` – switch the drone to `OFFBOARD` and arm automatically (**the drone will take off**);

The positive direction of `yaw_rate` rotation (when viewed from the top) is counterclockwise, `pitch_rate` rotation is forward, `roll_rate` rotation is to the left.

land

Switch the drone to landing mode (`AUTO.LAND` or similar).

> Set the `COM_DISARM_LAND` PX4 parameter to a value greater than 0 to enable automatic disarm after landing.

Landing the drone:

```
res = land()

if res.success:
    print('drone is landing')
```

Landing the drone (command line):

```
rosservice call /land "{}"
```

> In recent PX4 versions, the vehicle will be switched out of LAND mode to manual mode, if the remote control sticks are moved significantly.

release

If it's necessary to pause sending setpoint messages, use the `simple_offboard/release` service:

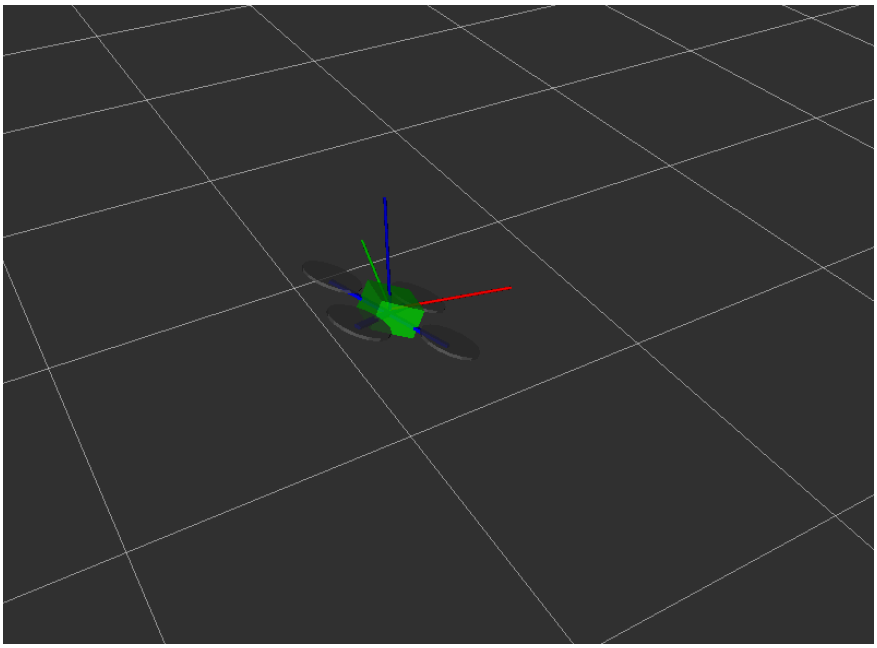
```
release = rospy.ServiceProxy('simple_offboard/release', Trigger)

release()
```

Additional materials

- ArUco-based position estimation and navigation.
- Program samples and snippets.

Coordinate systems (frames)



Main frames in the `clover` package:

- `map` has its origin at the flight controller initialization point and may be considered stationary. It is shown as a white grid on the image above;
- `base_link` is rigidly bound to the drone. It is shown by the simplified drone model on the image above;
- `body` is bound to the drone, but its Z axis points up regardless of the drone's pitch and roll. It is shown by the red, blue and green lines in the illustration;
- `navigate_target` is bound to the current navigation target (as set by the navigate service);
- `terrain` is bound to the floor at the current drone position (see the set_altitude service);
- `setpoint` is current position setpoint;
- `main_camera_optical` is the coordinate system, linked to the main camera;

Additional frames become available when ArUco positioning system is active:

- `aruco_map` is bound to the currently active ArUco map;
- `aruco_N` is bound to the marker with ID=N.

> Frames that are bound to the drone are oriented according to the ROS convention: the X axis points forward, Y to the left, and Z up.

3D visualization of the coordinate systems can be viewed using rviz.

tf2

Read more at http://wiki.ros.org/tf2

tf2 ROS package is used extensively in the Clover platform. tf2 is a set of libraries for C++, Python and other programming languages that are used to work with the frames. Internally, ROS nodes publish `TransformStamped` messages to `/tf` topic with transforms between frames at certain points in time.

The `simple_offboard` node can be used to request the drone position in an arbitrary frame by setting the `frame_id` argument appropriately in a call to `get_telemetry` service.

tf2 can be used from Python to transform coordinates (for objects like PoseStamped and PointStamped) from one frame to another

Code examples

Python

#

Function to fly to a point and wait for copter's arrival:

```
import math

def navigate_wait(x=0, y=0, z=0, yaw=float('nan'), speed=0.5, frame_id='', auto
    navigate(x=x, y=y, z=z, yaw=yaw, speed=speed, frame_id=frame_id, auto_arm=a

    while not rospy.is_shutdown():
        telem = get_telemetry(frame_id='navigate_target')
        if math.sqrt(telem.x ** 2 + telem.y ** 2 + telem.z ** 2) < tolerance:
            break
        rospy.sleep(0.2)
```

This function utilizes `navigate_target` frame for computing the distance to the target.

Using the function for flying to the point x=3, y=2, z=1 in marker's map:

```
navigate_wait(x=3, y=2, z=1, frame_id='aruco_map')
```

This function can be used for taking off as well:

```
navigate_wait(z=1, frame_id='body', auto_arm=True)
```

#

Land and wait until the copter lands:

```
land()
while get_telemetry().armed:
    rospy.sleep(0.2)
```

Usage:

```
land_wait()
```

#

Wait for copter's arrival to the navigate target:

```
import math

def wait_arrival(tolerance=0.2):
    while not rospy.is_shutdown():
        telem = get_telemetry(frame_id='navigate_target')
        if math.sqrt(telem.x ** 2 + telem.y ** 2 + telem.z ** 2) < tolerance:
            break
        rospy.sleep(0.2)
```

#

Calculate the distance between two points (**important**: the points are to be in the same coordinate system):

```
import math

def get_distance(x1, y1, z1, x2, y2, z2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2 + (z1 - z2) ** 2)
```

#

Approximation of distance (in meters) between two global coordinates (latitude/longitude):

```
import math

def get_distance_global(lat1, lon1, lat2, lon2):
    return math.hypot(lat1 - lat2, lon1 - lon2) * 1.113195e5
```

#

Disarm the drone (propellers will stop, **the drone will fall down**):

```
# Declaring a proxy:
from mavros_msgs.srv import CommandBool
arming = rospy.ServiceProxy('mavros/cmd/arming', CommandBool)

# ...

arming(False)  # disarm
```

#

Transform the position (`PoseStamped`) from one coordinate system to another using tf2:

```
import tf2_ros
import tf2_geometry_msgs
from geometry_msgs.msg import PoseStamped

tf_buffer = tf2_ros.Buffer()
tf_listener = tf2_ros.TransformListener(tf_buffer)

# ...

# Create PoseStamped object (or get it from a topic):
pose = PoseStamped()
pose.header.frame_id = 'map' # coordinate frame, in which the position is speci
pose.header.stamp = rospy.get_rostime() # the time for which the position is sp
pose.pose.position.x = 1
pose.pose.position.y = 2
pose.pose.position.z = 3
pose.pose.orientation.w = 1

frame_id = 'base_link' # target coordinate frame
transform_timeout = rospy.Duration(0.2) # timeout for transformation

# Transform the position from the old frame to the new one:
new_pose = tf_buffer.transform(pose, frame_id, transform_timeout)
```

#

Determine whether the copter is turned upside-down:

```
PI_2 = math.pi / 2
telem = get_telemetry()

flipped = abs(telem.roll) > PI_2 or abs(telem.pitch) > PI_2
```

#

Calculate the copter horizontal angle:

```
PI_2 = math.pi / 2
telem = get_telemetry()

flipped = not -PI_2 <= telem.roll <= PI_2 or not -PI_2 <= telem.pitch <= PI_2
angle_to_horizon = math.atan(math.hypot(math.tan(telem.roll), math.tan(telem.pi
if flipped:
    angle_to_horizon = math.pi - angle_to_horizon
```

#

Fly along a circular path:

```
RADIUS = 0.6  # m
SPEED = 0.3 # rad / s

start = get_telemetry()
start_stamp = rospy.get_rostime()

r = rospy.Rate(10)

while not rospy.is_shutdown():
    angle = (rospy.get_rostime() - start_stamp).to_sec() * SPEED
    x = start.x + math.sin(angle) * RADIUS
    y = start.y + math.cos(angle) * RADIUS
    set_position(x=x, y=y, z=start.z)

    r.sleep()
```

#

Repeat an action at a frequency of 10 Hz:

```
r = rospy.Rate(10)
while not rospy.is_shutdown():
    # Do anything
    r.sleep()
```

#

An example of subscription to a topic from MAVROS:

```
from geometry_msgs.msg import PoseStamped, TwistStamped
from sensor_msgs.msg import BatteryState
from mavros_msgs.msg import RCIn

def pose_update(pose):
    # Processing new data of copter's position
    pass

rospy.Subscriber('mavros/local_position/pose', PoseStamped, pose_update)
rospy.Subscriber('mavros/local_position/velocity', TwistStamped, velocity_updat
rospy.Subscriber('mavros/battery', BatteryState, battery_update)
rospy.Subscriber('mavros/rc/in', RCIn, rc_callback)

rospy.spin()
```

Information about MAVROS topics is available at the link.

#

Send an arbitrary MAVLink message to the copter:

```
from mavros_msgs.msg import Mavlink
from mavros import mavlink
from pymavlink import mavutil

mavlink_pub = rospy.Publisher('mavlink/to', Mavlink, queue_size=1)

# Sending a HEARTBEAT message:
msg = mavutil.mavlink.MAVLink_heartbeat_message(mavutil.mavlink.MAV_TYPE_GCS, (
msg.pack(mavutil.mavlink.MAVLink('', 2, 1))
ros_msg = mavlink.convert_to_rosmsg(msg)

mavlink_pub.publish(ros_msg)
```

#

Subscribe to all MAVLink messages from the flight controller and decode them:

```
from mavros_msgs.msg import Mavlink
from mavros import mavlink
from pymavlink import mavutil

link = mavutil.mavlink.MAVLink('', 255, 1)

def mavlink_cb(msg):
    mav_msg = link.decode(mavlink.convert_to_bytes(msg))
    print('msgid =', msg.msgid, mav_msg) # print message id and parsed message

mavlink_sub = rospy.Subscriber('mavlink/from', Mavlink, mavlink_cb)

rospy.spin()
```

#

React to the drone's mode switching (may be used for starting an autonomous flight, see example):

```
from mavros_msgs.msg import RCIn

# Called when new data is received from the transmitter
def rc_callback(data):
    # React on toggling the mode of the transmitter
    if data.channels[5] < 1100:
        # ...
        pass
    elif data.channels[5] > 1900:
        # ...
        pass
    else:
        # ...
        pass

# Creating a subscriber for the topic with the data from the transmitter
rospy.Subscriber('mavros/rc/in', RCIn, rc_callback)

rospy.spin()
```

#

Change the flight mode to arbitrary one:

```
from mavros_msgs.srv import SetMode

set_mode = rospy.ServiceProxy('mavros/set_mode', SetMode)

# ...

set_mode(custom_mode='STABILIZED')
```

#

Flip:

```
import math

PI_2 = math.pi / 2

def flip():
    start = get_telemetry()  # memorize starting position

    set_rates(thrust=1)  # bump up
    rospy.sleep(0.2)

    set_rates(pitch_rate=30, thrust=0.2)  # pitch flip
    # set_rates(roll_rate=30, thrust=0.2)  # roll flip

    while True:
        telem = get_telemetry()
        flipped = abs(telem.roll) > PI_2 or abs(telem.pitch) > PI_2
        if flipped:
            break

    rospy.loginfo('finish flip')
    set_position(x=start.x, y=start.y, z=start.z, yaw=start.yaw)  # finish fli

print(navigate(z=2, speed=1, frame_id='body', auto_arm=True))  # take off
rospy.sleep(10)

rospy.loginfo('flip')
flip()
```

Requires the special PX4 firmware for Clover. Before running a flip, take all necessary safety precautions.

#

Perform gyro calibration:

```
from pymavlink import mavutil
from mavros_msgs.srv import CommandLong
from mavros_msgs.msg import State

send_command = rospy.ServiceProxy('mavros/cmd/command', CommandLong)

def calibrate_gyro():
    rospy.loginfo('Calibrate gyro')
    if not send_command(command=mavutil.mavlink.MAV_CMD_PREFLIGHT_CALIBRATION,
        return False

    calibrating = False
    while not rospy.is_shutdown():
        state = rospy.wait_for_message('mavros/state', State)
        if state.system_status == mavutil.mavlink.MAV_STATE_CALIBRATING or stat
            calibrating = True
        elif calibrating and state.system_status == mavutil.mavlink.MAV_STATE_S
            rospy.loginfo('Calibrating finished')
            return True

calibrate_gyro()
```

> In process of calibration the drone should not be moved.

#

Enable and disable ArUco markers recognition dynamically (for example, for saving CPU resources):

```
import rospy
import dynamic_reconfigure.client

rospy.init_node('flight')
aruco_client = dynamic_reconfigure.client.Client('aruco_detect')

# Turn markers recognition off
aruco_client.update_configuration({'enabled': False})

rospy.sleep(5)

# Turn markers recognition on
aruco_client.update_configuration({'enabled': True})
```

#

Enable and disable Optical Flow dynamically:

```
import rospy
import dynamic_reconfigure.client

rospy.init_node('flight')
flow_client = dynamic_reconfigure.client.Client('optical_flow')

# Turn Optical Flow off
flow_client.update_configuration({'enabled': False})

rospy.sleep(5)

# Turn Optical Flow on
flow_client.update_configuration({'enabled': True})
```

#

> For RPi image version > 0.23.

Change the used ArUco markers map file dynamically:

```
import rospy
import dynamic_reconfigure.client

rospy.init_node('flight')
map_client = dynamic_reconfigure.client.Client('aruco_map')

map_client.update_configuration({'map': '/home/pi/catkin_ws/src/clover/aruco_p
```

#

Wait for global position to appear (finishing GPS receiver initialization):

```
import math

while not rospy.is_shutdown():
    if math.isfinite(get_telemetry().lat):
        break
    rospy.sleep(0.2)
```

#

Read flight controller's parameter:

```
from mavros_msgs.srv import ParamGet
from mavros_msgs.msg import ParamValue

param_get = rospy.ServiceProxy('mavros/param/get', ParamGet)

# Read parameter of type INT
value = param_get(param_id='COM_FLTMODE1').value.integer

# Read parameter of type FLOAT
value = param_get(param_id='MPC_Z_P').value.float
```

#

Set flight controller's parameter:

```
from mavros_msgs.srv import ParamSet
from mavros_msgs.msg import ParamValue

param_set = rospy.ServiceProxy('mavros/param/set', ParamSet)

# Set parameter of type INT:
param_set(param_id='COM_FLTMODE1', value=ParamValue(integer=8))

# Set parameter of type FLOAT:
param_set(param_id='MPC_Z_P', value=ParamValue(real=1.5))
```

#

Check, if the code is running inside a Gazebo simulation:

```
is_simulation = rospy.get_param('/use_sim_time', False)
```

#

You can move a physical object (link) in Gazebo (as well as change its velocity) using the `gazebo/set_link_state` service (of the type `SetLinkState`). For example, if you add a cube to the world (link `unit_box::link`), you can move it to the point (1, 2, 3):

```
import rospy
from geometry_msgs.msg import Point, Pose, Quaternion
from gazebo_msgs.srv import SetLinkState
from gazebo_msgs.msg import LinkState

rospy.init_node('flight')

set_link_state = rospy.ServiceProxy('gazebo/set_link_state', SetLinkState)

# Change link's position
set_link_state(LinkState(link_name='unit_box::link', pose=Pose(position=Point(1
```

> Simple object animation in Gazebo can be implemented using actors.

Working with a laser rangefinder

> Documentation for the image, versions, starting with **0.20**. For older versions refer to documentation for version **0.19**.

VL53L1X Rangefinder

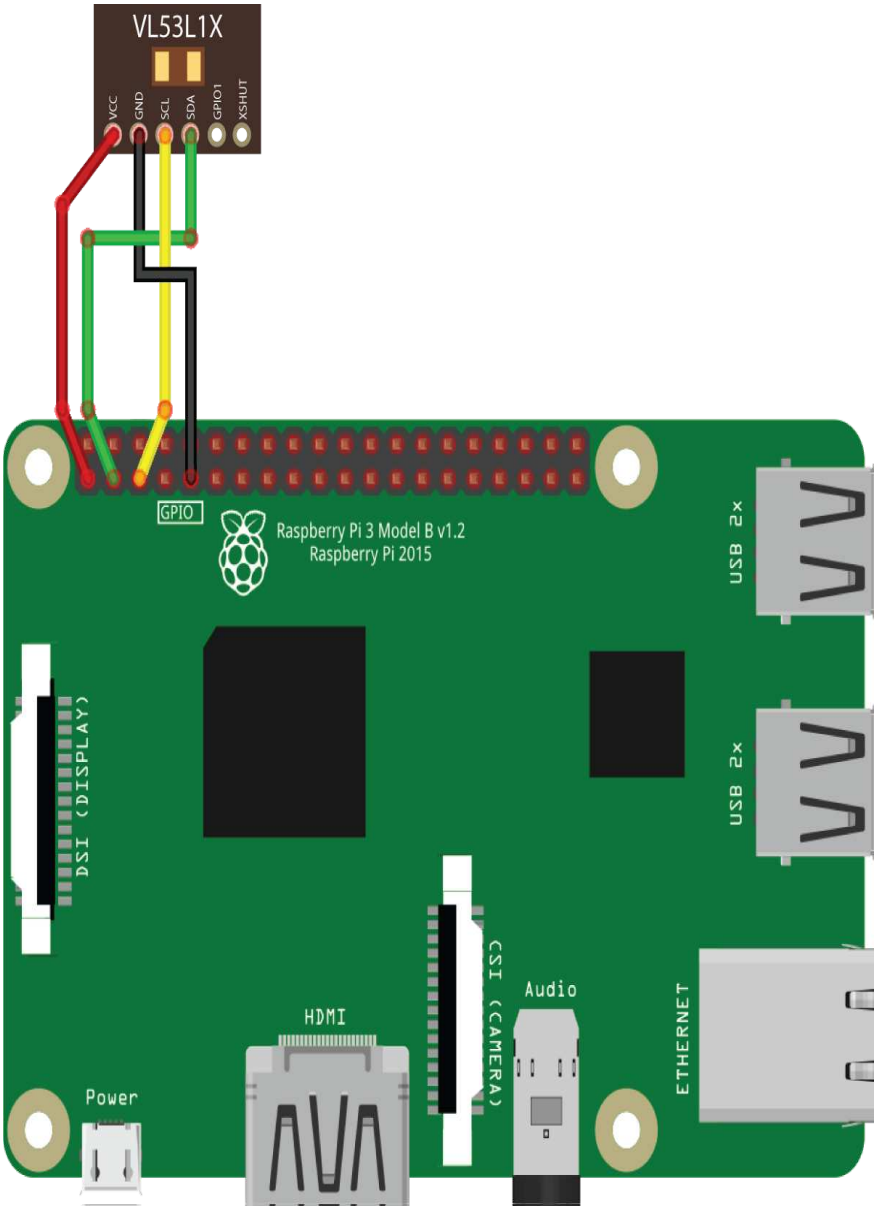
The rangefinder model recommended for Clover is STM VL53L1X. This rangefinder can measure distances from 0 to 4 m while ensuring high measurement accuracy.

The image for Raspberry Pi contains pre-installed corresponding ROS driver.

Connecting to Raspberry Pi

> Before using the rangefinder, please remove the protective film from it.

Connect the rangefinder to the 3V, GND, SCL and SDA pins via the I²C interface:



If the pin marked GND is occupied, you can use any other ground pin (look at the pinout for reference).

> You can connect several peripheral devices via the I²C interface simultaneously. Use a parallel connection for that.

Enabling the rangefinder

Connect via SSH and edit file `~/catkin_ws/src/clover/clover/launch/clover.launch` so that the VL53L1X driver is enabled:

```
<arg name="rangefinder_vl53l1x" default="true"/>
```

By default, the rangefinder driver sends the data to Pixhawk via the `/rangefinder/range` topic. To view data from the topic, use the following command:

```
rostopic echo /rangefinder/range
```

PX4 settings

> We recommend using our custom PX4 firmware for Clover for best laser rangefinder support.

PX4 should be properly configured to use the rangefinder data.

Set the following parameters when EKF2 is used (`SYS_MC_EST_GROUP` = `ekf2`):

- `EKF2_HGT_MODE` = `2` (Range sensor) – for flights over horizontal floor;
- `EKF2_RNG_AID` = `1` (Range aid enabled) – in other cases.

Set the following parameters when LPE is used (`SYS_MC_EST_GROUP` = `local_position_estimator, attitude_estimator_q`):

- The "pub agl as lpos down" flag should be set in the `LPE_FUSION` parameter – for flights over horizontal floor.

Receiving data in Python

In order to receive data from the topic, create a subscriber:

```
import rospy
from sensor_msgs.msg import Range

rospy.init_node('flight')

def range_callback(msg):
    # Process data from the rangefinder
    print('Rangefinder distance:', msg.range)

rospy.Subscriber('rangefinder/range', Range, range_callback)

rospy.spin()
```

Also it's possible to read one rangefinder measurement at a time:

```
from sensor_msgs.msg import Range

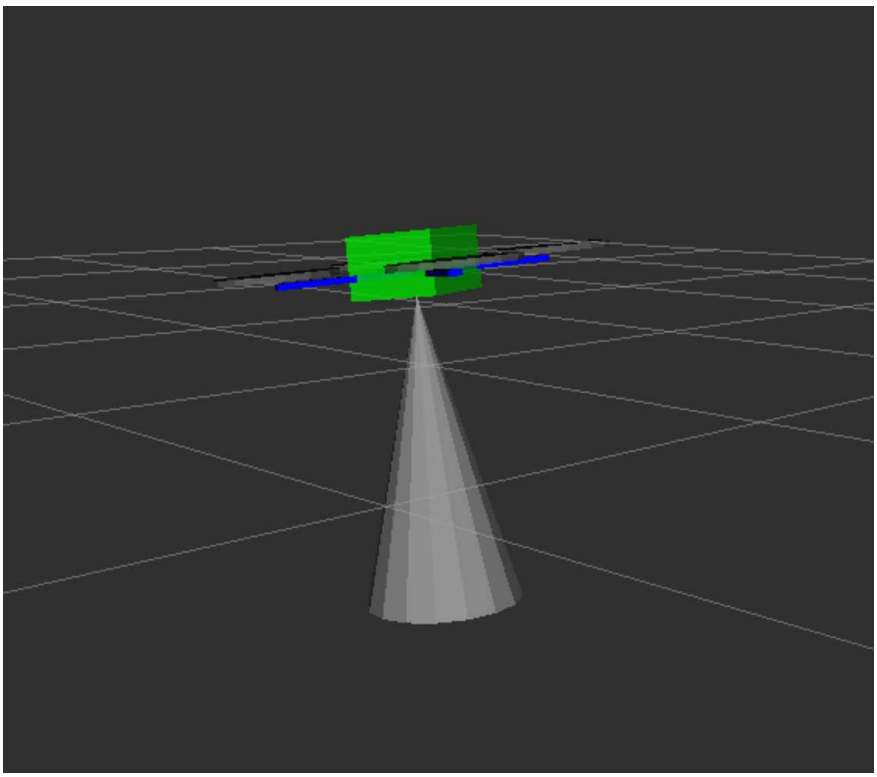
# ...

dist = rospy.wait_for_message('rangefinder/range', Range).range
```

Data visualization

You may use rqt_multiplot tool to plot rangefinder data.

rviz may be used for data visualization. To do this, add a topic of the `sensor_msgs/Range` type to visualization:



Read more about rviz and rqt.

Working with a LED strip

> Documentation for the image versions, starting with **0.21**. For older versions refer to documentation for version **0.20**.

Clover drone kits contain addressable LED strips based on *ws281x* drivers. Each LED may be set to any one of 16 million possible colors (each color is encoded by a 24-bit number). This allows making the Clover flight more spectacular, as well as show flight modes, display stages of current user program, and notify the pilot of other events.



Our Raspberry Pi image contains preinstalled modules for interfacing with the LED strip. They allow the user to:

- manage LED strip effects and animations (high-level control);
- control individual LED colors (low-level control);
- configure the strip to display flight events.

> LED strip can consume a lot of power! Powering it from a Raspberry Pi may overload the computer's power circuitry. Consider using a separate BEC as a power source.

High-level control

1. Connect the +5v and GND leads of your LED to a power source and the DIN (data in) lead to GPIO21. Consult the assembly instructions for details.
2. Enable LED strip support in `~/catkin_ws/src/clover/clover/launch/clover.launch`:

```
<arg name="led" default="true"/>
```

3. Configure the *ws281x* parameters in `~/catkin_ws/src/clover/clover/launch/led.launch`. Change the number of addressable LEDs and the GPIO pin used for control to match your configuration:

```
<arg name="led_count" default="58"/>  <!-- Number of LEDs in the strip --
<arg name="gpio_pin" default="21"/>   <!-- GPIO data pin -->
```

High-level interface allows changing current effect (or animation) on the strip. It is exposed as the `/led/set_effect` service. It has the following arguments:

- `effect` is the name of requested effect.
- `r`, `g`, `b` are RGB components of effect color. Each component is an integer in a 0 to 255 range.

Currently available effects are:

- `fill` (or an empty string) fills the whole strip with the requested color;
- `blink` turns the strip on and off, setting it to the requested color;
- `blink_fast` is the same, but faster;
- `fade` fades smoothly to the requested color;
- `wipe` fills the strip with the requested color one LED at a time;
- `flash` blinks twice and returns to the previous effect;
- `rainbow` creates a rainbow-like shifting effect;
- `rainbow_fill` cycles the strip through rainbow colors, filling the whole strip with the same color.

Python example:

```
import rospy
from clover.srv import SetLEDEffect

rospy.init_node('flight')

set_effect = rospy.ServiceProxy('led/set_effect', SetLEDEffect)  # define proxy

set_effect(r=255, g=0, b=0)  # fill strip with red color
rospy.sleep(2)

set_effect(r=0, g=100, b=0)  # fill strip with green color
rospy.sleep(2)

set_effect(effect='fade', r=0, g=0, b=255)  # fade to blue color
rospy.sleep(2)

set_effect(effect='flash', r=255, g=0, b=0)  # flash twice with red color
rospy.sleep(5)

set_effect(effect='blink', r=255, g=255, b=255)  # blink with white color
rospy.sleep(5)

set_effect(effect='rainbow')  # show rainbow
```

You can also set colors from your Bash shell:

```
rosservice call /led/set_effect "{effect: 'fade', r: 0, g: 0, b: 255}"
```

```
rosservice call /led/set_effect "{effect: 'rainbow'}"
```

Configuring event visualizations

It is possible to display current flight controller status and notify the user about some events with the LED strip. This is configured in the `~/catkin_ws/src/clover/clover/launch/led.launch` file in the *events effects table* section. Here is a sample configuration:

```
startup: { r: 255, g: 255, b: 255 }
connected: { effect: rainbow }
disconnected: { effect: blink, r: 255, g: 50, b: 50 }
<!-- ... -->
```

The left part is one of the possible events that the strip reacts to. The right part contains the effect description that you want to execute for this event.

Here is the list of supported events:

Event	Description	Default effect
`startup`	Clover system startup	White
`connected`	Successful connection to flight controller	Rainbow
`disconnected`	Connection to flight controller lost	Red blink
`armed`	Transition to Armed state	
`disarmed`	Transition to Disarmed state	
`acro`	Acro mode	Orange
`stabilized`	Stabilized mode	Green
`altctl`	Altitude mode	Yellow
`posctl`	Position mode	Blue
`offboard`	Offboard mode	Violet
`rattitude`, `mission`, `rtl`, `land`	Corresponding mode	
`error`	Error in one of ROS nodes or in the flight controller (*ERROR* message in `/rosout`)	Red flash
`low_battery`	Low battery (threshold is set in the `threshold` parameter)	Red fast blink

> You need to calibrate the power sensor for the `low_battery` event to work properly.

In order to disable LED strip notifications set `led_notify` argument in `~/catkin_ws/src/clover/clover/launch/led.launch` to `false`:

```
<arg name="led_notify" default="false"/>
```

Low-level control

You can use the `/led/set_leds` ROS service to control individual LEDs. It accepts an array of LED indices and desired colors.

Python example:

```
import rospy
from led_msgs.srv import SetLEDs
from led_msgs.msg import LEDStateArray, LEDState

rospy.init_node('flight')

set_leds = rospy.ServiceProxy('led/set_leds', SetLEDs)  # define proxy to ROS s

# switch LEDs number 0, 1 and 2 to red, green and blue color:
set_leds([LEDState(0, 255, 0, 0), LEDState(1, 0, 255, 0), LEDState(2, 0, 0, 255
```

You can also use this service from the your Bash shell:

```
rosservice call /led/set_leds "leds:
- index: 0
  r: 50
  g: 100
  b: 200"
```

Current LED strip state is published in the `/led/state` ROS topic. You can view the contents of this topic from your Bash shell:

```
rostopic echo /led/state
```

Using the same topic you can get the configured number os LEDs, using Python:

```
led_count = len(rospy.wait_for_message('led/state', LEDStateArray, timeout=10)
```

Working with GPIO

A GPIO (General-Purpose Input/Output) pin is a programmable digital signal pin on a circuit board or a microcontroller that may act as an input or an output. Raspberry Pi has a set of easily accessible GPIO pins, some of which have hardware PWM.

> Use the pinout for figuring out, which Raspberry Pi's pins support GPIO and PWM.

The `pigpio` library for interfacing with the GPIO pins is already preinstalled on the RPi image. To interact with this library, run the appropriate daemon:

```
sudo systemctl start pigpiod.service
```

To enable automatic launch of the daemon, run:

```
sudo systemctl enable pigpiod.service
```

> `pigpiod` may interfere with LED strip if configured improperly. Make sure that the strip is connected to GPIO21. On image versions lower than 0.17 change the service start string in `/lib/systemd/system/pigpiod.service` to `ExecStart=/usr/bin/pigpiod -l -t 0 -x 0x0FFF3FF0`.

Example of working with the library:

```
import time
import pigpio

# initializing connection to pigpiod
pi = pigpio.pi()

# set pin 11 mode for output
pi.set_mode(11, pigpio.OUTPUT)

# set signal of pin 11 to high
pi.write(11, 1)

time.sleep(2)

# set signal on pin 11 to low
pi.write(11, 0)

# ...

# setting pin 12 mode for input
pi.set_mode(12, pigpio.INPUT)

# read the state of pin 12
level = pi.read(12)
```

To find out the pins' numbers, consult the Raspberry Pi pinout.

Connecting servos

Servo motors are typically controlled with PWM signal. Extreme positions of servos are reached with signal widths approximately equal to 1000 and 2000 μs. Values for a specific servo can be determined experimentally.

Connect the signal wire of the servo to one of GPIO-pins of the Raspberry. To control a servo connected to the pin 13 use a code like this:

```
import time
import pigpio

pi = pigpio.pi()

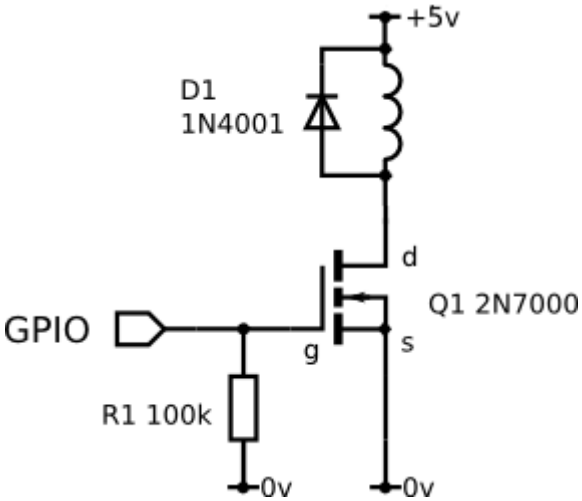
# set mode of pin 13 to output
pi.set_mode(13, pigpio.OUTPUT)

# set pin 13 to output PWM signal 1000 us
pi.set_servo_pulsewidth(13, 1000)

time.sleep(2)

# set pin 13 to output PWM signal 2000 us
pi.set_servo_pulsewidth(13, 2000)
```

Connecting an electromagnet



To connect an electromagnet use a field-effect transistor (MOSFET). Connect the MOSFET to one of GPIO-pins of the Raspberry Pi. To control the magnet connected to the pin 15 use a code like this:

```
import time
import pigpio

pi = pigpio.pi()

# set mode of pin 15 for output
pi.set_mode(15, pigpio.OUTPUT)

# enable the magnet
pi.write(15, 1)

time.sleep(2)

# disable the magnet
pi.write(15, 0)
```

A more comprehensive description of the Raspberry Pi GPIO pins and additional examples of circuits are available at the Embedded Linux wiki.

Working with the ultrasonic distance sensor

Ultrasonic distance sensor (*"sonar"*) is a distance sensor based on the principle of measuring the time of a sound wave (about 40 kHz) propagation to the obstacle and back. The sonar can measure the distance up to 1.5 – 3 m with the accuracy of several centimeters.

HC-SR04 distance sensor

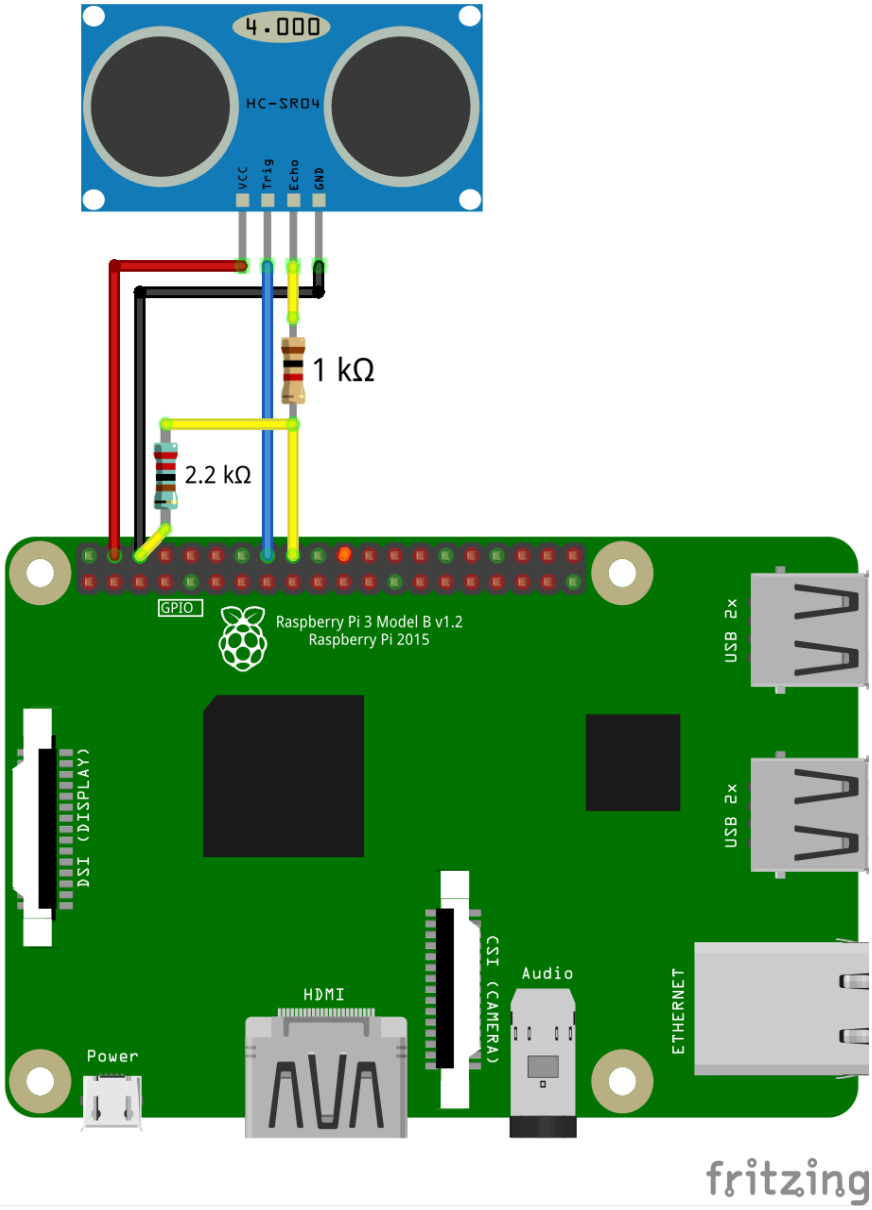


Installation

The distance sensor is attached to the body using double-sided tape. For obtaining acceptable results, the use of vibro-insulation is required. A piece of PU foam may be used for vibro-insulation.

Connection

Connect HC-SR04 to Raspberry Pi according to the connection diagram. Use 1.0 and 2.2 kΩ resistors and any free GPIO pins, e.g., 23 and 24:



> Instead of a 2.2 kΩ resistor, you can use two 1 kΩ resistors connected in series.

> There are several interchangeable pins **GND** and **VCC 5V** on Raspberry Pi. Use the [pinout] (https://pinout.xyz) to find them.

Reading the data

To read the data from distance sensor HC-SR04 library for working with GPIO is used – `pigpio`. This library is pre-installed in the Clover image, starting with version **v0.14**. For older versions of the image, use an installation guide.

To work with `pigpio`, start appropriate daemon:

```
sudo systemctl start pigpiod.service
```

You can also enable `pigpiod` auto launch on system startup:

```
sudo systemctl enable pigpiod.service
```

Thus, it becomes possible to interact with the `pigpiod` daemon from Python:

```
import pigpio
pi = pigpio.pi()
```

> See detailed description of Python API in `pigpio` documentation.

An example of the code for reading data from HC-SR04:

```
import time
import threading
import pigpio

TRIG = 23  # pin connected to the Trig pin of the sonar
ECHO = 24  # pin connected to the Echo pin of the sonar

pi = pigpio.pi()
done = threading.Event()

def rise(gpio, level, tick):
    global high
    high = tick

def fall(gpio, level, tick):
    global low
    low = tick - high
    done.set()

def read_distance():
    global low
    done.clear()
    pi.gpio_trigger(TRIG, 50, 1)
    if done.wait(timeout=5):
        return low / 58.0 / 100.0

pi.set_mode(TRIG, pigpio.OUTPUT)
pi.set_mode(ECHO, pigpio.INPUT)
pi.callback(ECHO, pigpio.RISING_EDGE, rise)
pi.callback(ECHO, pigpio.FALLING_EDGE, fall)

while True:
    # Reading the distance:
    print(read_distance())
```

Filtering the data

To filter (smooth out) the data and delete outliers, Kalman filter or a simple median filter can be used. An example of median filtering implementation:

```
import collections
import numpy

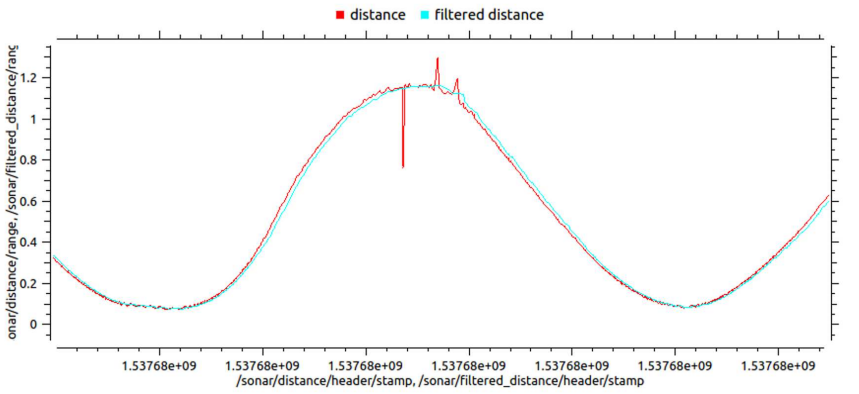
# ...

history = collections.deque(maxlen=10)  # 10 - number of samples for averaging

def read_distance_filtered():
    history.append(read_distance())
    return numpy.median(history)

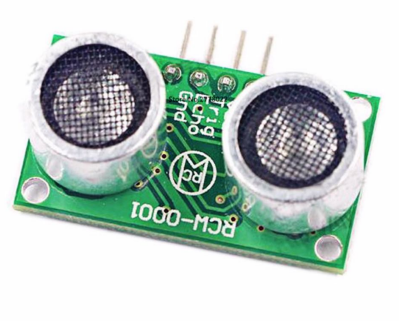
while True:
    print(read_distance_filtered())
```

An example of charts of initial and filtered data:



The source code of the ROS-node used for building the chart can be found on Gist.

RCW-0001 distance sensor



The RCW-0001 distance sensor is compatible with distance sensor HC-SR04. Use the instruction above to connect and work with it.

Flight

An example of a flight program with the use of simple_offboard, which makes the copter fly forward until the connected ultrasonic distance sensor detects an obstacle:

```
set_velocity(vx=0.5, frame_id='body', auto_arm=True) # flying forward at the ve

while True:
    if read_distance_filtered() < 1:
        # if the obstacle is closer than 1 m, hanging on the spot
        set_position(x=0, y=0, z=0, frame_id='body')
    rospy.sleep(0.1)
```

Working with the camera

> The following applies to the image version **0.24**, which is not yet released. Older documentation is still available for for version **0.23**.

Make sure the camera is enabled in the `~/catkin_ws/src/clover/clover/launch/clover.launch` file:

```
<arg name="main_camera" default="true"/>
```

Also make sure that position and orientation of the camera is correct.

The `clover` service must be restarted after the launch-file has been edited:

```
sudo systemctl restart clover
```

You may use rqt or web_video_server to view the camera stream.

Troubleshooting

If the camera stream is missing, try using the `raspistill` utility to check whether the camera works.

First, stop the `clover` service:

```
sudo systemctl stop clover
```

Then use `raspistill` to capture an image from the camera:

```
raspistill -o test.jpg
```

If it doesn't work, check the camera cable connections and the cable itself. Replace the cable if it is damaged. Also, make sure the camera screws don't touch any components on the camera board.

Camera parameters

Some camera parameters, such as image size, FPS cap, and exposure, may be configured in the `main_camera.launch` file. The list of supported parameters can be found in the cv_camera repository.

Additionally you can specify an arbitrary capture parameter using its OpenCV code. For example, add the following parameters to the camera node to set exposition manually:

```
<param name="property_0_code" value="21"/> <!-- property code 21 is CAP_PROP_AU
<param name="property_0_value" value="0.25"/> <!-- property values are normaliz
<param name="cv_cap_prop_exposure" value="0.3"/> <!-- set exposure to 30% of ma
```

Computer vision

The SD card image comes with a preinstalled OpenCV library, which is commonly used for various computer vision-related tasks. Additional libraries for converting from ROS messages to OpenCV images and back are preinstalled as well.

Python

An example of creating a subscriber for a topic with an image from the main camera for processing with OpenCV:

```
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from clover import long_callback

rospy.init_node('cv')
bridge = CvBridge()

@long_callback
def image_callback(data):
    img = bridge.imgmsg_to_cv2(data, 'bgr8')  # OpenCV image
    # Do any image processing with cv2...

image_sub = rospy.Subscriber('main_camera/image_raw', Image, image_callback)

rospy.spin()
```

> Image processing may take significant time to finish. This can cause an issue in rospy library, which would lead to processing stale camera frames. To solve this problem you need to use `long_callback` decorator from `clover` library, as in the example above.

Limiting CPU usage

When using the `main_camera/image_raw` topic, the script will process the maximum number of frames from the camera, actively utilizing the CPU (up to 100%). In tasks, where processing each camera frame is not critical, you can use the topic, where the frames are published at rate 5 Hz: `main_camera/image_raw_throttled`:

```
image_sub = rospy.Subscriber('main_camera/image_raw_throttled', Image, image_ca
```

Publishing images

To debug image processing, you can publish a separate topic with the processed image:

```
image_pub = rospy.Publisher('~debug', Image)
```

Publishing the processed image:

```
image_pub.publish(bridge.cv2_to_imgmsg(img, 'bgr8'))
```

The published images can be viewed using web_video_server or rqt.

Retrieving one frame

It's possibly to retrieve one camera frame at a time. This method works slower than normal topic subscribing and should not be used when it's necessary to process camera images continuously.

```
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

rospy.init_node('cv')
bridge = CvBridge()

# ...

# Retrieve a frame:
img = bridge.imgmsg_to_cv2(rospy.wait_for_message('main_camera/image_raw', Ima
```

Examples

Working with QR codes

> For high-speed recognition and positioning, it is better to use ArUco markers.

To program actions of the copter for the detection of QR codes you can use the pyZBar. This lib is installed in the last image for Raspberry Pi.

QR codes recognition in Python:

```
import rospy
from pyzbar import pyzbar
import cv2
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
from clover import long_callback

rospy.init_node('cv')
bridge = CvBridge()

@long_callback
def image_callback(msg):
    img = bridge.imgmsg_to_cv2(msg, 'bgr8')
    barcodes = pyzbar.decode(img)
    for barcode in barcodes:
        b_data = barcode.data.decode('utf-8')
        b_type = barcode.type
        (x, y, w, h) = barcode.rect
        xc = x + w/2
        yc = y + h/2
        print('Found {} with data {} with center at x={}, y={}'.format(b_type,

image_sub = rospy.Subscriber('main_camera/image_raw_throttled', Image, image_ca

rospy.spin()
```

See other computer vision examples in the `~/examples` directory of the RPi image.

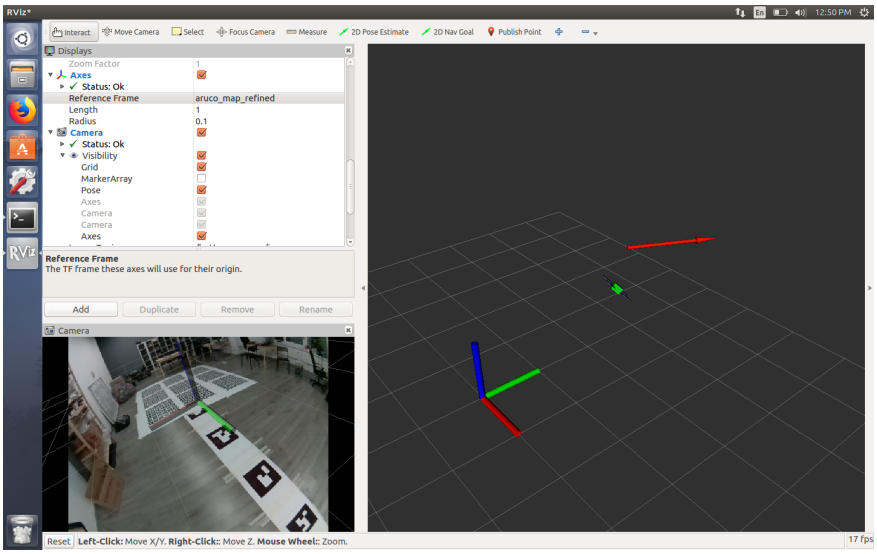
Video recording

To record a video you can use `video_recorder` node from `image_view` package:

```
rosrun image_view video_recorder image:=/main_camera/image_raw
```

The video file will be saved to a file `output.avi`. The `image` argument contains the name of the topic to record.

Using rviz and rqt



The rviz tool allows real-time visualization of all components of the robotic system —the system of coordinates, moving parts, sensors, camera images — on the 3D stage.

rqt is a set of GUI for analyzing and controlling ROS systems. For example, `rqt_image_view` allows viewing topics with images, `rqt_multiplot` allows plot charts by the values in topics, etc.

To use rviz and rqt, a PC running Ubuntu Linux (or a virtual machine such as Parallels Desktop Lite or VirtualBox) is required.

> You can use the preconfigured virtual machine image with ROS and Clover toolkit.

Install package `ros-noetic-desktop-full` or `ros-noetic-desktop` using the installation documentation.

Start rviz

To start the Clover state visualization in real time, connect to it via Wi-Fi (`clover-xxxx`) and run rviz, specifying an appropriate ROS_MASTER_URI:

```
ROS_MASTER_URI=http://192.168.11.1:11311 rviz
```

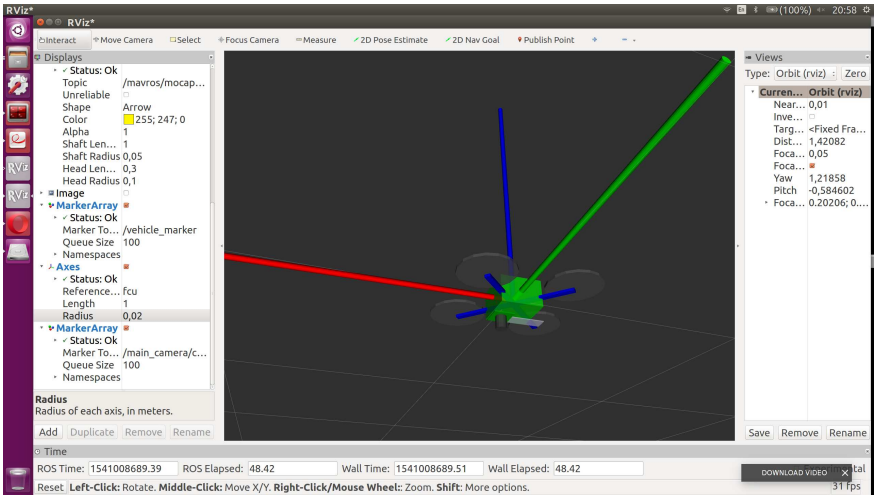
> In case of using a virtual machine for using rviz and other tools it might be necessary to change its network configuration to *bridge* mode (see details for VMware).

Using rviz

Visualization of the copter position

It is recommended to set the `map` frame as a reference frame. To visualize the copter, add visualization markers from topic `/vehicle_markers`. To visualize the camera of the copter, add visualization markers from topic `/main_camera/camera_markers`.

The result of copter and camera visualization is shown below:



Visualization of the environment

You can view a picture with augmented reality from the topic of the main camera `/main_camera/image_raw`.

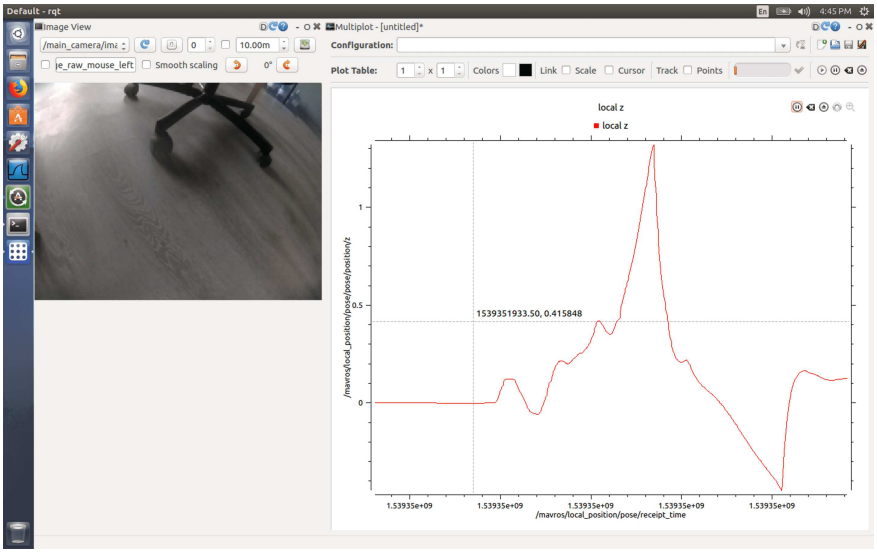
Axis or Grid configured to frame `aruco_map` will visualize the location on the map of ArUco marks.

jsk_rviz_plugins

It is also recommended to install additional useful plugins for rviz jsk_rviz_plugins. This kit allows visualizing topics like `TwistStamped` (velocity) `CameraInfo`, `PolygonArray`, and many more. To install, use command:

```
sudo apt-get install ros-melodic-jsk-visualization
```

Starting the rqt toolkit



To start rqt for monitoring Clover status, use command:

```
ROS_MASTER_URI=http://192.168.11.1:11311 rqt
```

An example of starting a specific plugin (`rqt_image_view`):

```
ROS_MASTER_URI=http://192.168.11.1:11311 rqt_image_view
```

Brief description of useful rqt plugins:

- `rqt_image_view` – viewing images from topics like `sensor_msgs/Image` ;
- `rqt_multiplot` – Building charts from the data from of arbitrary topics (installation: `sudo apt-get install ros-melodic-rqt-multiplot`);
- Bag – working with Bag-files.

Software autorun

> In the image version **0.20** `clever` package and service was renamed to `clover`. See previous version of the article for older images.

systemd

Main documentation: https://wiki.archlinux.org/title/Systemd.

All automatically started Clover software is launched as a `clover.service` systemd service.

The service may be restarted by the `systemctl` command:

```
sudo systemctl restart clover
```

Text output of the software can be viewed using the `journalctl` command:

```
journalctl -u clover
```

To run Clover software directly in the current console session, you can use the `roslaunch` command:

```
sudo systemctl restart clover
roslaunch clover clover.launch
```

You can disable Clover software autolaunch using the `disable` command:

```
sudo systemctl disable clover
```

roslaunch

Main documentation: http://wiki.ros.org/roslaunch.

The list of nodes / programs declared for running is specified in file `/home/pi/catkin_ws/src/clover/clover/launch/clover.launch`.

You can add your own node to the list of automatically launched ones. To do this, place your executable file (e.g. `my_program.py`) into folder `/home/pi/catkin_ws/src/clover/clover`. Then add the start of your node to `clover.launch`, for example:

```
<node name="my_program" pkg="clover" type="my_program.py" output="screen"/>
```

The started file must have *permission* to run:

```
chmod +x my_program.py
```

When scripting languages are used, a shebang should be placed at the beginning of the file, for example:

```
#!/usr/bin/env python3
```

Work with ROS from browser

Using the `roslibjs` library it's possible to work with all the ROS resources (topics, services, parameters) from JavaScript code within the browser, which allows creating various interactive web applications for drone.

All the required software is preinstalled in RPi image for Clover.

Example

An example of a web page, working with `roslib.js`:

```
<html>
    <script src="js/roslib.js"></script>
    <script type="text/javascript">
        // Establish roslibjs connection
        var ros = new ROSLIB.Ros({ url: 'ws://' + location.hostname + ':9090' }

        ros.on('connection', function () {
            // Connection callback
            alert('Connected');
        });

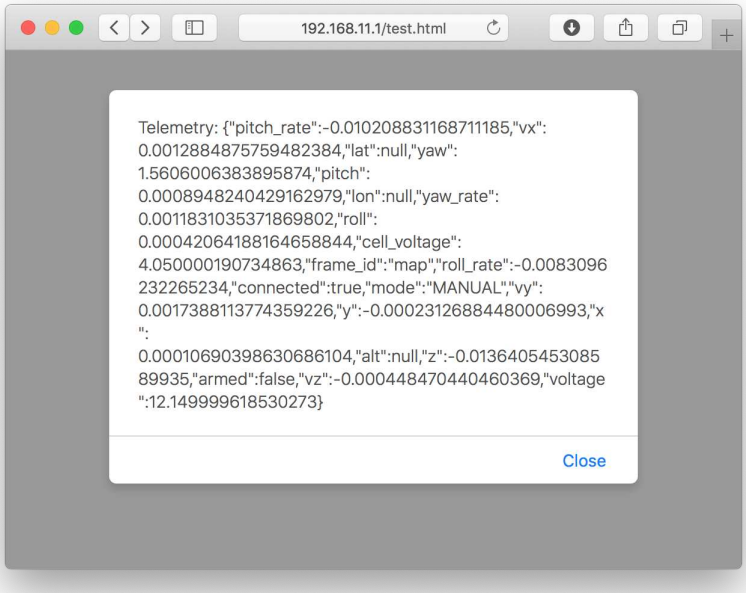
        // Declare get_telemetry service client
        var getTelemetry = new ROSLIB.Service({ ros: ros, name : '/get_telemetr

        // Call get_telemetry
        getTelemetry.callService(new ROSLIB.ServiceRequest({ frame_id: 'map' }),
            // Service respond callback
            alert('Telemetry: ' + JSON.stringify(result));
        });

        // Subscribe to `/mavros/state` topic
        var stateSub = new ROSLIB.Topic({ ros : ros, name : '/mavros/state', me
        stateSub.subscribe(function(msg) {
            // Topic message callback
            console.log('State: ', msg);
        });
    </script>
</html>
```

Taking off, landing and all the rest operations can be implemented in a similar way.

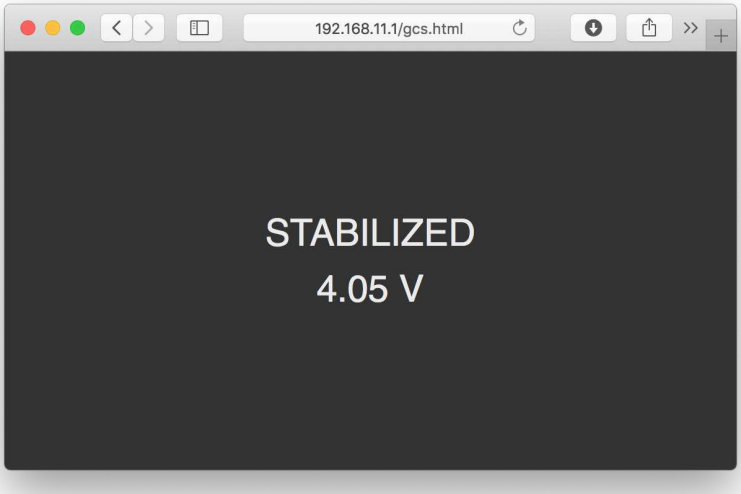
The page should be placed in the `/home/pi/catkin_ws/src/clover/clover/www/` directory. After that, it will be available at `http://192.168.11.1/clover/<page_name>.html`. When the page is opened, browser should show an alert with the drone telemetry and constantly print the state of the flight controller to the console.



See additional information in `roslibjs` tutorial.

Web GCS

See an example of simplified web ground control station on Clover at http://192.168.11.1/clover/gcs.html.



Blocks programming for Clover

Visual blocks programming feature has been added to the RPi image, starting with the version **0.21**. Blocks programming is implemented using Google Blockly library. Blocks programming integration can lower the entry barrier to a minimum.

Configuration

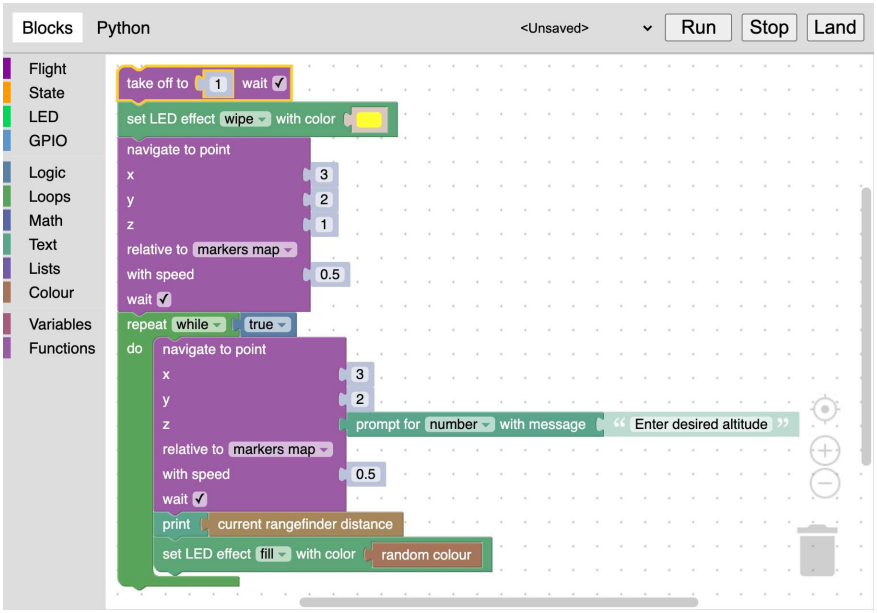
For correct work of the blocks programming, `blocks` argument in the Clover launch-file (`~/catkin_ws/src/clover/clover/launch/clover.launch`) should be equal to `true` :

```
<arg name="blocks" default="true"/>
```

Running

To run Clover's blocks programming interface, connect to Clover's Wi-Fi and go to web-page http://192.168.11.1/clover_blocks/ or click the link *Blocks programming* at the main page.

The page looks as follows:



Assemble your program using blocks in the menu at the left and then click *Run* button for running. You can also view generated Python-code, switching to *Python* tab.

The *Stop* button stops the program. Clicking *Land* button also stops the program and lands the drone.

Storing and loading



To store the program, open the menu at the top right, select *Save* item and input your program's name. The name should contain only Latin characters, hyphen, underline and dot characters. All your stored programs are available at the same menu.

Your programs are stored as XML-files in the `/catkin_ws/src/clover/clover_blocks/programs/` directory of the SD-card.

> Note also example programs, available at the same menu.

Blocks

The set of blocks is somewhat similar to the set of ROS-services of Clover's autonomous flights API. This section contains descriptions of some of them.

Clover's blocks are separated into 4 categories:

- Flight – autonomous flight related commands.
- State – blocks for obtaining the drone state parameters.
- LED – blocks for controlling LED strip.
- GPIO – blocks for working with GPIO pins.

The rest of categories contains standard Blockly's blocks.

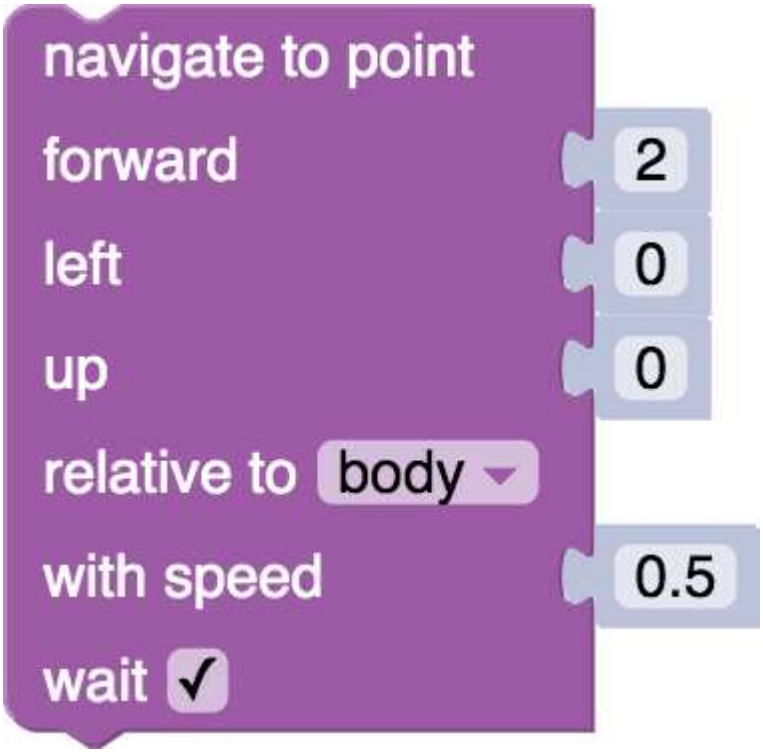
take_off



Take off to specified altitude in meters. The altitude may be an arbitrary block, that returns a number.

The `wait` flag specifies, if the drone should wait until take off is complete, before executing the next block.

navigate



Navigate to specified point. Coordinates are specified in meters.

The `wait` flag specifies, if the drone should wait until navigation is complete, before executing the next block.

Relative to field

This block allows to specify the coordinate frame of the target point:

- *body* – coordinates, relative to the drone: *forward*, *left*, *up*.
- *markers map* – coordinates, relative to the map of ArUco-markers.
- *marker* – coordinates, relative to an ArUco-marker; marker's ID input fields appears.
- *last navigate target* – coordinates, relative to the last specified navigate point.
- *map* – drone's local coordinate system, linked with the point of its initialization.
- *global* – global coordinates system (latitude and longitude) and relative altitude.
- *global, WGS 84 alt.* – global coordinates system and WGS 84 altitude.

land



Land the drone.

The `wait` flag specifies, if the drone should wait until landing is complete, before executing the next block.

wait



Wait specified time period in seconds. The time period may be an arbitrary block, that returns a number.

wait_arrival



Wait, until the drone reaches navigate-block's target point.

get_position



The block returns current position, velocity or yaw angle of the drone relative to the specified coordinate frame.

set_effect



The block allows to set animations to LED strip, similarly to `set_effect` ROS-service.

Example of using the block with a random color (colors-related blocks are located in *Colour* category):



Work with GPIO

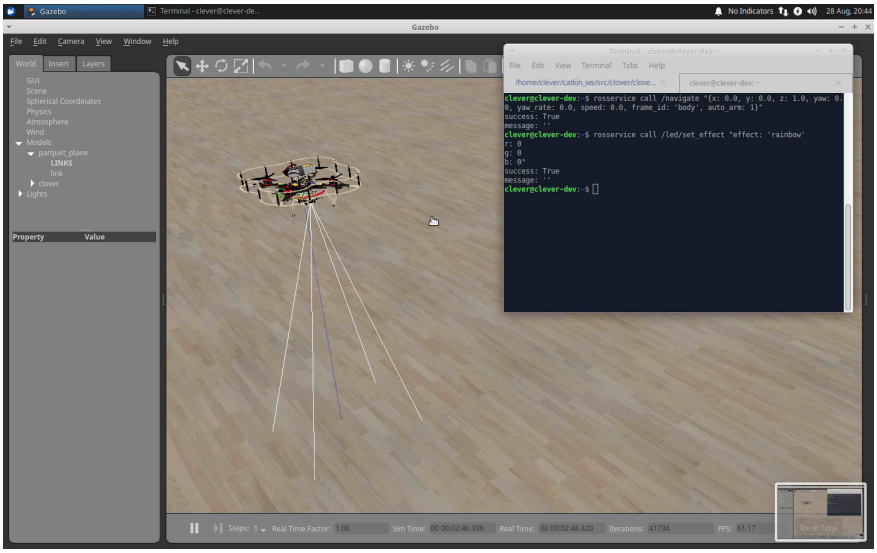
GPIO category contains blocks for working with GPIO. Note, that for correct work of these blocks, `pigpiod` daemon should be running:

```
sudo systemctl enable pigpiod.service
sudo systemctl start pigpiod.service
```

See details on GPIO in the appropriate article.

Simulation overview

The Clover simulation environment allows users to run and debug their code within a simulator while using most of the features available on the real drone. The simulation utilizes PX4 SITL mode and uses the same ROS code as the real drone. Most hardware is simulated as well.



Features

Basic, user-installable environment includes:

- high-quality Clover 4 visual model;
- Gazebo plugins for Clover-specific hardware (e.g. LED strip);
- modification-friendly `xacro` drone descriptions;
- sample models and worlds;
- `roslaunch` files for quick simulation launching and configuration.

Additionally, a virtual machine image that mimics the real drone as closely as possible is provided with the following features:

- easy access to the simulation environment;
- Visual Studio Code editor, preconfigured to work with ROS packages;
- Monkey web server for web-based Clover plugins;
- always-running `roscore` service;
- ROS GUI tools (`rviz` , `rqt`).

Architecture

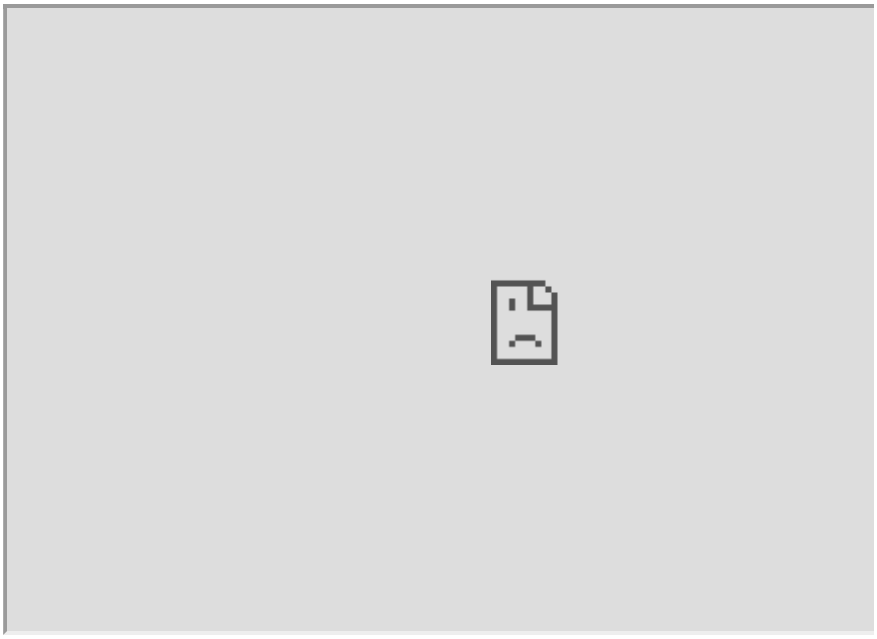
The simulation environment is based on the following components:

- Gazebo, a state-of-the-art robotics simulator;
- PX4, specifically its SITL (software-in-the-loop) components;

- `sitl_gazebo` package containing Gazebo plugins for PX4;
- ROS packages and Gazebo plugins.

Video

Short video review of the simulator:



Native setup

Setting up the simulation environment from scratch requires some effort, but results in the most performant setup, with less chance of driver issues.

Prerequisites: **Ubuntu 20.04**.

Install ROS

Install ROS Noetic using the official installation manual (Desktop or Full install).

Add sourcing ROS' `setup.bash` initialization script to your `.bashrc`:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Install required tools:

```
sudo apt install build-essential git python3-pip python3-rosdep
```

Create a workspace for the simulation

Create a workspace for the simulation:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Clone Clover sources:

```
cd ~/catkin_ws/src
git clone --depth 1 https://github.com/CopterExpress/clover
git clone --depth 1 https://github.com/CopterExpress/ros_led
git clone --depth 1 https://github.com/ethz-asl/mav_comm
```

Install all dependencies using `rosdep`:

```
cd ~/catkin_ws
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src -y
```

Install Python dependencies:

```
sudo /usr/bin/python3 -m pip install -r ~/catkin_ws/src/clover/clover/requireme
```

Get PX4 sources

PX4 will be built along with the other packages in our workspace. You may clone it directly into the workspace or put it somewhere and symlink to `~/catkin_ws/src`. We will need to put its `sitl_gazebo` and `mavlink` submodules into `~/catkin_ws/src` as well.

Clone PX4 sources and make the required symlinks:

```
git clone --recursive --depth 1 --branch v1.12.3 https://github.com/PX4/PX4-Aut
ln -s ~/PX4-Autopilot ~/catkin_ws/src/
ln -s ~/PX4-Autopilot/Tools/sitl_gazebo ~/catkin_ws/src/
ln -s ~/PX4-Autopilot/mavlink ~/catkin_ws/src/
```

> You may use more recent PX4 version, but something may not work as expected in that case.

> If clone fails with network error (`fatal: fetch-pack: invalid index-pack output`), set HTTP version 1.1 using `git config --global http.version HTTP/1.1` command (don't forget to return it back after clone using `git config --global http.version HTTP/2`). Alternative solution is cloning the repository and submodules through SSH using `git config --global url."git@github.com:".insteadOf https://github.com/` command (requires setting up valid SSH key in GitHub profile settings).

Install PX4 prerequisites

PX4 comes with its own script for dependency installation. We may as well leverage it:

```
cd ~/catkin_ws/src/PX4-Autopilot/Tools/setup
sudo ./ubuntu.sh
```

This will install everything required to build PX4 and its SITL environment.

> You may want to skip installing the ARM toolchain if you're not planning on compiling PX4 for your flight controller. To do this, use the `--no-nuttx` flag: `sudo ./ubuntu.sh --no-nuttx`.

Install more required Python packages:

```
pip3 install --user toml
```

Add the Clover airframe

Add the Clover airframe to PX4 using the command:

```
ln -s ~/catkin_ws/src/clover/clover_simulation/airframes/* ~/PX4-Autopilot/ROMF
```

Install geographiclib datasets

`mavros` package requires geographiclib datasets to be present:

```
sudo /opt/ros/noetic/lib/mavros/install_geographiclib_datasets.sh
```

Build the simulator

Build your workspace:

```
cd ~/catkin_ws
catkin_make -j1
```

> The `-j1` flag means that the build will not use parallel processes, as building with parallel processes on a virtual machine may run out of memory. If you have enough memory, you may not use this flag.

Run the simulator

In order to be sure that everything was built correctly, try running the simulator for the first time:

```
roslaunch clover_simulation simulator.launch
```

You can test autonomous flight using example scripts in `~/catkin_ws/src/clover/clover/examples` directory.

Additional steps

To make it possible to run Gazebo simulation environment without Clover (`gazebo` command), add into your `.bashrc` sourcing Gazebo's initialization script:

```
echo "source /usr/share/gazebo/setup.sh" >> ~/.bashrc
```

Optionally, install roscore systemd service to have roscore running in background:

```
sed -i "s/pi/$USER/g" ~/catkin_ws/src/clover/builder/assets/roscore.service
sudo cp ~/catkin_ws/src/clover/builder/assets/roscore.service /etc/systemd/syst
sudo systemctl enable roscore
sudo systemctl start roscore
```

Web tools setup

Install any web server to serve Clover's web tools (`~/.ros/www` directory), e. g. Monkey:

```
wget https://github.com/CopterExpress/clover_vm/raw/master/assets/packages/monk
sudo dpkg -i /tmp/monkey_*.deb
sed "s/pi/$USER/g" ~/catkin_ws/src/clover/builder/assets/monkey | sudo tee /etc
sudo sed -i 's/SymLink Off/SymLink On/' /etc/monkey/monkey.conf
sudo cp ~/catkin_ws/src/clover/builder/assets/monkey.service /etc/systemd/syste
sudo systemctl enable monkey
sudo systemctl start monkey
```

Create `~/.ros/www` using the following command:

```
rosrun clover www
```

If the set of packages containing a web part (through `www` directory) is changed, the above command also must be run.

Simulation VM setup

In addition to native installation instructions, we provide a preconfigured developer virtual machine image. The image contains:

- Ubuntu 20.04 with XFCE lightweight desktop environment;
- ROS packages required to develop for the Clover platform;
- QGroundControl;
- preconfigured Gazebo simulation environment;
- Visual Studio Code with C++ and Python plugins.

> The default username on the VM is `clover`, with password `clover`.

The VM is an easy way to set up a simulation environment, but can be used as a development environment for a real drone as well.

Downloading

You can download the latest VM image in the VM releases repository.

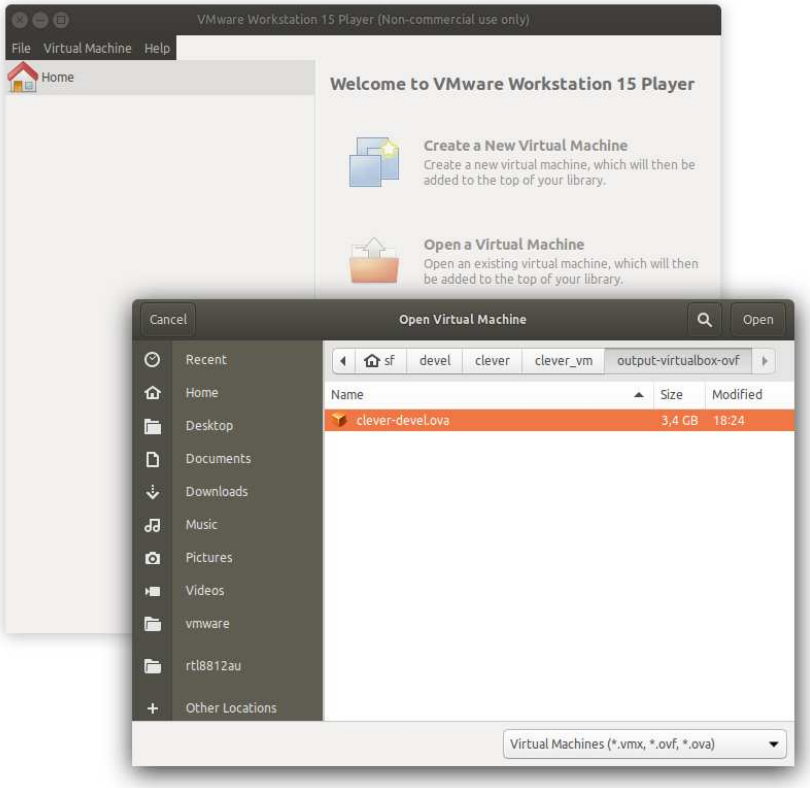
Setting up the VM

You need to use a VM manager that supports OVF format, like VirtualBox, VMware Player or VMware Workstation.

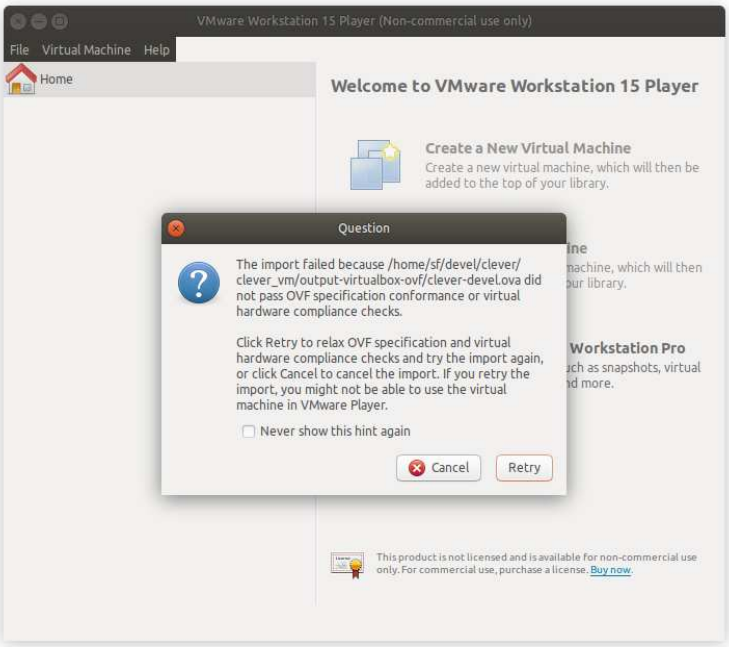
> At the time of writing VirtualBox had issues running the VM, particularly with 3D applications. We recommend using VMware Player or VMware Workstation if possible. The following steps assume you're using VMware Player.

Make sure that you have hardware virtualization enabled in your BIOS/UEFI (it may be supported by your hardware but turned off by default). The steps to enable virtualization differ from manufacturer to manufacturer, but should be described in your system manual. Consult your system's manufacturer if you're having trouble turning virtualization on.

1. Import the OVA archive into your virtualization environment. Use the **Open a Virtual Machine** option in VMware Player:



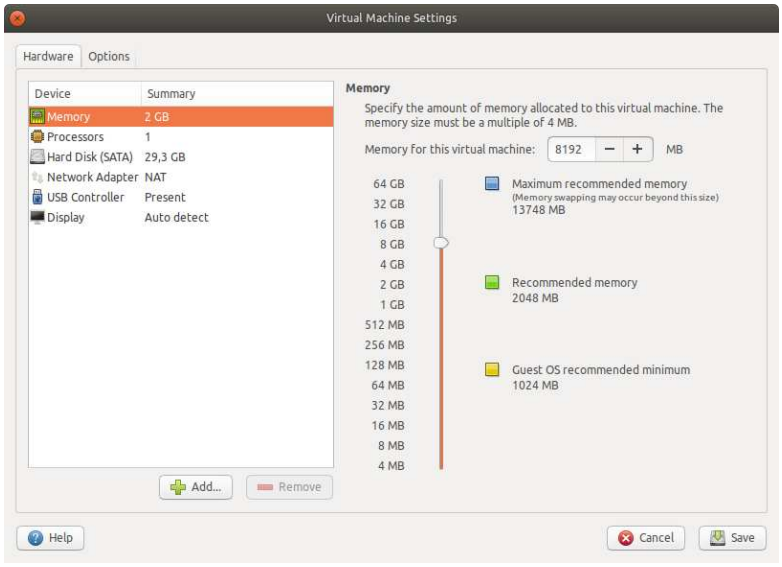
You may see a dialog box with a warning about the VM format:



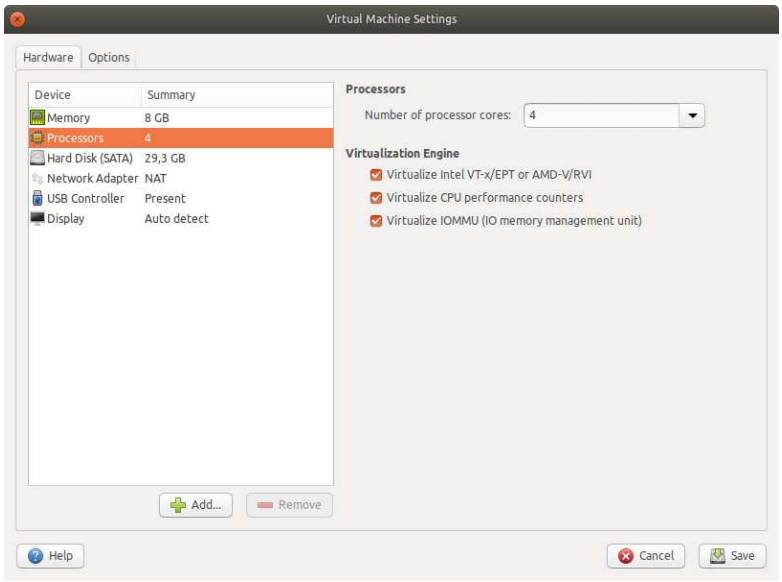
You can safely ignore the warning and press **Retry**.

2. Right-click on the VM name and select **Virtual Machine Settings**. In the new window, set the following parameters:

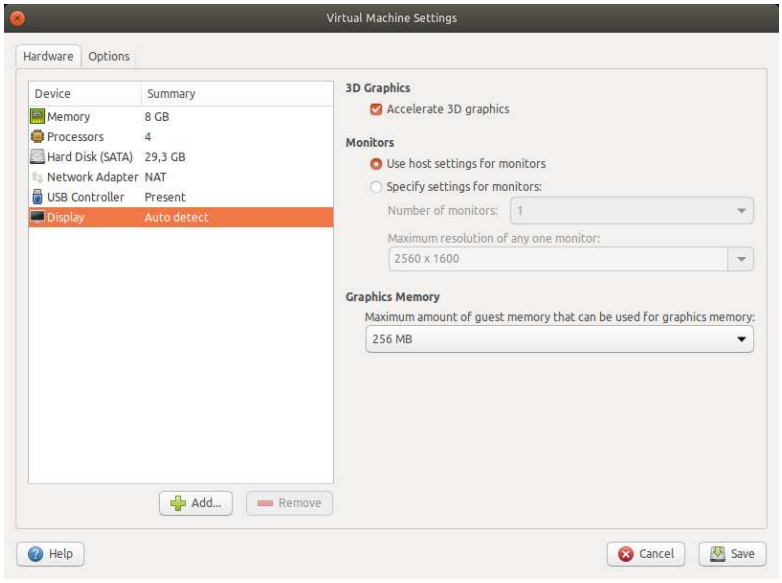
- increase the amount of memory available to the virtual machine (a good rule of thumb is 2048 MB per CPU core, but no less than 4 GB):



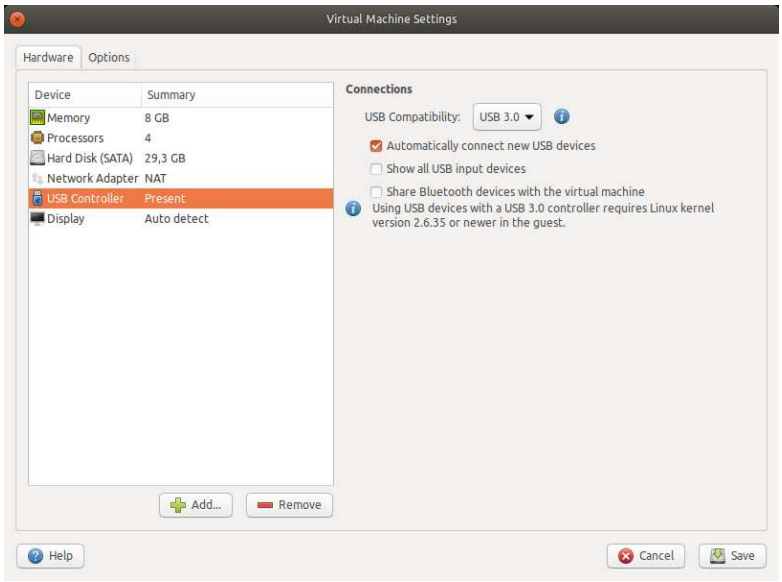
- increase the amount available CPU cores:



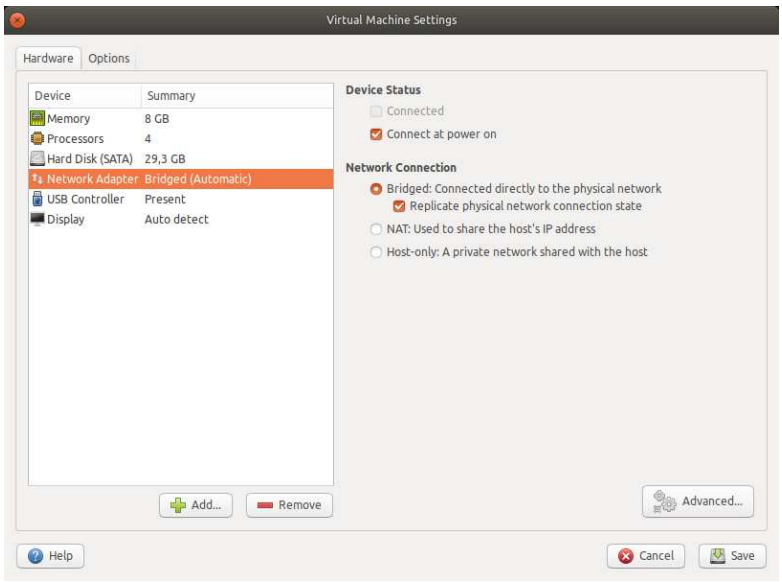
- enable 3D acceleration:



- enable USB 2.0/3.0 controller (if you plan to connect external devices to the VM):

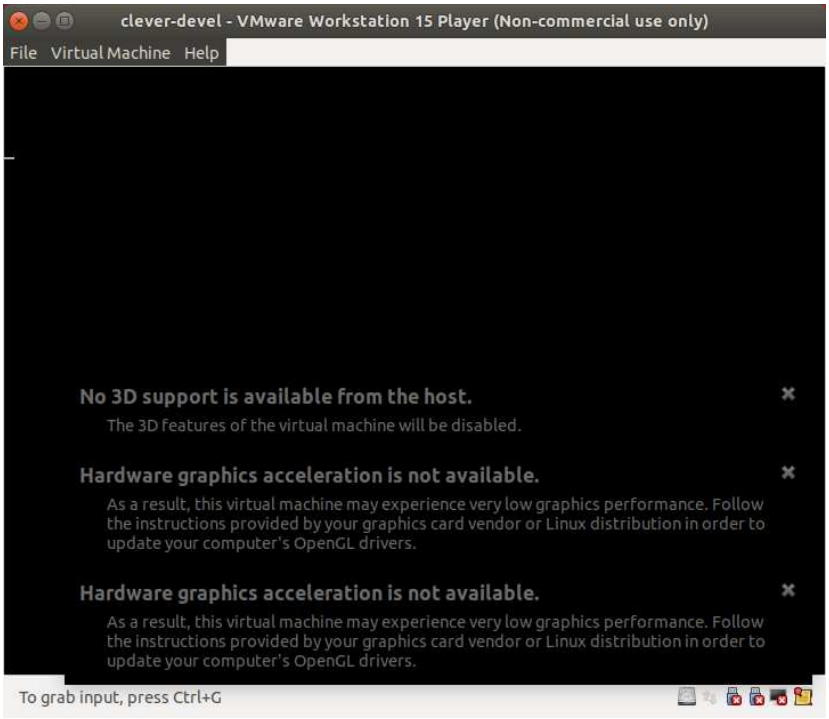


- optionally enable the "bridged" network connection (if you plan to connect to a real drone):



> Some host network adapters may not work well with the bridged network. Consider using external USB Wi-Fi adapters managed by your VM to connect to a real drone.

3. "Power on" the virtual machine. You may see a warning message about your host system not providing 3D acceleration:

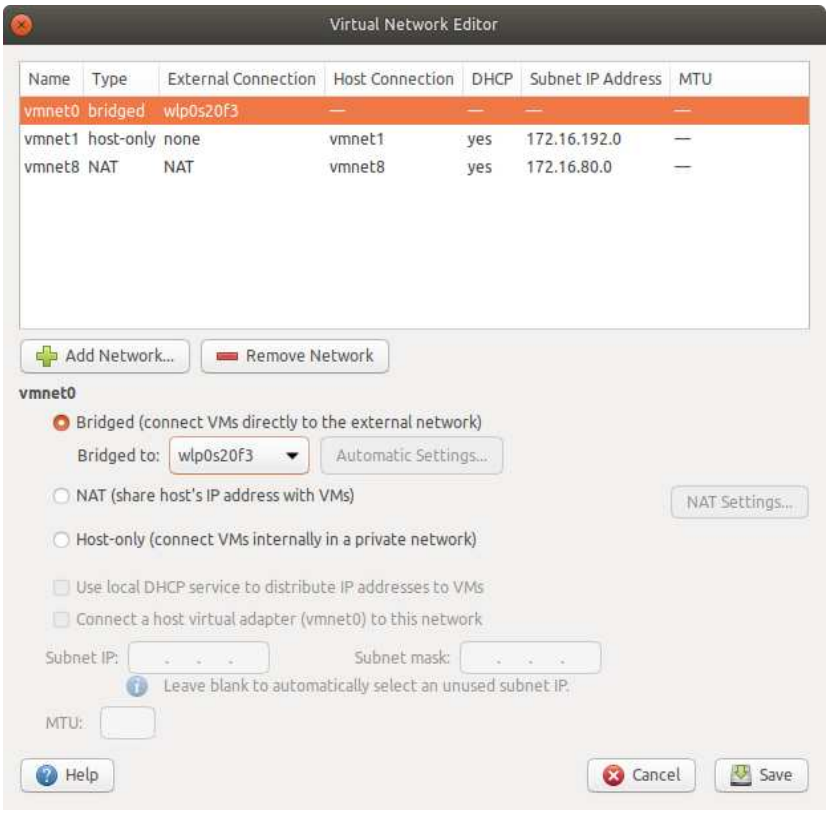


Make sure you have the latest GPU drivers for your host system. If the warnings persist, add the following line to `clover-devel.vmx` (actual file name may differ based on the VM name):

```
mks.gl.allowBlacklistedDrivers = "TRUE"
```

You can find this file in a folder where the VM is imported to.

4. (Bridged networking only) Set up network bridge configuration in VM settings or using `vmware-netcfg` utility (in Linux):



Select `vmnet0` in the networks list, set it to *Bridged*, and choose the adapter you are planning to use to connect to drone in the drop-down menu.

Using the simulator

The Clover simulation environment allows the user to test their code without any risk of equipment damage. Additionally, the virtual machine-based environment has additional (non-ROS) services that are present on a real drone, like Monkey web server.

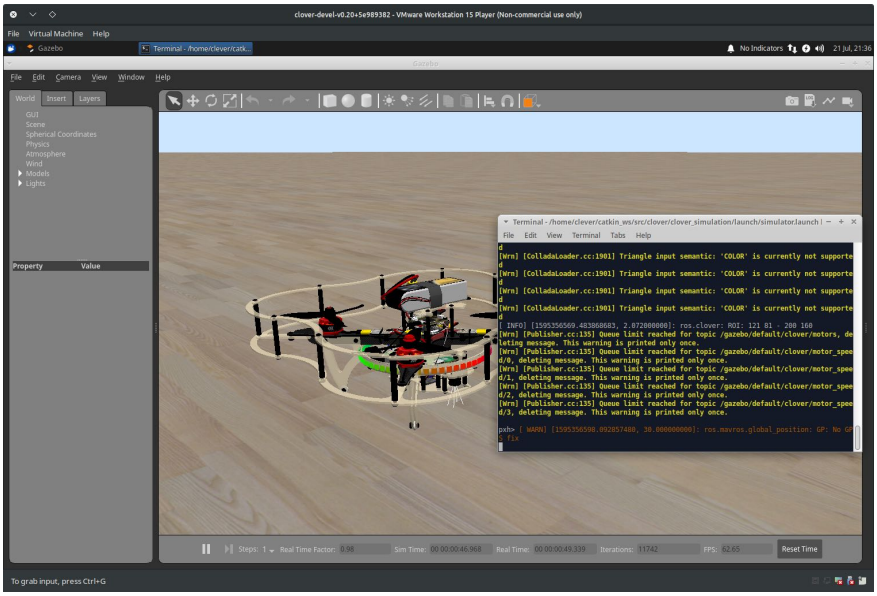
Running the simulation

After setting up the simulation packages or importing and running the VM, you can use `roslaunch` to start Gazebo simulation:

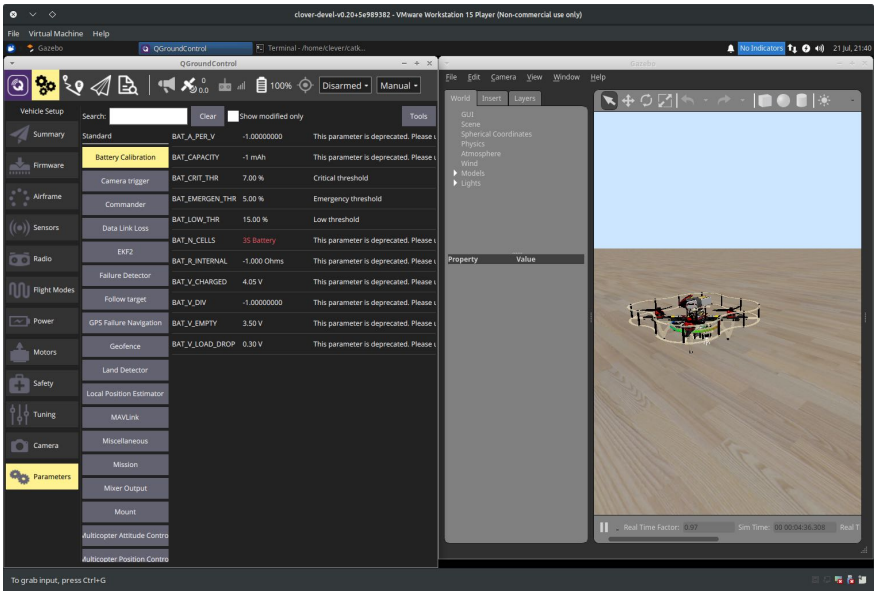
```
# Be sure to activate your workspace first
source ~/catkin_ws/devel/setup.bash
roslaunch clover_simulation simulator.launch
```

> Alternatively, if you are using the VM, just double-click on the `Gazebo PX4` icon on the desktop.

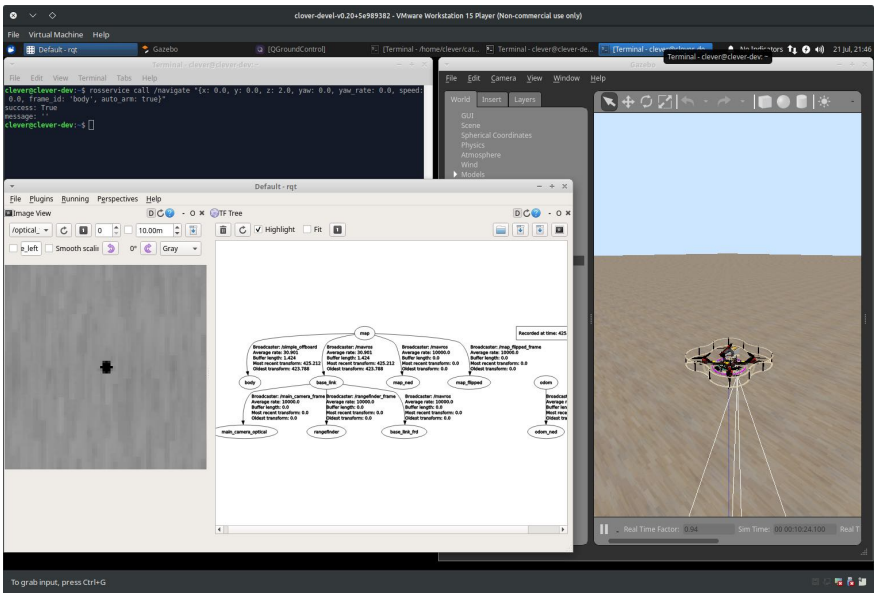
This will launch Gazebo server and client, the PX4 SITL binary and Clover nodes. The terminal in which the command was run will display diagnostic messages from the nodes and PX4, and will accept input for the PX4 command interpreter:



You can use QGroundControl to configure the simulated drone parameters, plan missions (if GPS is simulated) and control the drone using a joystick:

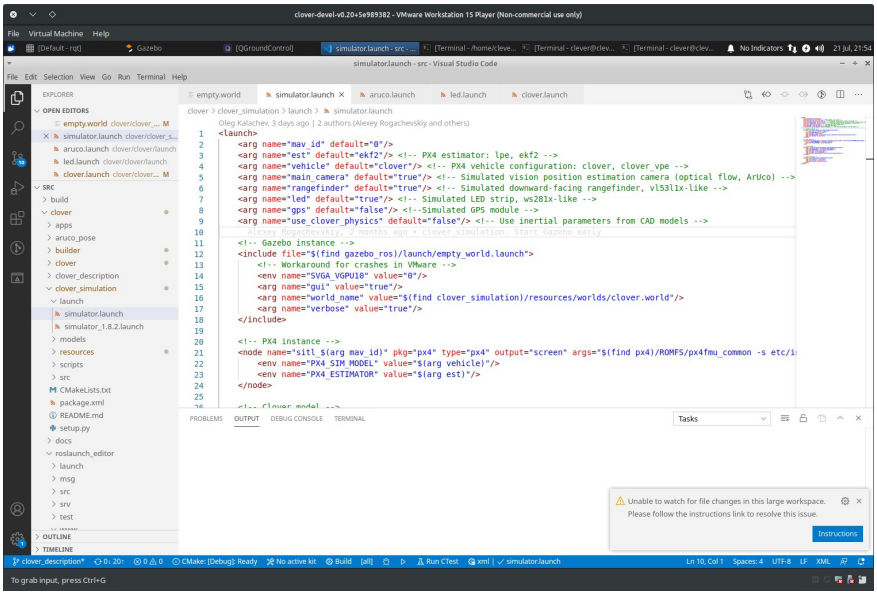


You can also use our simplified OFFBOARD control to control the drone, and traditional ROS GUI utilities like rviz and rqt to analyze the drone state:



Configuring the simulation

The simulation can be configured by passing additional arguments to the `roslaunch` command or by changing the `~/catkin_ws/src/clover/clover_simulation/launch/simulator.launch` file. Nodes that provide ArUco detection, optical flow calculation and other services can be configured by changing their respective `.launch` files, just like on a real drone.



Enabling GPS

In order to enable GPS sensor, set the `gps` argument in `simulator.launch` to `true`:

```
<arg name="gps" value="true"/>
```

Turn also on the *use GPS* flag in the `EKF2_AID_MASK` PX4 parameter (using QGroundControl).

Camera

If you don't need the camera when flying using GPS, it may be disabled in `simulator.launch` file:

```
<arg name="main_camera" default="false"/>
```

Another sensors

If you wish to add additional sensors or change their placement, you will have to change the drone description. The description file is located in `~/catkin_ws/src/clover/clover_description/urdf/clover/clover4.xacro`, and uses the xacro format to build URDF description.

Changing the default world

Gazebo plugins for the drone currently require the `real_time_update_rate` world parameter to be 250, and `max_step_size` to be 0.004. Using other values will not work. Consider using `~/catkin_ws/src/clover/clover_simulation/resources/worlds/clover.world` as a base.

Performance suggestions

Gazebo simulation environment is resource-intensive, and requires a fast CPU and a decent GPU for real-time execution. However, the simulation may still work on less powerful systems at slower-than-realtime rate. Below are some suggestions for running Gazebo on hardware that does not allow realtime execution.

Use `throttling_camera` plugin

By default, Gazebo does not slow simulation down for visual sensors. This can be remedied by using the `throttling_camera` plugin from `clover_simulation`.

You can enable it for the drone by changing the `maintain_camera_rate` argument to `true` in `clover_description/launch/spawn_drone.launch`:

```
    <!-- Slow simulation down to maintain camera rate -->
    <arg name="maintain_camera_rate" default="true"/>
```

The plugin will collect publishing rate statistics and slow the simulation down so that the camera publishing rate is no less than requested. However, large slowdowns may negatively affect non-ROS software that connects to PX4. If your simulation is being slowed down to values lower than 0.5 of realtime, consider using the next suggestion.

Set simulation speed

Since v1.9 the PX4 SITL setup supports setting the simulation speed by setting the `PX4_SIM_SPEED_FACTOR` environment variable. Its value is picked up by PX4 startup scripts, which in turn reconfigure it to expect a certain speedup/slowdown.

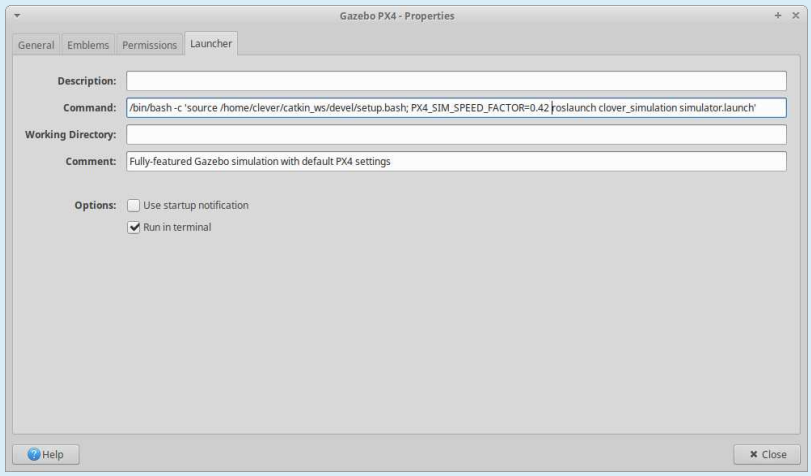
You should set its value to the actual real time factor that you get with `throttling_camera`. The real time factor may be found in the Gazebo GUI window at the bottom:



In this example you should set `PX4_SIM_SPEED_FACTOR` to `0.42` when launching the simulation:

```
PX4_SIM_SPEED_FACTOR=0.42 roslaunch clover_simulation simulator.launch
```

If you are using the VM, it may be convenient to put the value in the Gazebo desktop shortcut. Right-click on the Gazebo icon, select "Properties..." and add `PX4_SIM_SPEED_FACTOR=0.42` to the Command field as follows:



Allocate more resources to the VM

The virtual machine may benefit from several CPU cores, especially if the cores are not very performant. In our tests, a four-core machine with only a single core allocated to the VM was unable to run the simulation, with constant interface freezes and dropped ROS messages. The same machine with all four cores available to the VM was able to run the simulation at 0.25 real-time speed.

Do note that you should not allocate more resources than you have on your host hardware.

Changing the map of ArUco-markers in the simulator

In order to change the map of ArUco-markers in the simulator, you can use the following command:

```
rosrun clover_simulation aruco_gen --single-model --source-world=$(catkin_find
```

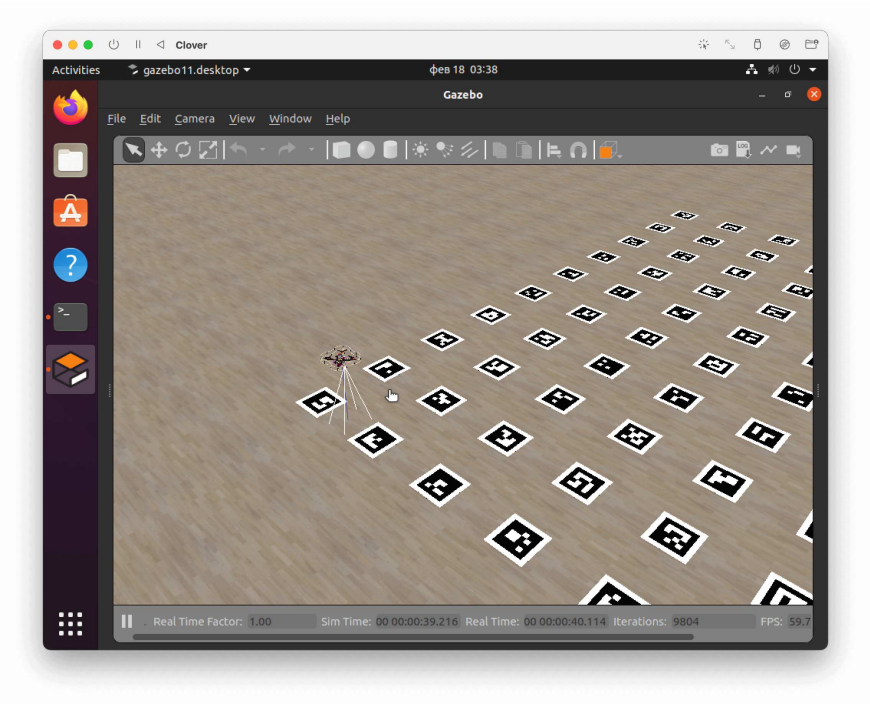
In this example, `map.txt` is the name of markers name.

Running simulator on M1 powered computer

There is no preconfigured VM image for ARM64 architecture of M1 chip (Apple Silicon), so the only possibility is to install the simulation software manually.

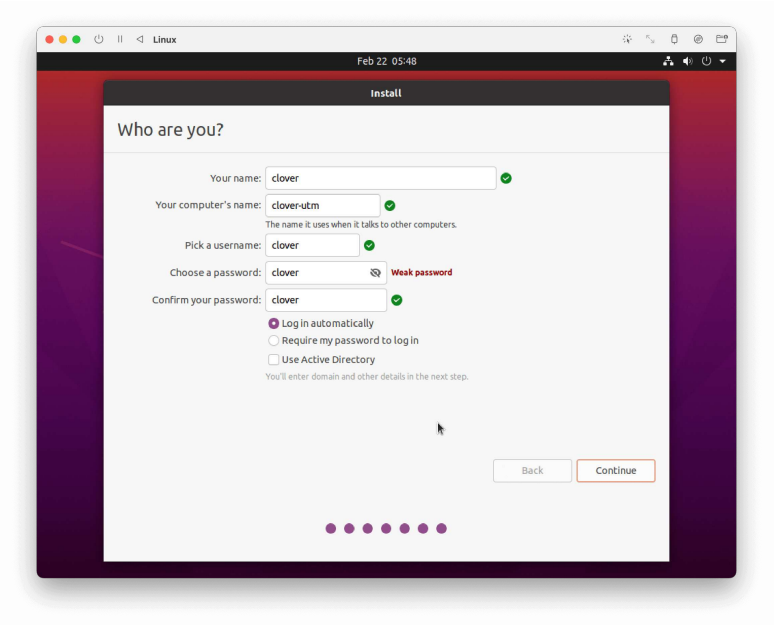
The recommended virtual machine hypervisor is UTM app. Also it's possible to use **VMware Fusion Public Tech Preview** with M1 support.

Simulation installation with UTM



1. Download UTM App from the official site mac.getutm.app and install it.
2. Download Ubuntu Linux 20.04 installation iso-file for ARM64 architecture using the link: https://clovervm.ams3.digitaloceanspaces.com/focal-desktop-arm64.iso.
3. Create a new virtual machine in UTM, using the following settings:
   - **Type**: Virtualize.
   - **Operating System**: Linux.
   - **Boot ISO Image**: choose downloaded file `focal-desktop-arm64.iso`.
   - **Memory**: 4096 MB or more.
   - **CPU Cores**: 4 or more.
   - Turn on *Enable hardware OpenGL acceleration* option.
   - **Storage**: 20 GB or more.
4. Run the created virtual machine.
5. Choose *Install Ubuntu* in the menu and install it using the installation master.

- Recommended apps: *Minimal installation*.
- Installation type: *Erase disk and install Ubuntu*.
- Input your account parameters, for example:



6. Finish the installation and run the system (you need to eject the virtual CD or choose *Boot from next volume* in the boot menu).
7. Install the simulation using the native setup manual.

Troubleshooting

Black screen

If you see a black screen on your virtual machine, try to run the machine without the GPU support.

In virtual machine settings, choose *Display*, and set *Emulated Display Card* menu to *virtio-ramfb*. Run you machine. If it runs successfully, change the setting back to *virtio-ramfb-gl (GPU Supported)* and run it again.

Problem with `git clone`

The following error can occur while performing `git clone`:

```
on git clone if error: RPC failed; curl 56 GnuTLS recv error (-54): Error in th
fatal: the remote end hung up unexpectedly
fatal: early EOF
fatal: index-pack failed
```

In this case, change the type of the network card to bridged. In the virtual machine settings, choose *Network*, and set *Network Mode* menu to *Bridged (Advanced)*.

Later, if some network issues occur, change the network mode back to *Shared Network*.

ROS

Main documentation: https://wiki.ros.org.



ROS is a widely used framework for developing complex and distributed robotic systems. The Clover autonomous flights platform is based on ROS.

Installation

ROS is already installed on the RPi image.

To install ROS on your PC you may address the official installation documentation. For a quick start it's recommended to use the virtual machine image with ROS and Clover simulator.

Concepts

Nodes

Main article: https://wiki.ros.org/Nodes.

ROS node is a special program (usually written in Python or C++) that communicates with other nodes via ROS topics and ROS services. Dividing complex robotic systems into isolated nodes provides certain advantages: reduced coupling of the code, increased reusability and reliability.

Many robotic libraries and drivers are made as ROS nodes.

In order to turn an ordinary program into a ROS node, include the `rospy` (Python) or `roscpp` (C++) library, and insert the initialization code.

An example of a ROS node in Python:

```
import rospy

rospy.init_node('my_ros_node')  # the name of the ROS node

rospy.spin() # entering an infinite loop...
```

> Any autonomous flight script for Clover is a ROS node.

Topics

Main article: https://wiki.ros.org/Topics

A topic is a named data bus used by the nodes for exchanging messages. Any node can *publish* a message to any topic, and *subscribe* to any topic.

Для каждого созданного топика должен быть задан тип сообщений, которые по нему передаются. ROS включает в себя большое количество стандартных типов сообщений, покрывающих различные аспекты робототехники, но при необходимости возможно создание собственных типов сообщений. Примеры стандартных типов сообщений:

Each topic has the a of messages it passes. ROS include a lot of standard message types, covering different aspects of robotics. Creating custom message types is also possible. Example of standard message types:

Message type	Description
`std_msgs/Int64`	Integer number.
`std_msgs/Float64`	Double-precision floating-point number.
`std_msgs/String`	String.
`geometry_msgs/PoseStamped`	Position and orientation of an object in a given coordinate system and a time stamp (widely used for passing the robot pose or some robot's part pose).
`geometry_msgs/TwistStamped`	Linear and angular velocity of an object in a given coordinate system and a time stamp.
`sensor_msgs/Image`	Image (see the article on working with the camera).

> See the rest of standard message types in packages: `common_msgs`, `std_msgs`, `geometry_msgs`, `sensor_msgs`, and others.

Example of publishing a message of type `String`) in a topic `/foo` in Python:

```
from std_msgs.msg import String

rospy.init_node('my_ros_node')

foo_pub = rospy.Publisher('/foo', String, queue_size=1)  # creating a Publisher

foo_pub.publish(data='Hello, world!')  # publishing the message
```

Example of subscription to a topic `/foo`:

```
import rospy
from std_msgs.msg import String

rospy.init_node('my_ros_node')

def foo_callback(msg):
    print(msg.data)

# Subscribing. When a message is received in topic /foo, function foo_callback
rospy.Subscriber('/foo', String, foo_callback)
```

You can read a topic message once, using `wait_for_message` function:

```
msg = rospy.wait_for_message('/foo', String, timeout=3)  # wait for a message
```

You can also work with topics using the `rostopic` utility. For example, using the following command, you can view messages published in topic `/mavros/state`:

```
rostopic echo /mavros/state
```

The `rostopic info` command shows the type of messages in the topic, and `rostopic hz` shows frequency of published messages.

Also you can monitor and visualize topics using graphical tools of ROS.

Services

Main article: https://wiki.ros.org/Services

A service can be assimilated to the a function that can be called from one node, and processed in another one. The service has a name that is similar to the name of the topic, and 2 message types: request type and response type.

Thus, ROS services implement remote procedure call (RPC) pattern.

Example of invoking a ROS service in Python:

```
import rospy
from clover.srv import GetTelemetry

rospy.init_node('my_ros_node')

# Creating a wrapper for the get_telemetry service of the clover package with
get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)

# Invoking the service, and getting the quadcopter telemetry:
telemetry = get_telemetry()
```

You can also work with the services using the `rosservice` utility. For instance, you can call service `/get_telemetry` from the command line:

```
rosservice call /get_telemetry "{frame_id: ''}"
```

More examples of using the services for Clover quadcopter autonomous flights are available in the documentation for node simple_offboard.

Names

Main article: https://wiki.ros.org/Names.

Any topic, service or a parameter is identified with a unique name. A ROS name is hierarchical structure with a `/` symbol as a separator (which is close to a file name in a file system).

Examples of ROS names:

- `/` (global namespace)
- `/foo`
- `/stanford/robot/name`
- `/wg/node1`

This names are global (close to global names in a file system). In practice, it's recommended to use *private* or *relative* names.

Private name

Each node can use its own private namespace (corresponding its name) for its resources. For example, `aruco_detect` node may publish such topics:

- `/aruco_detect/markers`
- `/aruco_detect/visualization`
- `/aruco_detect/debug`

When a node is referring its private resource, instead of `/aruco_detect/` namespace it may use `~` symbol:

- `~markers`
- `~visualization`
- `~debug`

Thus, creating a `foo` topic and the private namespace would look like this:

```
private_foo_pub = rospy.Publisher('~foo', String, queue_size=1)
```

Relative name

Several nodes may group into a common namespace (for example, when there are several robots in the network). For referring topics and services in the current namespace, the opening `/` symbol is omitted.

Example of create a `foo` topic in the current namespace:

```
relative_foo_pub = rospy.Publisher('foo', String, queue_size=1)
```

> Generally, it's recommended to use private or relative names instead of global ones.

Working on several PCs

Main article: http://wiki.ros.org/ROS/Tutorials/MultipleMachines.

The advantage of using ROS is the possibility of distributing the nodes across several PCs in the network. For example, a node that recognizes an image may be run on a more powerful PC; the node that controls the copter may be run directly on a Raspberry Pi connected to the flight controller, etc.

MAVROS

Main article is available in the official documentations: http://wiki.ros.org/mavros

MAVROS (MAVLink + ROS) is a ROS package that allows controlling drones via the MAVLink protocol. MAVROS supports PX4 and APM flight stacks. Communication may be established via UART, USB, TCP or UDP.

MAVROS subscribes to certain ROS topics that can be used to send commands, publishes telemetry to other topics, and provides services.

The MAVROS node is automatically started in the Clover launch-file. In order to set the type of connection, change the `fcu_conn` argument.

> Simplified interaction with the drone is possible with the use of [`simple_offboard`] package (simple_offboard.md).

> Some MAVROS plugins are disabled by default in the `clover` package in order to save resources. For more information, see the `plugin_blacklist` parameter in `/home/pi/catkin_ws/src/clover/clover/launch/mavros.launch`.

Main services

`/mavros/set_mode` — set flight mode of the controller. Most often used to set the OFFBOARD mode to accept commands from Raspberry Pi.

`/mavros/cmd/arming` — arm or disarm drone motors (change arming status).

Main published topics

`/mavros/state` — status of connection to the flight controller and flight controller mode.

`/mavros/local_position/pose` — local position and orientation of the copter in the ENU coordinate system.

`/mavros/local_position/velocity` — current speed in local coordinates and angular velocities.

`/mavros/global_position/global` — current global position (latitude, longitude, altitude).

`/mavros/global_position/local` — the global position in the UTM coordinate system.

`/mavros/global_position/rel_alt` — relative altitude (relative to the arming altitude).

Messages published in the topics may be viewed with the `rostopic` utility, e.g., `rostopic echo /mavros/state`. See more in working with ROS.

Main topics for publication

`/mavros/setpoint_position/local` — set target position and yaw of the drone (in the ENU coordinate system).

`/mavros/setpoint_position/global` – set target position in global coordinates (latitude, longitude, altitude) and yaw of the drone.

`/mavros/setpoint_position/cmd_vel` — set target linear velocity of the drone.

`/mavros/setpoint_attitude/attitude` and `/mavros/setpoint_attitude/att_throttle` — set target attitude and throttle level.

`/mavros/setpoint_attitude/cmd_vel` and `/mavros/setpoint_attitude/att_throttle` — set target angular velocity and throttle level.

Topics for sending raw packets

`/mavros/setpoint_raw/local` — sends SET_POSITION_TARGET_LOCAL_NED message. Allows setting target position/target speed and target yaw/angular yaw velocity. The values to be set are selected using the `type_mask` field.

`/mavros/setpoint_raw/attitude` — sends SET_ATTITUDE_TARGET message. Allows setting the target attitude /angular velocity and throttle level. The values to be set are selected using the `type_mask` field

`/mavros/setpoint_raw/global` — sends SET_POSITION_TARGET_GLOBAL_INT. Allows setting the target attitude in global coordinates (latitude, longitude, altitude) and flight speed.

Supplementary materials

This section contains articles that are not included in the main sections, as well as articles by the users on various subjects related to UAV.

To learn more about publishing text in this section, see the article "Contributing to Clover".

COEX Pix

The **COEX Pix** flight controller is a modified Pixracer FCU. It is a part of the **Clover 4** quadrotor kit.

> The source files of the COEX Pix flight controller are published under the CC BY-NC-SA license.

Revision 1.1

Physical specs

- Board size: 35x35 mm.
- Mounting hole pattern: standard 30.5 mm.
- Mounting hole diameter: 3.2 mm.
- Board mass: 9 g.
- Operating temperature range: -5..+65 ºC.
- Input voltage: 4.8..5.5 V.

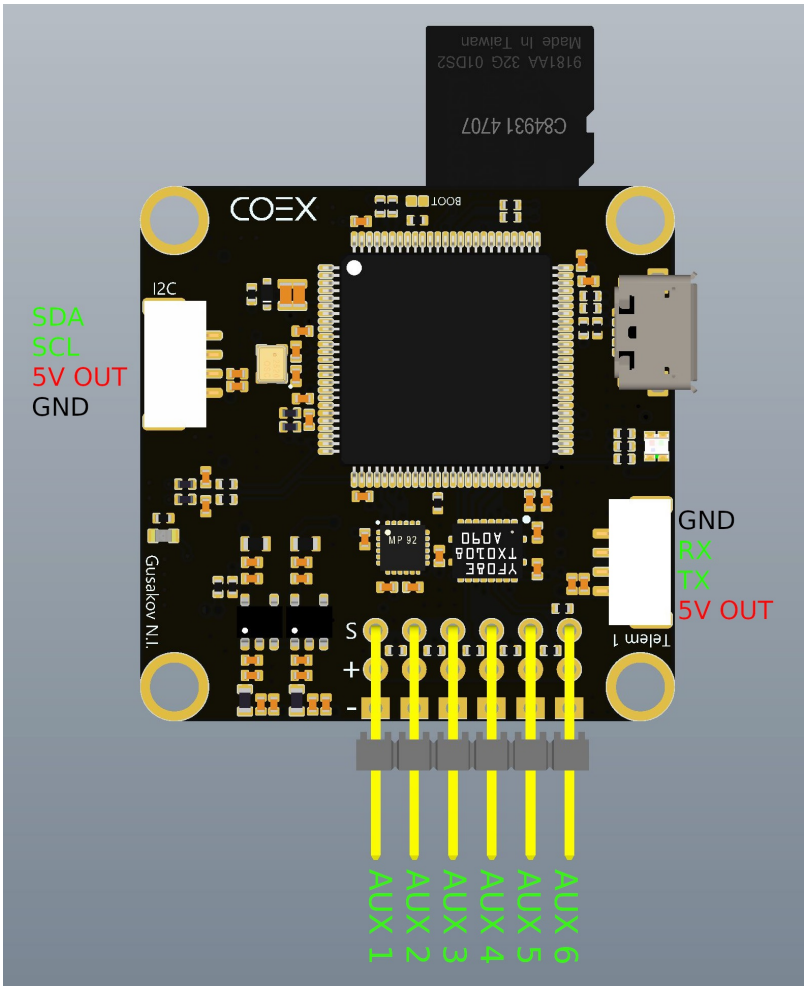
Key features

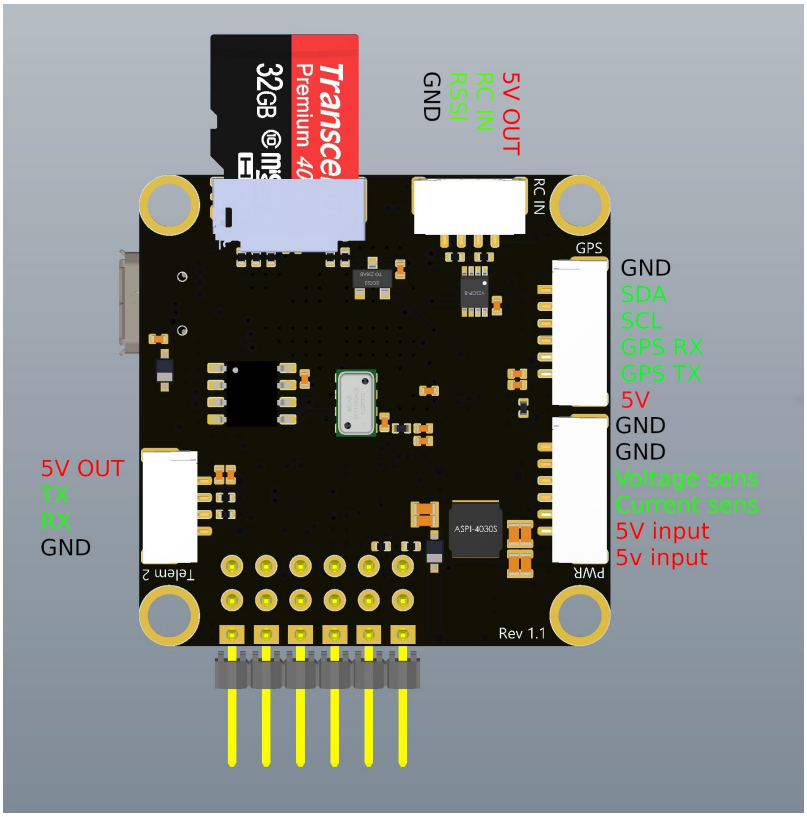
- Main System-on-Chip: *STM32F427VIT6*.
- FRAM chip: *FM25V02A*
- Built-in sensors:
  - *MPU9250* 9DOF accelerometer/gyroscope/magnetometer.
  - *MS5607* barometer.

Ports

- *TELEM 1* (JST-GH 4 pin) – telemetry port 1, UART.
- *TELEM 2* (JST-GH 4 pin) – telemetry port 2, UART.
- *GPS* (JST-GH 6 pin) – GNSS (UART) and external compass (I2C) port.
- *I2C* (JST-GH 4 pin) – I2C port for supported devices (shares lanes with *GPS* port).
- *PWR* (JST-GH 6 pin) – port for PDB connection (COEX PDB or compatible), with two power lanes, two ground lanes, and voltage and current sensor inputs.
- *RC IN* (JST-GH 4 pin) – RC input port with RSSI pin. Supports PPM and S.BUS protocols.
- Micro USB port for PC connection (USB 2.0/1.1).
- MicroSD slot (supports up to 32gb microSD cards).
- 6 servo outputs for ESCs and other peripherals.

Port pinouts


SDA
SCL
5V OUT
GND
I2C
GND
RX
TX
5V OUT
Telem 1
S
+
-
AUX 1
AUX 2
AUX 3
AUX 4
AUX 5
AUX 6
Gusakov N.I.


5V OUT
RC IN
RSSI
GND
RC IN
GPS
GND
SDA
SCL
GPS RX
GPS TX
5V
GND
GND
Voltage sens
Current sens
5V input
5v input
5V OUT
TX
RX
GND
Telem 2
PWR
Rev 1.1

> On rev. 1.0 boards *RC IN* port and microSD slot are switched. Pinout for the *RC IN* port is the same on these boards.

Mounting suggestions

Important: The board is meant to be installed with a non-standard orientation (roll 180º, yaw 90º) on the Clover airframe. Therefore, the `SENS_BOARD_ROT` PX4 parameter should be set to `ROLL 180, YAW 90`.

Usage notes

In order to reduce magnetic interference from the PDB and power cables you should mount the FCU as far away from these parts as possible. You should have at least 15 mm clearance from high-power parts.

You may want to disable internal compass if you're using an external GNSS+compass module.

If your drone does not have a protective cover for the FCU, you should place a piece of foam over the barometer.

The FCU has power passthrough from the *PWR* input to the servo rail. Supplying additional power to the servo rail is not recommended if the *PWR* input is used. Powering the FCU from USB and *PWR*/*AUX* inputs is acceptable.

Board specifics

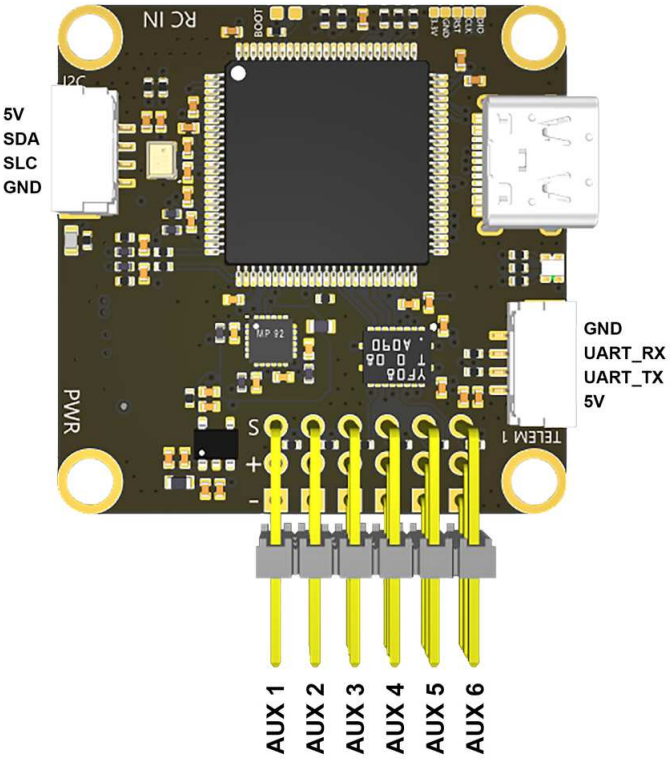
The board utilizes low-noise DC-DC converters, voltage inputs have LC and ferrite filters.

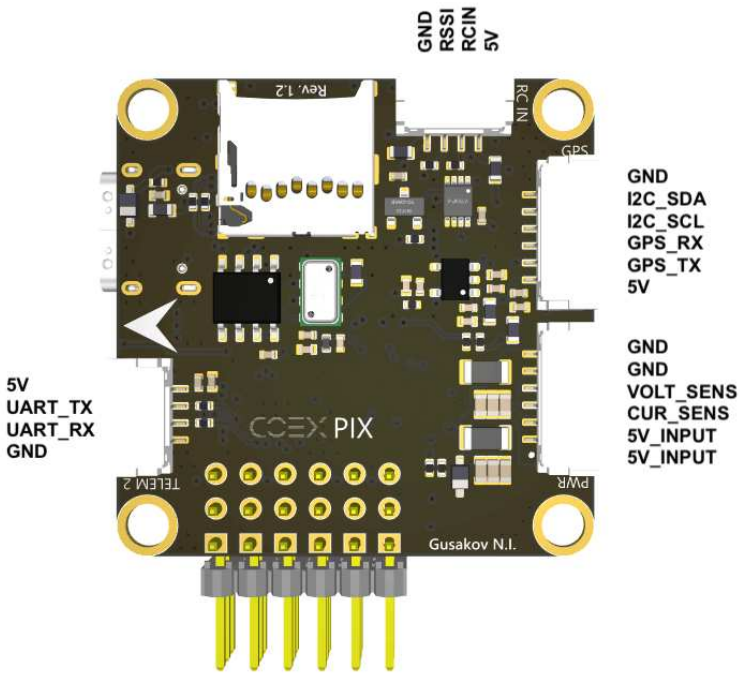
Revision 1.2

Innovations

- Replaced USB Micro-B connector with USB Type-C connector.
- Changed the slot for microSD cards to a deeper one.
- Changed pin assignments on the I2C connector.
- Added ferrite filters in the power circuit.

Port pinouts


RC IN
I2C
5V
SDA
SCL
GND
PWR
TELEM 1
GND
UART_RX
UART_TX
5V
AUX 1
AUX 2
AUX 3
AUX 4
AUX 5
AUX 6


GND
RSSI
RCIN
5V
Rev. 1.2
RCIN
GPS
GND
I2C_SDA
I2C_SCL
GPS_RX
GPS_TX
5V
5V
UART_TX
UART_RX
GND
COEX PIX
TELEM 2
PWR
GND
GND
VOLT_SENS
CUR_SENS
5V_INPUT
5V_INPUT
Gusakov N.I.

COEX PDB

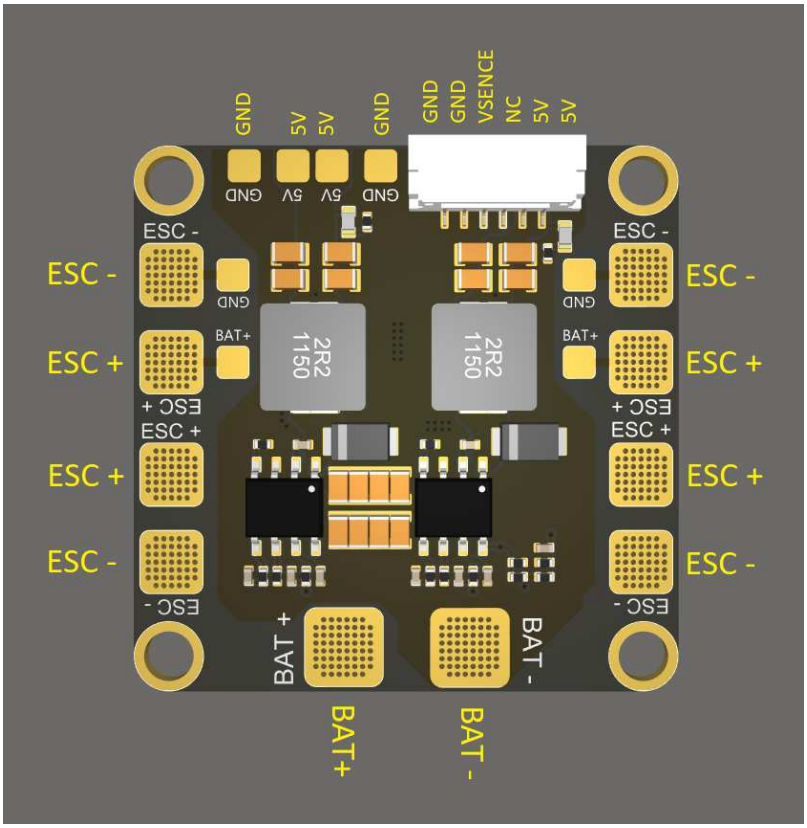
COEX PDB is the power distribution board used in Clover 4 Drone kit.

Board size: 35x35 mm.

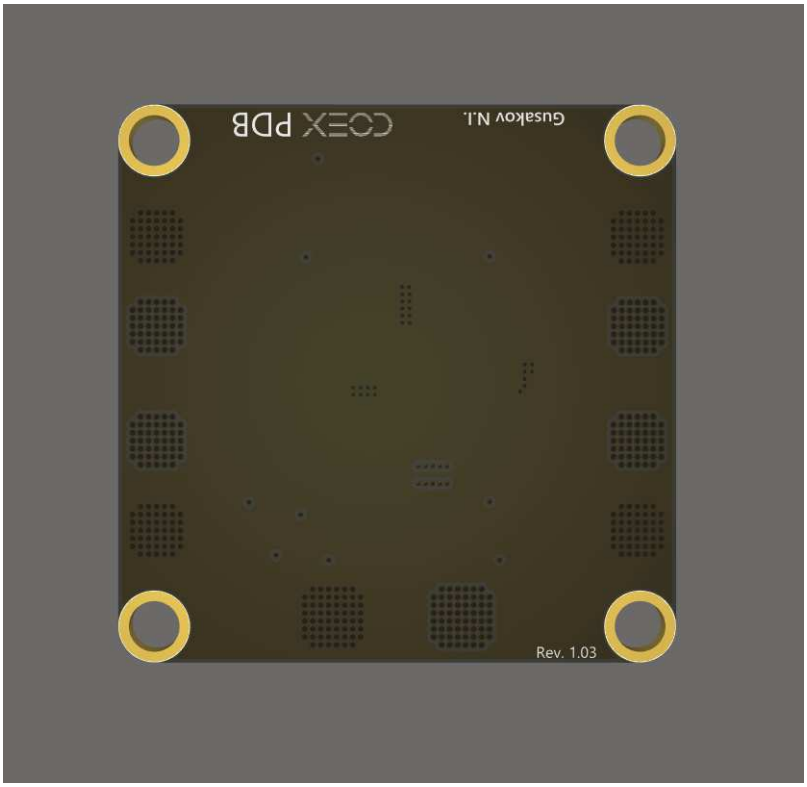
> The source files of the COEX PDB board are published under the CC BY-NC-SA license.

Port pinouts

Top view



Bottom view


COEX PDB
Gusakov N.I.
Rev. 1.03

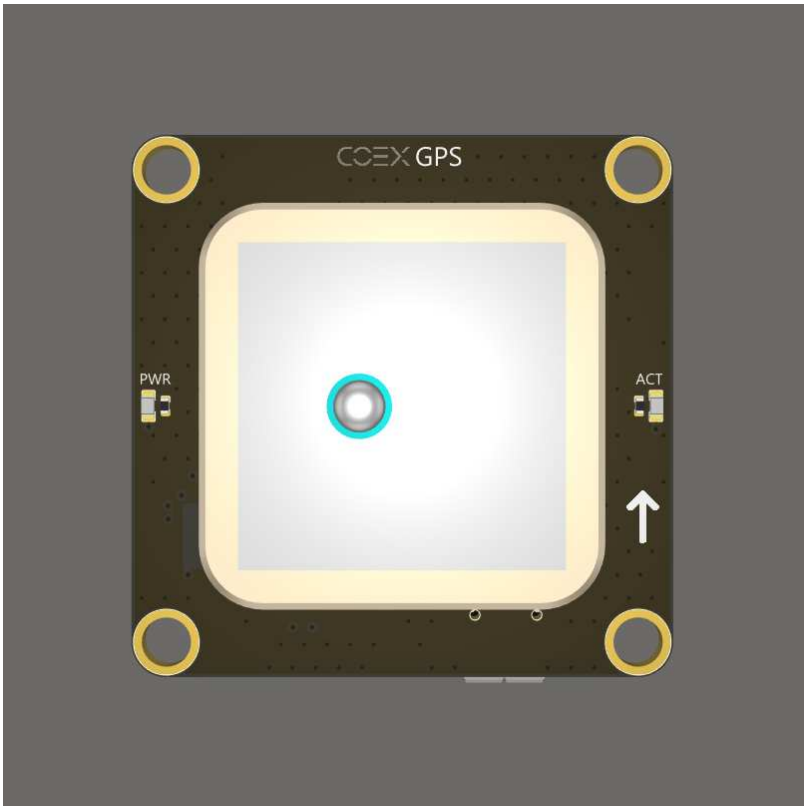
COEX GPS

The GNSS receiver **COEX GPS** is compatible with the COEX Pix flight controller. This receiver comes with a COEX Clover Drone Kit.

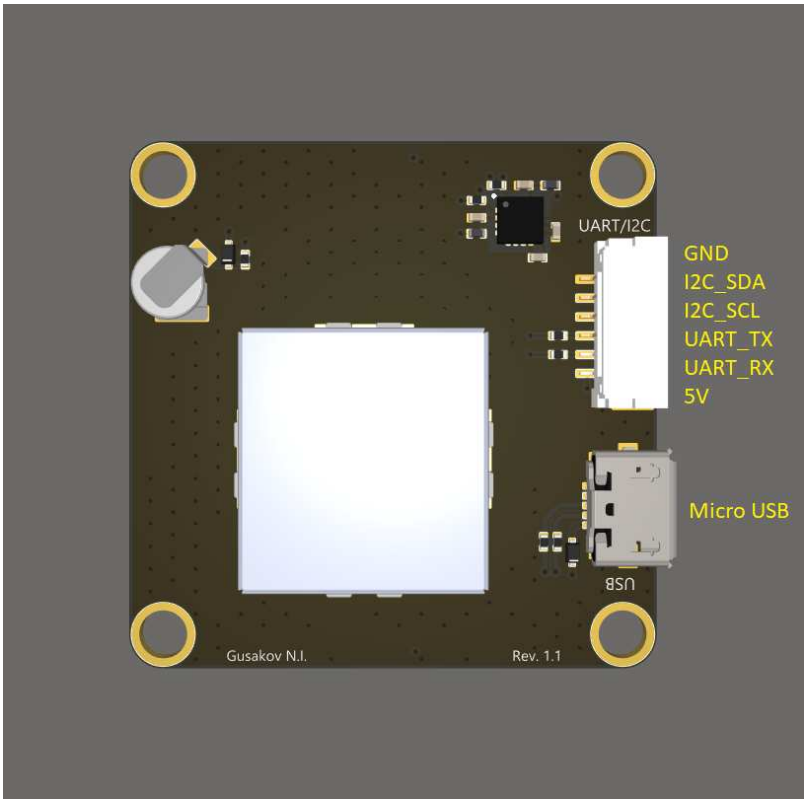
> The source files of the COEX GPS board are published under the CC BY-NC-SA license.

Port pinouts

Top view



Bottom view


UART/I2C
GND
I2C_SDA
I2C_SCL
UART_TX
UART_RX
5V
Micro USB
USB
Gusakov N.I.
Rev. 1.1

Connecting to Raspberry Pi using SSH keys

This instruction will allow you to quickly connect to the Raspberry Pi. In just one second. Without entering a password.

Basic information on working with SSH can be found in the section SSH access to Raspberry Pi. In this section you will find advanced information on using SSH, as well as a number of recommendations on using SSH when working with Clover.

General information

SSH (*secure shell*) is a network protocol that allows you to remotely control the operating system on the computer you are connected to. It is similar to a protocol such as *telnet*, but allows you to encrypt network traffic during interaction. Thus, the transfer of passwords and other secret information is hidden. The Raspberry Pi operating system supports SSH communication, like many other common Linux-based systems.

SSH allows you not only to organize work in the command shell, but also to transfer files, as well as tunnel transmitted data from other protocols, such as information from a video camera or telemetry. In addition, SSH supports several authentication modes (that is, verification of the connecting user), with its help it is possible to connect to the Clover not only using a password, but also password-free access (authentication by a key pair, i.e. SSH keys).

Password authentication

Authentication by password on the image of RPi for Clover is enabled by default and the password can be used to enter into the command shell of the minicomputer. On computers with Linux operating systems (and primarily on servers connected to the Internet), the ability to login with a password is usually disabled, since there is a more secure authentication method.

> It is not recommended to disable logging into Clover by password, since you can completely lose access to the command shell over the network.

When connecting to RPi for the first time, you will see the notification with a suggestion to save a unique *fingerprint*. The stored information is accumulated on computers from which SSH login to RPi is performed, and is checked for sudden substitution.

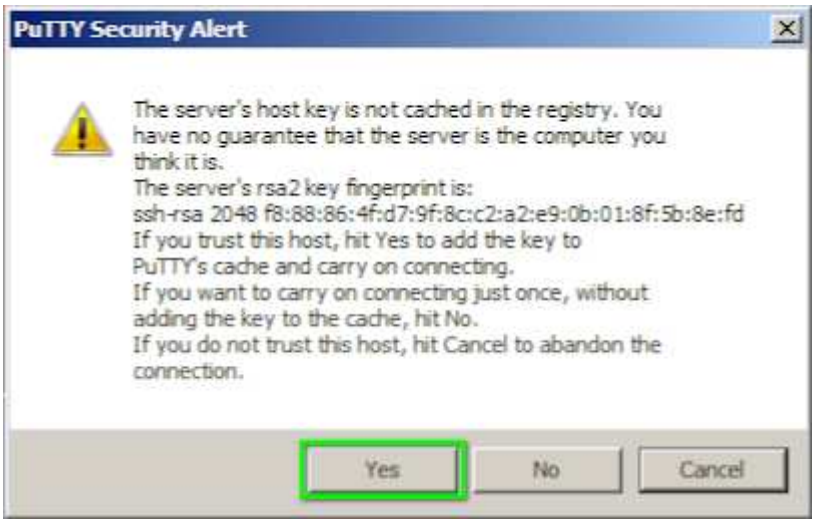
On Linux and Unix (Mac OS) the first SSH-connection to the RPi looks like this:

```
ssh pi@192.168.11.1
# The authenticity of host '192.168.11.1 (192.168.11.1)' can't be established.
# ED25519 key fingerprint is SHA256:4w/7MqTgrtsqPwKnVAMISpouaOJNqzUew2NkJjldMWI
# This key is not known by any other names
# Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
# Warning: Permanently added '192.168.11.1' (ED25519) to the list of known host
# pi@192.168.11.1's password: *********
# Linux clover-3270 5.10.17-v7l+ #1414 SMP Fri Apr 30 13:20:47 BST 2021 armv7l

whoami
# pi

exit
```

In graphical programs in Windows, you will periodically see window with similar warnings.



Windows 10 has a built-in SSH client that can be run from the command line, see the Microsoft usage guide at this link.

Authentication using SSH keys

SSH keys are a convenient, fast alternative way to connect to the Raspberry Pi, which does not require entering a password. In particular, when operating with Clover, this method is convenient because it saves time, and therefore battery power, and the time limit allocated for events in flight zones. In addition, using SSH keys opens up opportunities for convenient use of other programs that you would hardly use if you needed to type a password every time.

The SSH key is divided into two parts: the pair consists of a so-called *private* and *public* key. The key is generated once. One part of the key (open) is transferred once to the remote computer to which the connection will be made, the second part of the key (private) is stored on the computer that will connect, the private part of the key is not transferred anywhere.

> The public key is copied once to the Raspberry Pi, and the private key is stored in the laptop as a file.

Preparation

In order for a key pair to appear, it must be generated. In Linux and Unix (Mac OS), there is a program `ssh-keygen` with which we will get the key pair we need (**attention!** commands are executed not in Raspberry Pi, and not in the virtual machine of the Gazebo simulator, but in the command shell of the laptop from which you will connect to the Clover):

Before using the keys, you need to perform a number of actions to configure access rights *on the laptop*:

```
# one-time setting of access rights to user directories
chmod o-rwx $HOME
mkdir ~/.ssh
chmod g-rwx,o-rwx ~/.ssh
touch ~/.ssh/config ~/.ssh/known_hosts
chmod 600 ~/.ssh/config ~/.ssh/known_hosts
```

> The `.ssh` directory in the user's home folder is the standard storage location for both key pairs and SSH connection settings, so we prohibit access to it by the Others group (*outsiders*). Modern Linux distributions check access rights to files in the `.ssh` directory and may refuse authentication by key pairs.

Generating an SSH key pair

Generating a pair of SSH keys in the `~/.ssh` directory on the laptop:

```
ssh-keygen -f ~/.ssh/id_clover -C "SSH key for Clover" -N ""
# Your identification has been saved in /home/galina/.ssh/id_clover
# Your public key has been saved in /home/galina/.ssh/id_clover.pub

chmod 400 ~/.ssh/id_clover*
```

Copying SSH key to Raspberry Pi

After that connect to Raspberry Pi via WiFi and continue to enter commands *on the laptop* to copy the key to the minicomputer:

```
ssh-copy-id -i ~/.ssh/id_clover.pub pi@192.168.11.1
# pi@192.168.11.1's password: *********
```

As a result, the so-called *public* part of the key will be copied from the laptop to the RPi microcomputer, and the *private* part will remain on the laptop. To verify the connection without entering a password, use the command indicating the path where the SSH key is located:

```
ssh -i ~/.ssh/id_clover pi@192.168.11.1
```

If the terminal does not require you to enter a password to connect to the RPi, then you did everything correctly and the SSH key pair works. Now you can type the exit command from the SSH terminal to continue configuring the laptop:

```
pi@clover-3270:~ $ exit
# logout
# Connection to 192.168.11.1 closed.

galina@Thinkpad-X1:~/.ssh$
```

Configuring SSH connection to Clover

Now let's set up the SSH terminal in such a way that you don't have to enter the path to the private key every time. This is done by editing the `~/.ssh/config` file *on a laptop*. Open the file in a text editor and add the following lines to the file (if there is already some information there, then put them at the end of the file):

```
Host 192.168.11.1
    User pi
    IdentityFile ~/.ssh/id_clover
    PreferredAuthentications publickey,password
    PubkeyAuthentication yes
    PasswordAuthentication yes
    ConnectTimeout 1
    TCPKeepAlive yes
    ServerAliveInterval 2
    ServerAliveCountMax 3
    StrictHostKeyChecking no
```

This setting:

- affects the operation of the SSH terminal when connected to a computer with the IP address `192.168.11.1` ;
- if the user name is not specified, the name `pi` will be used automatically;
- the private key `~/.ssh/id_clover` will be used automatically;
- if the key does not fit for some reason (it was replaced on one laptop, but forgot to replace it on another), then the SSH terminal will switch to password authentication (settings `PreferredAuthentications` , `PubkeyAuthentication` , `PasswordAuthentication`);
- if communication with RPi cannot be established (WiFi is not yet connected), then the SSH connection will not hang, but will be completed quickly (setting `ConnectTimeout`);
- if the connection with RPi is suddenly severed, the SSH connection will not hang, but will be completed quickly (settings `TCPKeepAlive` , `ServerAliveInterval` , `ServerAliveCountMax`);
- the unique SSH *fingerprints* of the RPi microcomputers mentioned above will no longer be checked (the settings `StrictHostKeyChecking`).

This will solve a lot of inconveniences associated with using SSH connections.

> If you have several Raspberry Pi-based drones in your laboratory, and several laptops, then you can **generate SSH keys once**, copy them to all drones and spread them across all laptops, then you can quickly access any of the drones from any laptop.

Now, to connect to RPi from a Linux terminal, you just need to type `ssh 1[TAB][TAB][ENTER]` and the ip address `192.168.11.1` will be automatically updated on the command line, because the command shell uses addresses from the file `~/.ssh/config` and is able to "guess" your intentions to connect to the Clover. By pressing enter, you will instantly find yourself in the RPi terminal.

> Graphical programs for Windows that support working with SSH keys, which you can use: PuTTY and MobaXterm.

Copying files using SSH

To copy a file `circle_flight.py` from the laptop to the RPi to the user's home folder `pi`, you can also use SSH. To do this, type the command in the command shell:

```
# first we specify 'what' we copy, and then 'where'
scp circle_flight.py 192.168.11.1
```

To copy `output.avi` file from the `examples` RPi' folder to the laptop, use a similar command:

```
# after the ':' character (colon), you can specify the path on the remote compu
# the path specified as './' means the current folder where the file will be co
scp 192.168.11.1:examples/output.avi ./
```

Remote command launch via SSH

To run a command at laptop on the RPi (that is, remotely), you can also use SSH.

Raspberry shutdown command:

```
ssh 192.168.11.1 'sudo shutdown now'
```

Example of a Python script' startup command:

```
ssh -t 192.168.11.1 'ROS_HOSTNAME=`hostname`.local && . /opt/ros/noetic/setup.b
```

In order to remotely start video recording, you can run the command:

```
ssh -t 192.168.11.1 'ROS_HOSTNAME=`hostname`.local && . /opt/ros/noetic/setup.b
```

Step-by-step guide on autonomous flight with Clover 4

> The following applies to image version **0.20** and up. See previous version of the article for older images.

This manual contains links to other articles in which each of the topics addressed is discussed in more detail. If you encounter difficulties while reading one of these articles, it is recommended that you return to this manual, since many operations here are described step by step and some unnecessary steps are skipped.

Raspberry Pi initial setup

- Install Raspberry Pi and a camera on the drone according to the manual.
- Download the system image here.
- Burn the image to the microSD card.
- Insert the card into Raspberry Pi.
- Connect power to Raspberry Pi and wait for the Wi-Fi network to appear. To do this, connect the Raspberry Pi to the computer via the microUSB cable. On Raspberry Pi, the green LED should flash blink. It shows that Raspberry Pi works properly.

  > Before connecting the Raspberry Pi to the computer via USB, you need to remove the 5V power cable from Raspberry Pi. Otherwise, there may be problems with power.

- Connect to Wi-Fi and open the web interface (this article).

  After the first power-up, the network appears with a delay. You need to wait until the system is fully loaded. If the Clover network does not appear in the list of networks for a long time, reopen the window with the network selection. Then the list of networks will be updated.

> Now if you have connected to the Clover's Wi-Fi network, it is recommended to open the local version of this guide, otherwise the links will not work.

- Connect to Raspberry Pi via SSH.

  Web access is the easiest way. Follow the instructions in the article SSH Access.

- You can change the name and password of the network if you want to. See the article "[Network Settings] (network.md # change-password-or-ssid-network-name)". The remaining operations with the network are unnecessary.
- Use the nano editor to edit files. Instructions for working with nano.

> In nano, you can only move the cursor with the arrow keys on the keyboard.

- Reboot Raspberry Pi:

```
sudo reboot
```

The connection will temporary close, a new network will be created and you will need to reconnect to it.

- Make sure that the camera is working correctly. Follow the link http://192.168.11.1:8080 and click `image_raw`.

For more information read "Viewing images from cameras".

If the image is blurry, you need to focus the lens. To do this, twist the lens in one or the other direction. Continue to twist until the image becomes clear.

> The red LED on the camera should be lit: it means that the camera is currently capturing image. If the LED does not light: either the camera is connected incorrectly, or the operating system did not boot yet, or there is an error in settings.

Basic commands

You will need the basic Linux commands, as well as special Clover commands, to work efficiently in the system.

Show list of files and folders:

```
ls
```

Go to certain directory by entering the path too it (catkin_ws/src/clover/clover/launch/):

```
cd catkin_ws/src/clover/clover/launch/
```

Go to home directory:

```
cd
```

Open the file `file.py`:

```
nano file.py
```

Open the file clover.launch by entering the full path to it (it works even if you're in a different directory):

```
nano ~/catkin_ws/src/clover/clover/launch/clover.launch
```

Save file (press sequentially):

```
Ctrl+X; Y; Enter
```

Delete a file or folder called `name` (WARNING: the operation will not request confirmation. Be careful!):

```
rm -rf name
```

Make a new directory called `myfolder`:

```
mkdir myfolder
```

Raspberry Pi complete reboot:

```
sudo reboot
```

Reboot only the `clover` service:

```
sudo systemctl restart clover
```

Perform selfcheck:

```
rosrun clover selfcheck.py
```

Stop a program:

```
Ctrl+C
```

Start a program `myprogram.py` using Python:

```
python3 myprogram.py
```

Journal of the events related to `clover` package. Scroll the list by pressing Enter or Ctrl+V (scrolls faster):

```
journalctl -u clover
```

Open the sudoers file with super user rights (this particular file doesn't open without sudo. You can use sudo to open other locked files or run programs that require super user rights):

```
sudo nano /etc/sudoers
```

Setting Raspberry Pi for autonomous flight

Most of the parameters for autonomous flight are located in the following directory: `~/catkin_ws/src/clover/clover/launch/`.

- Enter the directory:

```
cd ~/catkin_ws/src/clover/clover/launch/
```

The `~` symbol stands for home directory of your user. If you are already in the directory, you can go with just the command:

```
cd catkin_ws/src/clover/clover/launch/
```

> Tab can automatically complete the names of files, folders or commands. You need to start entering the desired name and press Tab. If there are no conflicts, the name will be auto completed. For example, to quickly enter the path to the `catkin_ws/src/clover/clover/launch/` directory, after entering `cd`, you can start typing the following key combination: `c-Tab-s-Tab-c-Tab-c-Tab-l-Tab`. This way you can save a lot of time when writing a long command, and also avoid possible mistakes in writing the path.

- In this folder you need to configure three files:
  - `clover.launch`
  - `aruco.launch`
  - `main_camera.launch`
- Open the file `clover.launch`:

```
nano clover.launch
```

You must be in the directory in which the file is located. If you are in other directory, you can open the file by writing the full path to it:

```
nano ~/catkin_ws/src/clover/clover/launch/clover.launch
```

If two users are editing a file at the same time, or if previously the file was closed incorrectly, nano will not display the file contents, it will ask for permission to display the file. To grant permission, press Y.

If the content of a file is still empty, you may have entered the file name incorrectly. You need to pay attention to the extension. If you entered a wrong name or extension, nano will create a new empty file named this way, which is undesirable. Such file should be deleted.

- Find the following line in clover.launch file:

```
<arg name="aruco" default="false"/>
```

Replace `false` with `true`:

```
<arg name="aruco" default="true"/>.
```

This will activate the ArUco marker detection module.

- Open the file `aruco.launch` :

```
nano aruco.launch
```

- Here you need to activate some parameters. Go to the article for more detail.

  Here is what you should get:

```
<arg name="aruco_detect" default="true"/>
<arg name="aruco_map" default="true"/>
<arg name="aruco_vpe" default="true"/>`
```

- Generate the ArUco markers field. See the article Map-based navigation with ArUco markers for details. To generate markers, you need to enter a command with specific values.

  Here is the example generating command where:

  - marker length = 0.335 m (`length`)
  - 10 columns (x)
  - 10 rows (y)
  - distance between markers on the x axis = 1 m (`dist_x`)
  - distance between markers on the y axis = 1 m (`dist_y`)
  - the first marker's ID = 0 (`first`)
  - the marker map name is default: map.txt
  - the marker map numbering is from the top left corner (key `--top-left`)

```
rosrun aruco_pose genmap.py 0.335 10 10 1 1 0 > ~/catkin_ws/src/clover/aru
```

  In most maps, numbering starts with a zero marker. Also, in most cases, numbering starts from the upper left corner, so when generating, it is very important to enter the key `--top-left` .

> If you choose a different name for your ArUco map, you also need to change it in the `aruco.launch` . Find the line `<param name="map" value="$(find aruco_pose)/map/map.txt"/>` and replace map.txt with your map name.

- Edit the `main_camera.launch` for setting up the camera:

  Read more in the article. "Camera orientation".

  In this file, you need to edit the line with the camera location parameters. The line looks like this:

```
<node pkg="tf2_ros" type="static_transform_publisher" name="main_camera_fr
```

In the file you will find many lines similar to this, but most of them are commented out (i.e. not readable) and only one is uncommented. These are pre-configured settings from which you can choose the one you need.

Comment in XML is `<!--` at the beginning of a line and `-->` at the end of a line. An example of a commented line:

```
<!--<node pkg="tf2_ros" type="static_transform_publisher" name="main_camer
```

An example of an uncommented line (the line will be read by the program):

```
<node pkg="tf2_ros" type="static_transform_publisher" name="main_camera_fr
```

The writing above each of these lines indicates which camera position the line corresponds to. If the camera cable goes forward relative to the drone, and the camera is pointing down, you need to select the following setting:

```
<!-- camera is oriented downward, camera cable goes forward  [option 2] --
```

To select the desired setting, you need to uncomment the corresponding line, and comment out another similar line so that there are no conflicts.

- Save changes. Press sequentially:

```
Ctrl+x; y; Enter
```

- Restart the `clover` service:

```
sudo systemctl restart clover
```

Setting the flight controller

- Flash the flight controller with modified firmware. You can download it here in the section "Flashing the flight controller".
- Instructions for flashing and calibrating the flight controller are in the same article.

> Don't forget to choose the downloaded firmware when you flash the flight controller.

Connecting the flight controller with Raspberry Pi

- Connect the Raspberry Pi and the Pixracer via the microUSB cable. The cable should be tightly fastened and passed through the bottom of the drone to not get into the propellers.
- Connect remotely to the flight controller through QGroundControl.

  All the necessary settings for that are already set in Clover. Now you need to create a new connection in QGroundControl. Use the settings from this article.

Remote controller setup

- Flight modes setup is described in the article "Flight modes".

  Set channel 5 to SwC switch; channel 5 to SwA switch. Or you can use any other switches you like.

Clover selfcheck

Perform selfcheck when you have set up your drone or when you have faced problems. The selfcheck process is described in the article "Automated self checks"

- Run the command:

```
rosrun clover selfcheck.py
```

Writing a program

The article "Simple OFFBOARD" describes working with `simple_offboard` module that helps to easily program a drone. All the basic flight functions are described in this article, as well as code snippets.

- Copy the Python code example from "The use of Python language" section and paste in code editor (e.g. Visual Studio Code, PyCharm, Sublime Text, Notepad++)
- Save the document with .py extension for highlighting the code.
- Add flight logic. The examples of such functions are given in the article. You need to call functions for taking off, flying to point and landing.
- Taking off.

  Use `navigate` function to take off. Add this line at the bottom of the program.

```
navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
```

  Add this line to add delay to the program. It gives you time for doing previously called operation.

```
rospy.sleep(3)
```

> It is important to allocate time to execute the `navigate` function, otherwise the drone, without waiting for the previous command to execute, will immediately proceed to the next. For allocating time, use the `rospy.sleep ()` command. The time in seconds is indicated in parentheses. The function `rospy.sleep ()` refers to the previous `navigate` command, and not to the next. This is the time we give to fly to the point indicated in previous `navigate` (the one that is just above the `rospy.sleep ()`).

- Set the drone's position in the marker field coordinate system.

  For doing that you need to call a `navigate`, set the coordinates and coordinate system (`frame_id`):

```
navigate(x=1, y=1, z=1.5, speed=1, frame_id='aruco_map')
```

- As the result you get:

```
navigate(x=0, y=0, z=1.5, speed=0.5, frame_id='body', auto_arm=True)
rospy.sleep(3)
navigate(x=1, y=1, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(5)
```

> Note that the parameter `auto_arm=True` is only set once on the first takeoff. In other cases it should not be set True because it prevents overtaking the control.

- If you want to add other points for the drone's mission, add another `navigate` and `rospy.sleep()`. Calculate time individually for each point, depending on the speed of flight and the distance between two points.

  If you want to add the points with coordinates (3, 3, 1.5):

```
navigate(x=3, y=3, z=1.5, speed=1, frame_id='aruco_map')
rospy.sleep(3)
```

> Coordinates should not exceed the size of your field. If the field is 4x4 meters in size, the maximum coordinate value is 4.

- After reaching all of the points you need to land. The following line is placed at the end of the program:

```
land()
```

Writing the program to the drone

The easiest way to send the program is to copy the content of the program, create a new file in the command line and paste the program text into the file.

- To create the file `myprogram.py`, run the command:

```
nano myprogram.py
```

You can select any name you want, but it is not recommended to use spaces and special characters. In addition, the program extension should always end with `.py`

- Paste text in the input field. If you use Butterfly web access on Windows or Linux:

```
Ctrl+Shift+V
```

On Mac you can click `Cmd+v` .

- Save the file:

```
Ctrl+x; Y; Enter
```

Starting the program

- It is necessary to carefully prepare the drone, remote control and program before you fly autonomously. Run `selfcheck.py` . Make sure the drone flies well in manual mode.
- Turn on the drone and wait for the system to boot. A red light on the camera means that the system has booted.
- Check drone's flight in POSCTL mode.
- To do this, take off above the markers in STABILIZED mode and turn the SwC switch (or the one you have set) to the lower position - POSCTL mode.

> You need to be ready to immediately switch back to STABILIZED mode if the drone gets out of control!

Set the left stick (throttle) to the middle position. The drone has to hover in place. If so, you can land the drone and proceed to the next step. If not, you need to find the reason for the problem.

- Before you start your program, set the SwC switch to the middle position. It will help you to take control of the drone. For taking control, switch your mode switch (SwC by default) to any other flight mode.
- Set the left stick (throttle) to the middle position so that in case of taking control the drone won't fall down.
- Run the program:

```
python3 my_program.py
```

> After completion of the program , the drone can land incorrectly and continue to fly over the floor. In this case, you need to intercept control.

- If you want to stop the program before it ends, press `Ctrl+C`. If didn't work, press `Ctrl+Z`, but it is not recommended.

Hostname

> The following applies to image version **0.20** and up. See previous version of the article for older images.

By default the hostname of the Clover drone is set to `clover-xxxx`, where `xxxx` are random numbers. These numbers are the same as in the Wi-Fi SSID.

Thus, Clover is accessible on machines that support mDNS as `clover-xxxx.local`. You can use this name to access Clover over SSH:

```
ssh pi@clover-xxxx.local
```

Also, this name can be used in place of IP-address to open Clover web pages in browser, accessing ROS master, etc.

Changing hostname

In some situations it is necessary to change Clover's hostname. You can use the `hostnamectl` utility for that:

```
sudo hostnamectl set-hostname newname
```

Where `newname` is the new name of the machine. `hostnamectl` utility will change the name in `/etc/hostname` file.

You should also put the new name to `/etc/hosts` file:

```
127.0.1.1    newname newname.local
```

Setting `newname.local` is necessary to allow ROS to resolve this name in situations where all the network interfaces are down (when Wi-Fi is turned off or disconnected).

> Changing the hostname does not affect the Wi-Fi SSID (and vice versa, changing the Wi-Fi SSID won't affect the hostname).

PX4 Simulation

> This article is about running a standalone PX4 simulation with a generic quadcopter and **is outdated**. Consider using our configuration for a more Clover-like experience.

Main article: https://dev.px4.io/en/simulation/

PX4 simulation is possible in Linux and macOS with the use of physical environment simulation systems jMAVSim and the Gazebo.

jMAVSim is a lightweight environment intended only for testing multi-rotor aircraft systems; Gazebo is a versatile environment for all types of robots.

Launching PX4 SITL

1. Clone repository from PX4.

```
git clone https://github.com/PX4/Firmware.git
cd Firmware
```

jMAVSim

Main article: https://dev.px4.io/en/simulation/jmavsim.html

For simulation using the jMAVSim lightweight environment, use the following command:

```
make posix_sitl_default jmavsim
```

To use the LPE position calculation module instead of EKF2, use:

```
make posix_sitl_lpe jmavsim
```

Gazebo

Main article: https://dev.px4.io/en/simulation/gazebo.html

To get started, install Gazebo 7. On a Mac:

```
brew install gazebo7
```

On Linux (Debian):

```
sudo apt-get install gazebo7 libgazebo7-dev
```

Start simulation from the Firmware folder:

```
make posix_sitl_default gazebo
```

You can run a simulation in headless mode (without a window client). To do this, use the following command:

```
HEADLESS=1 make posix_sitl_default gazebo
```

Connection

QGroundControl will automatically connect to the running simulation on startup. The operation will be the same as, as in the case of a real flight controller.

To connect MAVROS to the simulation, use the UDP Protocol, a local IP address, and port 14557, for example:

```
roslaunch mavros px4.launch fcu_url:=udp://@127.0.0.1:14557
```

Navigation using vertical ArUco-markers

The algorithm of the navigation through visual ArUco-markers, that was implemented in the Clever image, supports the flexible configuration of the markers in area. It allows you to place them on any surface, at any angle.

Installing the vertical camera mount

For a better recognition of the markers, you need to set the camera vertically so that the lens is pointed parallel to the horizon.

> The configuration file allows you to configure the location of the camera in area relative to the copter in any way. For your convenience, we will review the option of installing the camera at an angle of 90 degrees to the horizon in the direction of the copter's nose.

Camera mount, 3D printing

Print the camera mount.

Install the mount in a convenient location, so that the camera has a minimum number of unnecessary objects (protection, legs, propellers, beams) — all of it will negatively affect the recognition of the markers.

Setting the camera transform

To set the camera position at the desired angle, open the file `main_camera.launch`, located in `~/catkin_ws/src/clover/clover/launch/`.

```
nano ~/catkin_ws/src/clover/clover/launch/main_camera.launch
```

In the parameters `direction_x`, `direction_y`, set empty values manually or enter the following lines:

```
sed -i "/direction_z/s/default=\".*\"/default=\"\"/" /home/pi/catkin_ws/src/clo
sed -i "/direction_y/s/default=\".*\"/default=\"\"/" /home/pi/catkin_ws/src/clo
```

Edit one of the configuration lines or add the line shown bellow:

```
<node pkg="tf2_ros" type="static_transform_publisher" name="main_camera_frame"
```

. Only one camera configuration can be used at a time. If you insert the line above, don't forget to comment the currently active one. The syntax highlighting system will help you determine that — the active line will be highlighted in a different color than the comments. To comment, add the `<!--` and `-->` symbols at the beginning and the end respectively.

If you are using the marker map, where the markers have equal distances along the x and y axes, you can use script for creating markers map `gen_map.py` . Otherwise, you will need to set them manually. To do this, go to the directory `map_name.txt` and create a map file. Fill out your map according to the map syntax. Here is an example of a marker map with a random marker location:

. When filling out the map, select one of the markers as the origin, and measure the distance to all other markers relative to it. If all your parameters are oriented same way, you can choose not to specify all 8 parameters, but only the first 5: the marker index, size, and its location in space along the x, y, and z axes, respectively.

```
106 0.33    0   0   0
103 0.33    1.53    0.23    0
153 0.40    -0.56   1.36    0
```

After you fill out the map, you need to apply it. To do it, edit the file `aruco.launch` , located in `~/catkin_ws/src/clover/clover/launch/` . Change the line `<param name="map" value="$(find aruco_pose)/map/map_name.txt"/>` , where `map_name.txt` is the name of your map file.

If you are using markers that are not linked to horizontal surfaces (floor, ceiling), you must blank the `placement` argument in the same file:

```
<arg name="placement" default=""/>
```

After all the settings, call `sudo systemctl restart clover` to restart the `clover` service.

Configuring the PID coefficients

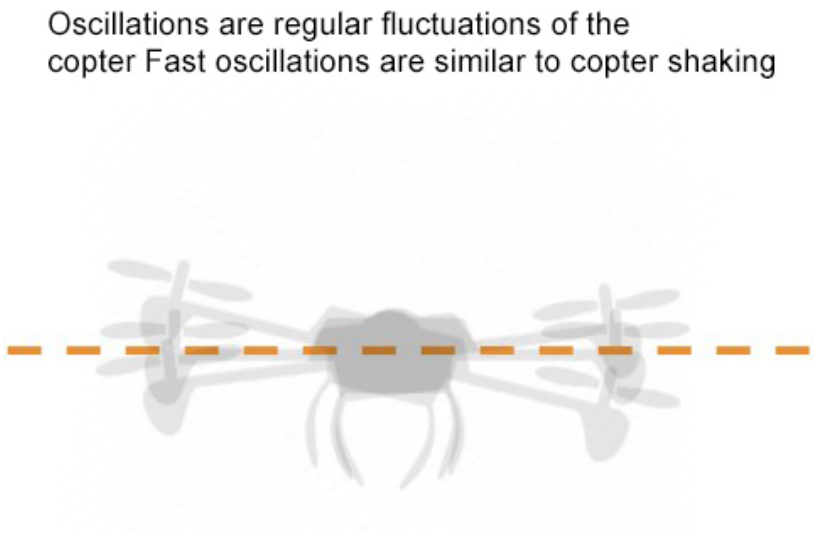
In practice, the most common problem is the presence of rapid oscillations that occur due to the too large values of parameter P. In this situation, you should decrease this value (all parameters are set experimentally, based on copter behavior).

It is also worth checking that the oscillations do not occur during sharp descent (otherwise, reduce P). Slow copter rocking from side to side while trying to maintain the predetermined position is related to excessive value I. If the copter swings during movement, the value should be increased. If the copter keeps the preset position poorly, you should increase the D parameter; it the D parameter is too high or too low, oscillations occur.

> Adjustment of the D parameter should start with the minimum values, which are 3 – 4 times lower than default values, if any.

The Rate Pitch and Rate Roll parameters should be the same.

YAW parameters should be changed individually, according to the above instruction (usually the yaw doesn't require serious adjustment, you may leave it default).



An example of ROLL oscillation

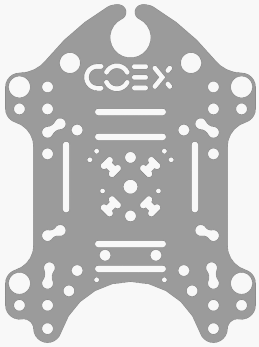
Model files for parts

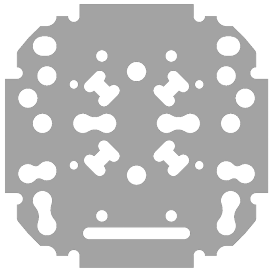
This page contains models and drawings of some of the drone parts. They can be used for 3D printing and/or laser cutting replacement parts.

Clover 4.2

Milling

Preview	Part	Files

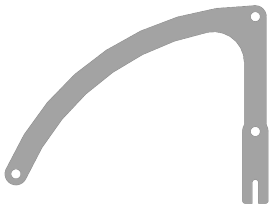
	**Deck mount**. Function: Deck for installing battery and Raspberry Pi Material: Monolithic polycarbonate 2mm. Quantity: 1 pcs.	`Deck Mount.dxf`
		`Deck Mount.ipt`
		`Deck Mount.stp`

	**Deck mount small**. Function: Deck for mounting FPV cameras and mounting stiffening plates. Material: Monolithic polycarbonate 2mm. Quantity: 1 pcs.	`Deck Mount Small.dxf`
		`Deck Mount Small.ipt`
		`Deck Mount Small.stp`

	**Grab deck**. Function: Deck for installing grippers and external peripherals (camera, rangefinder). Material: Monolithic polycarbonate 2mm. Quantity: 1 pcs.	`Grab Deck.dxf`
		`Grab Deck.ipt`
		`Grab Deck.stp`

	LED mount plate. Function: Fixing the LED strip. Material: Monolithic polycarbonate 2mm. Quantity: 1 pcs.	`LED mount plate.dxf`
		`LED mount plate.ipt`
		`LED mount plate.stp`

	**Prop guard**. Function: Prevent damage of propellers. Material: Monolithic polycarbonate 2mm. Quantity: 4 pcs.	`Prop Guard.dxf`
		`Prop Guard.ipt`
		`Prop Guard.stp`

	**Prop guard mount**. Function: Arc for securing the guard. Material: Monolithic polycarbonate 2mm. Quantity: 2 pcs.	`Prop Guard Mount.dxf`
		`Prop Guard Mount.ipt`
		`Prop Guard Mount.stp`

	**Small leg**. Function: Standard footing element. Material: Monolithic polycarbonate 2mm. Quantity: 2 pcs.	`Small Leg.dxf`
		`Small Leg.ipt`
		`Small Leg.stp`

	**Arm**. Function: motor mount. Material: carbon fibre 2mm. Quantity: 4 pcs.	`Arm.dxf`
		`Arm.ipt`
		`Arm.stp`

	**Central plate**. Function: a place to mount PCBs. Material: carbon fibre 2mm. Quantity: 1 pc.	`Central Plate.dxf`
		`Central Plate.ipt`
		`Central Plate.stp`

	**Small deck mount**. Function: holds stiffener plates. Material: carbon fibre 2mm. Quantity: 1 pc.	`Deck Mount Small.dxf`
		`Deck Mount Small.ipt`
		`Deck Mount Small.stp`

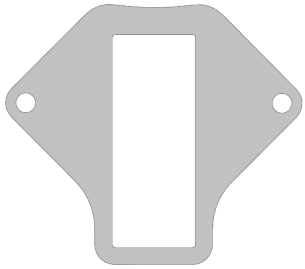
	**Stiffener plate**. Function: frame stiffener. Material: carbon fibre 2mm. Quantity: 4 pcs.	`Stiffener Plate.dxf`
		`Stiffener Plate.ipt`
		`Stiffener Plate.stp`

Clover 4.2 WorldSkills

Milling

Preview	Part	Files

	**Big leg**. Function: Extended footing element. Material: Monolithic polycarbonate 2mm. Quantity: 2 pcs.	`Big Leg.dxf`
		`Big Leg.ipt`
		`Big Leg.stp`

	**Grip spacer**. Function: spacer for the gripper plates. Material: monolithic polycarbonate 2mm. Quantity: 1 pcs.	`Grab Spacer.dxf`
		`Grab Spacer.ipt`
		`Grab Spacer.stp`

3D print

Mechanical gripper

- **Left claw**: `grip_left.stl`.
- **Right claw**: `grip_right.stl`.

Material: SBS Glass. Infill: 100%. Quantity: 1 pcs.

Clover 4

3D print

- Battery holder – `battery_holder.stl`. Filament: PLA/ABS/SBS. Infill: 50% or more.

Laser cut

- Reinforcing Pad – `reinforcing_pad.dxf`

Contributed models for Clover 4

Reinforced mounting plate for Jetson Nano and additional equipment by Vyacheslav Buzov.

Laser cut

- Reinforced plate base (for Jetson Nano) – `reinforced_plate_base.dxf`
- Reinforced plate rib – `reinforced_plate_rib.dxf` (x2)
- Camera pad for reinforced plate – `reinforced_plate_camera_pad.dxf`

Clover 3

3D print

- Camera case – `camera_case.stl`. Filament: PLA/ABS/SBS.
- Camera mount – `camera_mount.stl`. Filament: PLA/ABS/SBS.
- Camera plate – `camera_plate.stl`. Filament: PLA/ABS/SBS.
- Mounting deck small – `mounting_deck_small.stl`. Filament: PLA/ABS/SBS.

Laser cut

- Big leg – `big_leg.dxf`.
- Deck mount – `deck.dxf`.
- Prop guard – `prop_guard.dxf`.
- Prop guard fork – `prop_guard_mount.dxf`.
- Spacer – `grab_spacer.dxf`.
- Leg – `leg.dxf`.

- LED mount plate – `led_mount_plate.dxf`.
- Mounting deck small – `mounting_deck_small.dxf`.

Milling

- Central plate – `central_plate.dxf`.
- Arm – `arm.dxf`.

ROS Melodic package installation and setup

In order to use tools such as rqt, rviz and others as well as running the simulator (SITL), you will need to install and setup ROS package

> For more details on installation refer to the main article.

ROS Melodic installation on Ubuntu

To find the correct package version, you will need to change the settings of your repositories. Go to "Software and updates" and enable `restricted`, `universe` and `multiverse`.

Set up your system so that software form `packages.ros.org` can be installed :

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) ma
```

Configure access keys in your system for correct download:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key C1
```

Make sure that your packages are up to date:

```
sudo apt-get update
```

Now you can install the ROS package itself.

- If you plan to use ROS together with the simulator (also includes tools such as rqt, rviz and others):

  ```
  sudo apt-get install ros-melodic-desktop-full
  ```

- If you plan to use ROS exclusively for tools rqt, rviz etc.:

  ```
  sudo apt-get install ros-melodic-desktop
  ```

After the package has installed, initialize `rosdep`. Package `rosdep` will allow to easily install dependencies for the source files that you whish to compile. Running some essential components of ROS will as well require this package.

```
sudo rosdep init
rosdep update
```

If you are not comfortable with entering environment variables manually each time, you may configure it in a way that it add itself in your bash session on every new shell startup:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

If you whish to install any additional packages for your ROS Melodic simply use:

```
sudo apt-get install ros-melodic-PACKAGE
```

Camera calibration

Camera calibration can significantly improve the quality of nodes related to computer vision: ArUco markers detection and optical flow.

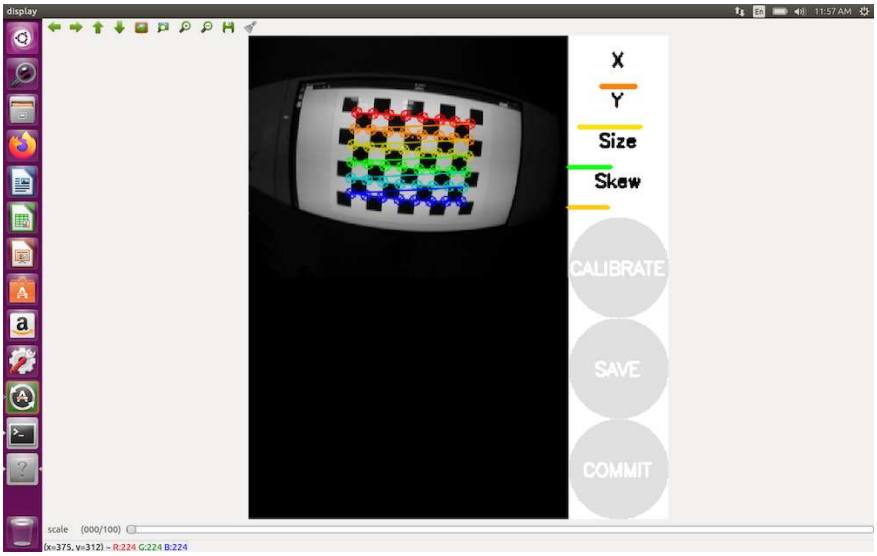
Camera calibration process allows to define the parameters reflecting the specific lens installed. These parameters include focal lengths, principal point (which depends on camera lens placement regarding the centre), distortion coefficient *D*. You can read more about camera distortion model used in the OpenCV documentation.

There are several tools allowing to calibrate the camera and store calculated parameters into the system. Usually they use calibration images, "chessboards" or combinations of "chessboards" and ArUco-marker grids (ChArUco).

camera_calibration ROS-package

Main tutorial:
http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration.

In order to calibrate the camera with the `camera_calibration` ROS-package you need a computer with OS GNU/Linux and ROS Noetic installed.



1. Using the Terminal, install `camera_calibration` package to your computer:

   ```
   sudo apt-get install ros-noetic-camera-calibration
   ```

2. Download the chessboard – chessboard.pdf. Print the chessboard on paper or open it on the computer screen.

3. Connect to the Clover Wi-Fi network.

4. Run camera calibration (on your computer):

```
ROS_MASTER_URI=http://192.168.11.1:11311 rosrun camera_calibration camerac
```

> Change the value *0.108* to actual size a square on the chessboard in metres. For example, value *0.03* corresponds to 3 cm.

5. When the calibration program starts, move the drone so the calibration board is observed from different angles:
   - Place the chessboard in the left, right, top and bottom part of the frame.
   - Rotate the chessboard around all 3 axes.
   - Move camera toward and away from the chessboard, so that it is observed from different distance.
6. Click the *CALIBRATE* button, when it's active. The process of calculation will take several minutes.

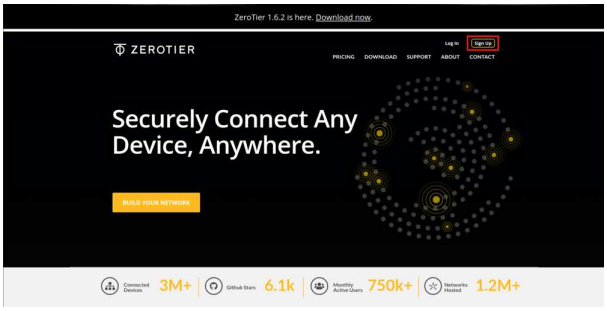
   When the calculation is done, you'll see calculated parameters in the terminal. The corrected camera image view will be displayed as well. If calibration was successful all straight lines will remain straight on the image displayed.

7. Click the *COMMIT* button to store calculated calibration parameters. The result will be stored in the main Clover camera calibration file: `/home/pi/catkin_ws/src/clover/clover/camera_info/fisheye_cam_320.yaml`.

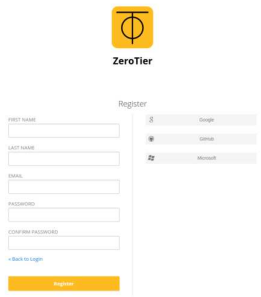
Creating a virtual network ZeroTier One and connecting to it

Creating and configuring a ZeroTier network

1. Go to ZeroTier website.

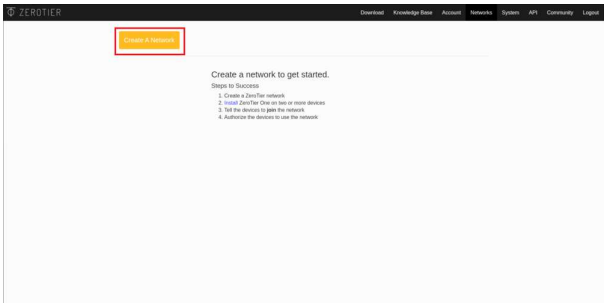


2. Sign up on ZeroTier.

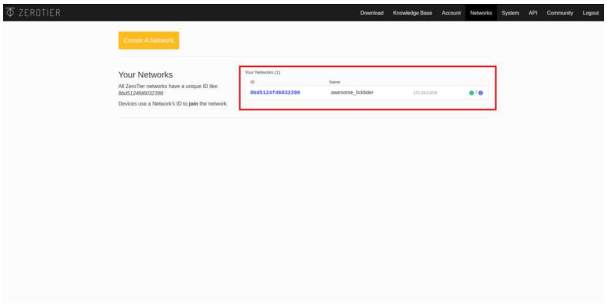


3. Go to your account.

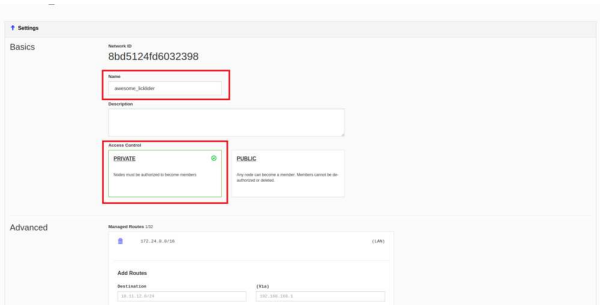
4. Click on the *Create A Network*.



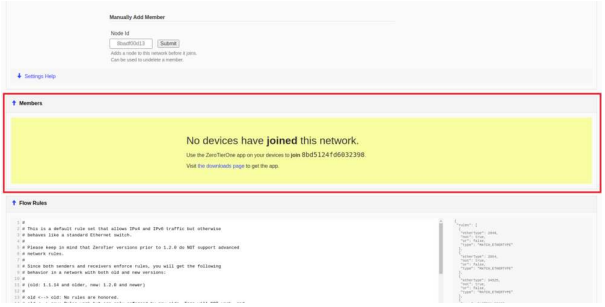
5. After that, you will see the network you created, its ID and name. Click on the network to configure it.



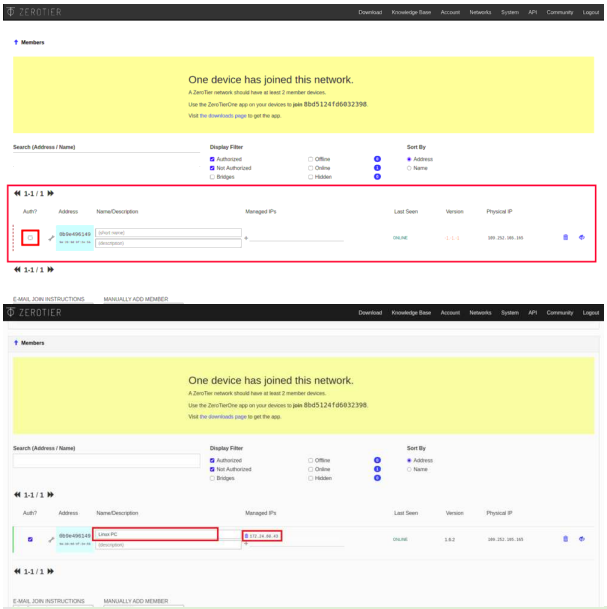
6. In the window that appears you can change the network name and connection privacy.



7. Scroll down to the *Members* column. It will say that there are no users on the network.



8. Devices connected to the network will be displayed in this column. To allow them to connect to the network, activate the *Auth?* checkbox. The connected device will automatically be given an internal IP address, which will then be used to communicate with this device.



specify names for new devices, it will help you distinguish them from each other in the future.

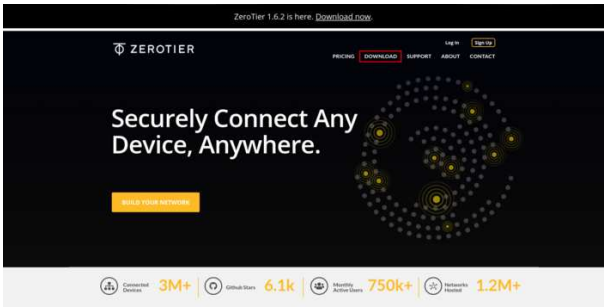
9. Repeat the last step for all the devices that you want to connect.

> ZeroTier network supports up to 50 users simultaneously for free use.

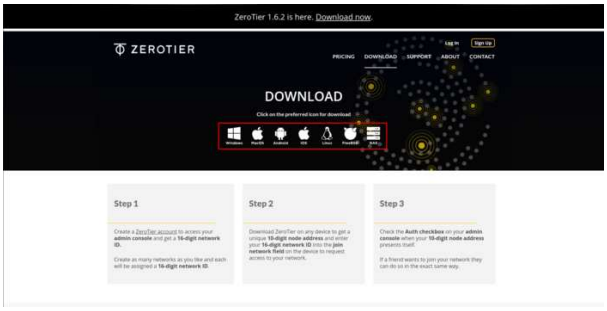
Setup on Windows

Installing the app

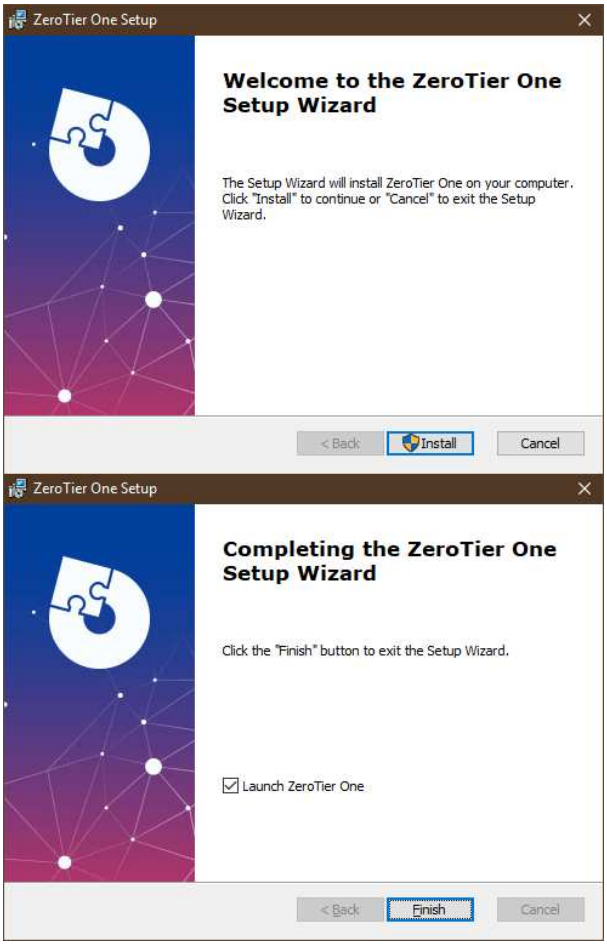
1. Go to the ZeroTier website.



2. Click on the Windows icon.

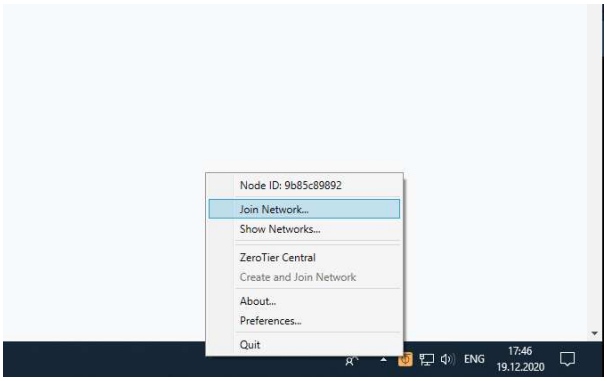


3. Download and run the `ZeroTier One.msi` file.

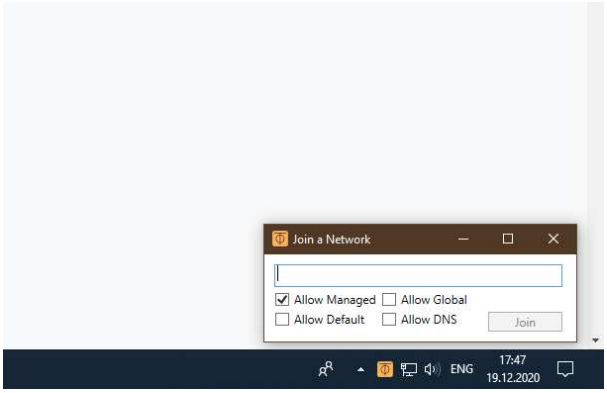


Network connection

1. Run ZeroTier One.
2. Click on the ZeroTier One icon in the taskbar.
3. Click on the *Join Network...* to connect to the network.



4. In the window that appears, enter your network ID and click *Join*.

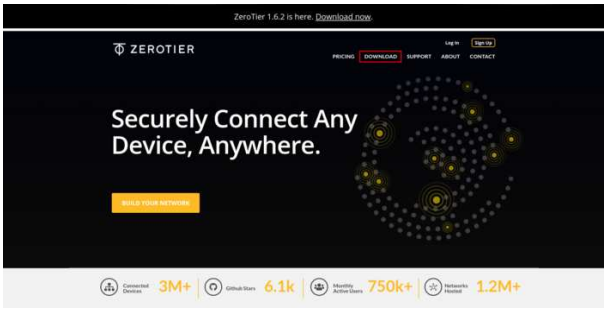


5. Allow using the new network.

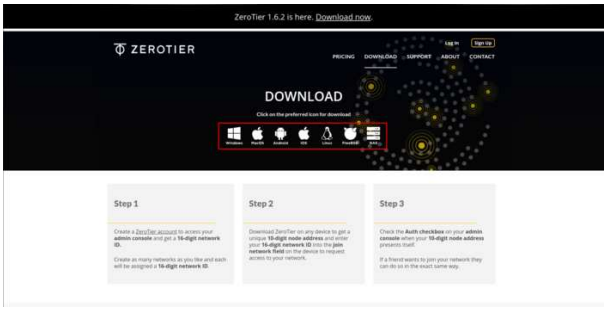
Setup on iOS

Installing the app

1. Go to the ZeroTier website.



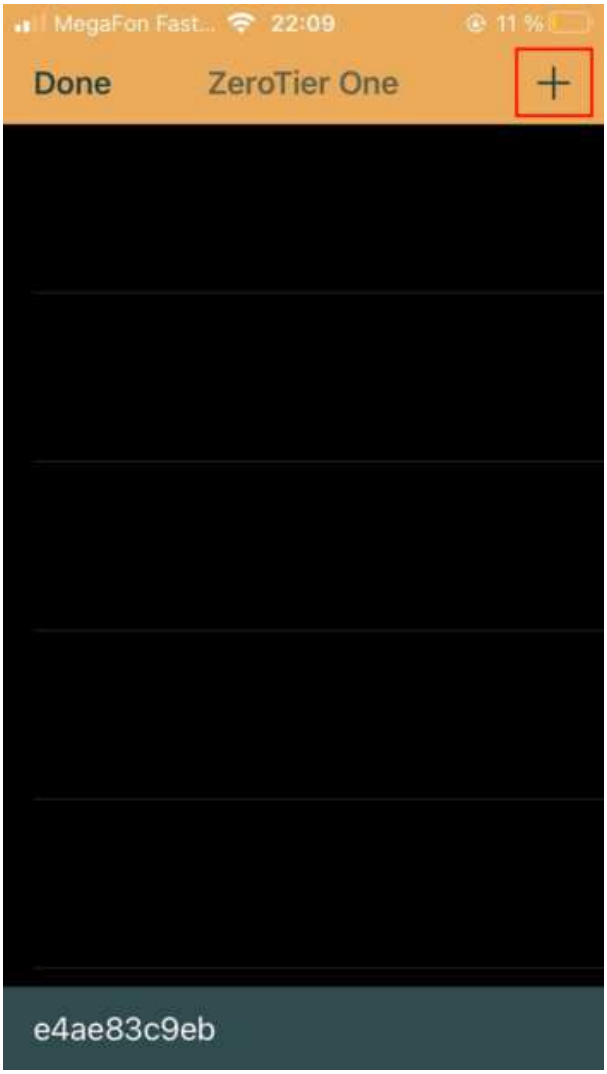
2. Click on the iOS icon.



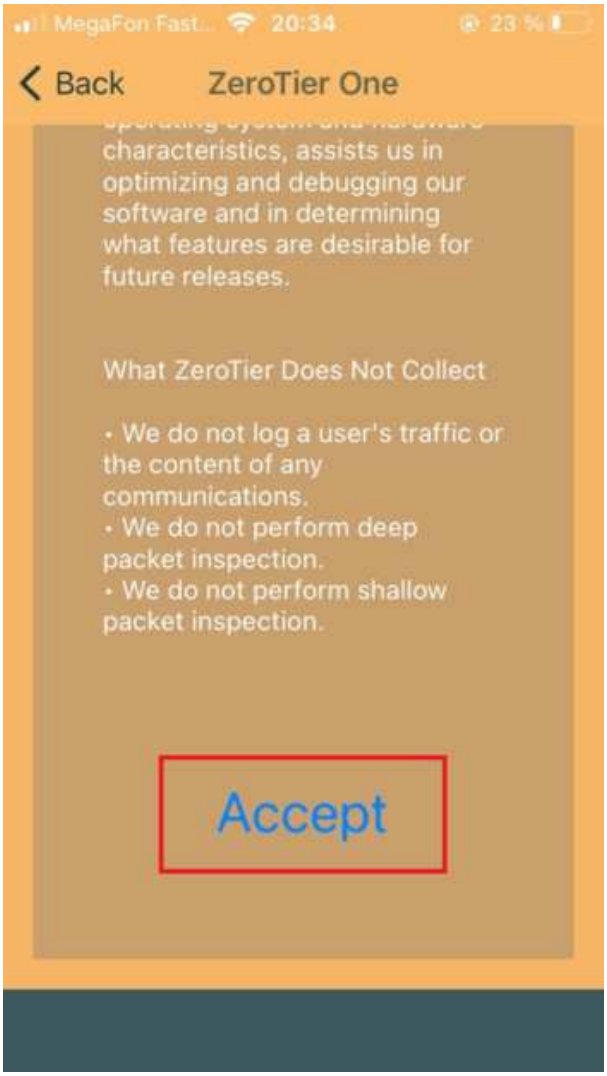
3. Install the *ZeroTier One* app.

Network connection

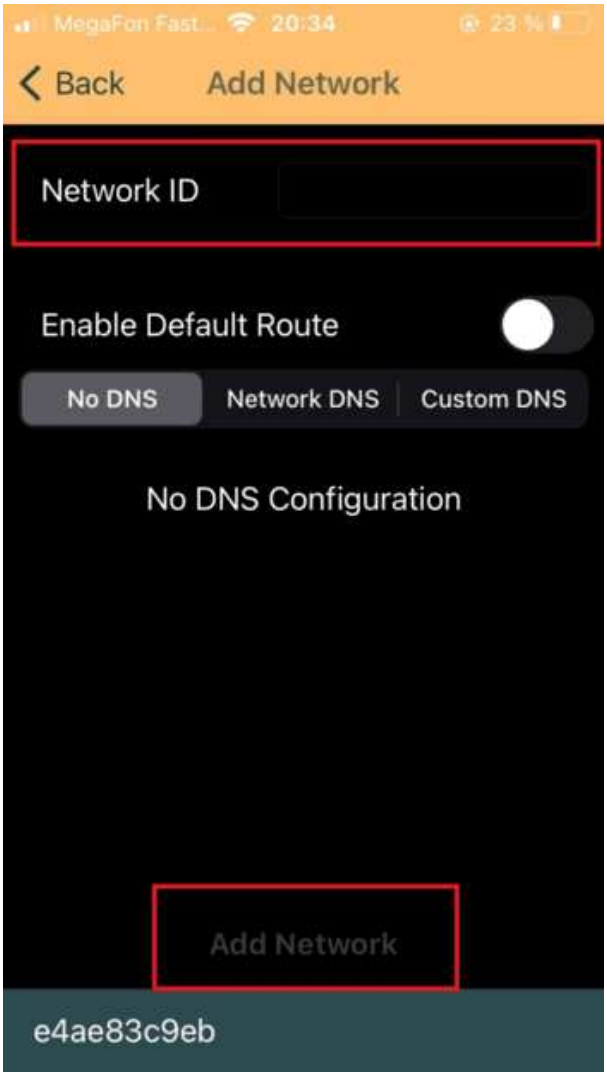
1. Run ZeroTier One app.
2. Click on + to add a new connection.



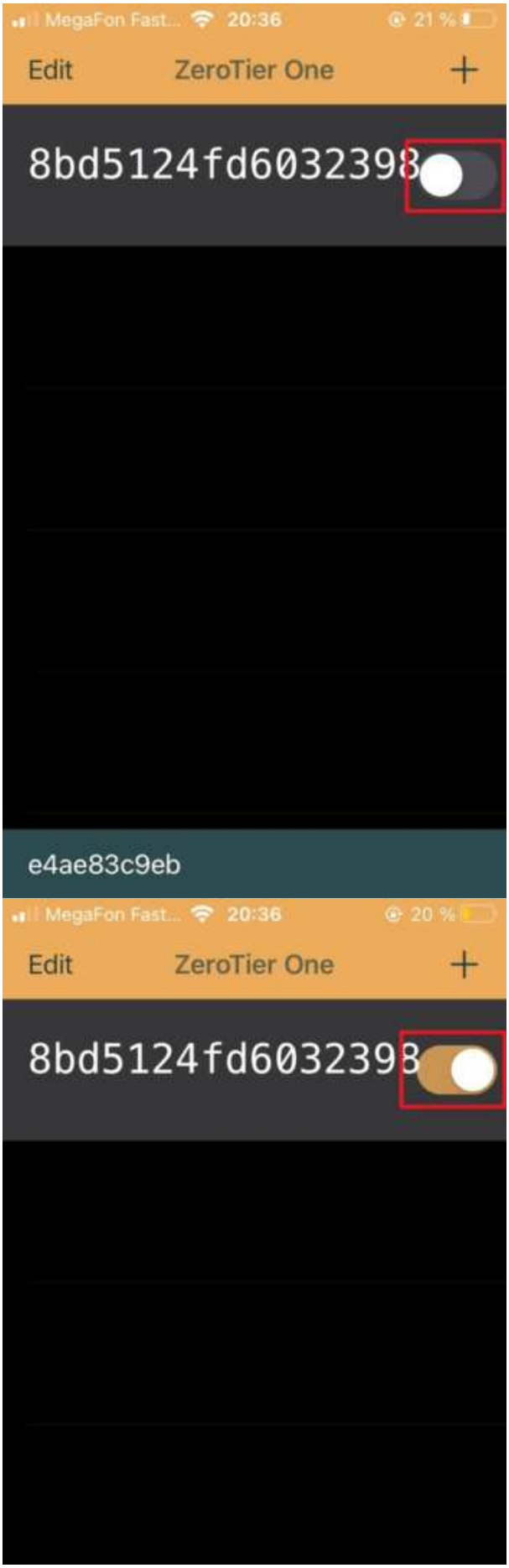
3. Confirm the privacy policy.

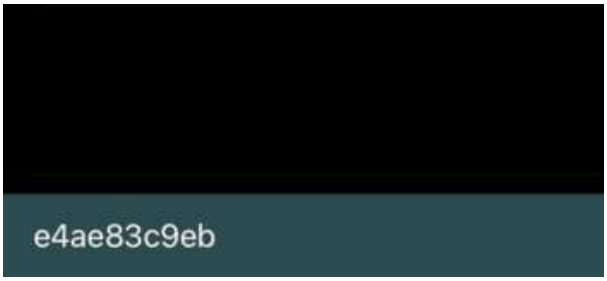


4. Enter your network ID and click *Add Network*.



5. Confirm adding the new VPN configuration.
6. Connect to the VPN network by sliding the network activation slider.


MegaFon Fast... 20:36 21 %
Edit
ZeroTier One
8bd5124fd6032398
e4ae83c9eb
MegaFon Fast... 20:36 20 %
Edit
ZeroTier One
8bd5124fd6032398



Setup on Linux (PC, Raspberry Pi)

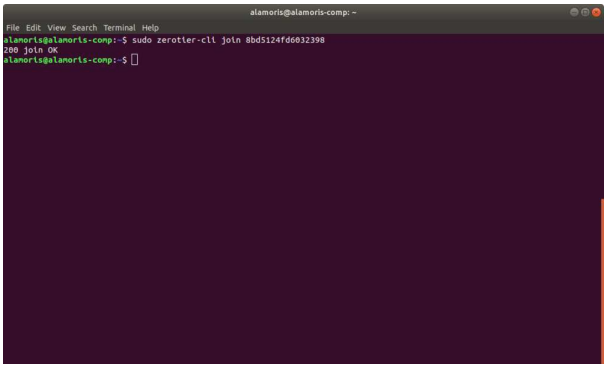
Installing the app

1. Open the console by pressing the keyboard shortcut *ctrl* + *alt* + *t* or type *terminal* in the program search bar.
2. Enter the ZeroTier installation command.

```
curl -s  https://install.zerotier.com  | sudo bash
```

Network connection

1. Open the console.
2. Enter the command `sudo zerotier-cli join network-id`, where `network-id` is your network ID.

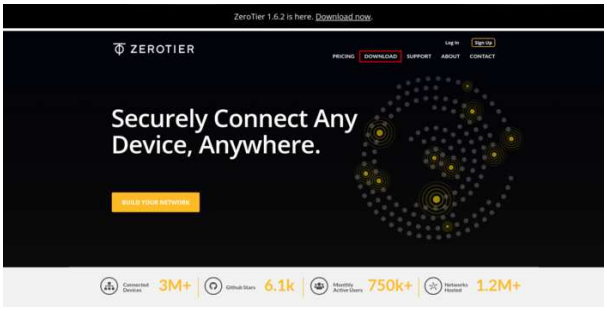


3. If the connection is successful, the corresponding message will be displayed in the console.

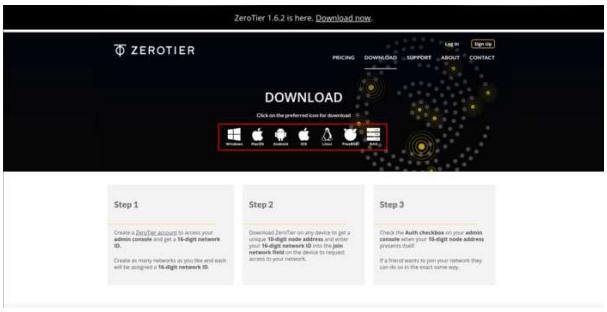
Installing and configuring on macOS

Installing the app

1. Go to the ZeroTier website.



2. Click on the macOS icon.



3. Download and run `ZeroTier One.pkg` file.
4. Install the ZeroTier One app.

Network connection

1. Run ZeroTier One app.
2. Click on the ZeroTier One icon in the taskbar .
3. In the window that appears, click on *Join Network...*.

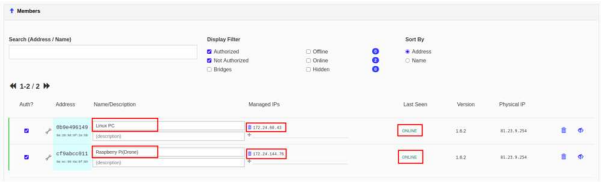


4. In the *Enter Network ID* field, enter your network ID.

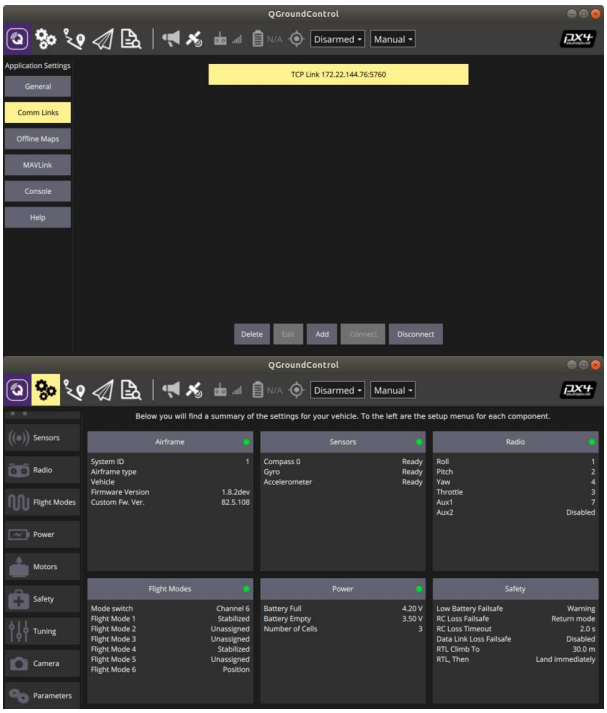


Connecting to the copter

1. Make sure that ZeroTier is working and connected to the network on the drone and control device. To do this, make sure that these have an *Online* status.



2. Make sure that all devices have local IP addresses - *Managed IPs*.
3. Open GQC and in the *Comm Links* tab add a TCP connection specifying the IP of the drone. Read more about remote connection here.



Multi-copter control with 4G communication

The fourth generation mobile communication is a convenient tool for transmitting and receiving information at high speed. Nowadays, the coverage area of mobile operators allows to connect to the Internet at high speed from almost any point.

> To transfer any data from your drone to the ground control station (e.g., QGroundControl) and back, you need to set up your own VPN network.

Connecting 4G modem to Raspberry Pi

Connect a 4G modem with SIM card to the USB port of your Raspberry Pi.

When connected, some modems recognize in the system as a network card, without any additional settings.

4G modem example: *USB 4G Huawei E3372H*



But some other popular modems, for instance *Quectel EP06*, do not start the internet connection automatic. In this case you should use utilities like `qmi-network` and `udhcpc`. To install this utilities enter the appropriate command line:

```
sudo apt install libqmi-utils udhcpc
```

Next to start the internet connection proceed following:

```
sudo ip link set wwan0 down
echo 'Y' | sudo tee /sys/class/net/wwan0/qmi/raw_ip
sudo ip link set wwan0 up
sudo qmi-network /dev/cdc-wdm0 start
sudo udhcpc -q -f -i wwan0
```

Reed more about it in this article.

To check the internet connection enter the appropriate command line:

```
ping -I wwan0 -c 5 8.8.8.8
```

To check the speed of the internet connection you could use the `speedtest` utility:

```
sudo apt install speedtest-cli
speedtest
```

Connecting Raspberry Pi to the VPN

Create the VPN network keys to connect Raspberry Pi and the ground station.

To connect Raspberry Pi to your network, install the OpenVPN package:

```
sudo apt-get install openvpn
```

Move your keys to the `/etc/openvpn/client` directory. For convenience, use the graphical SFTP data transfer interface, for example: WinSCP, FileZilla, etc.

To enable the client mode, you must activate the keys you have transmitted. Keys can be generated in various formats, for example: `.ovpn`, `.conf`. The key or configuration used on your copter should be strictly in `.conf` format.

Initialize the service that uses your keys to connect in client mode:

```
sudo systemctl enable openvpn-client@config-name
```

where *config-name* is the name of your configuration file.

If everything is done correctly, every time the system restarts, the service client will automatically connect to your network.

Before starting work, do not forget to set up and enable VPN connection on your PC.

Alternatively we recommend to use the ZeroTier VPN-service.

Copter control via QGroundControl

> We suggest using the UDP transfer protocol to control the drone, which provides less delay, at the cost of no guarantee of receiving the package, which is very important during the flight.

Make sure your copter and ground station are connected to your network.

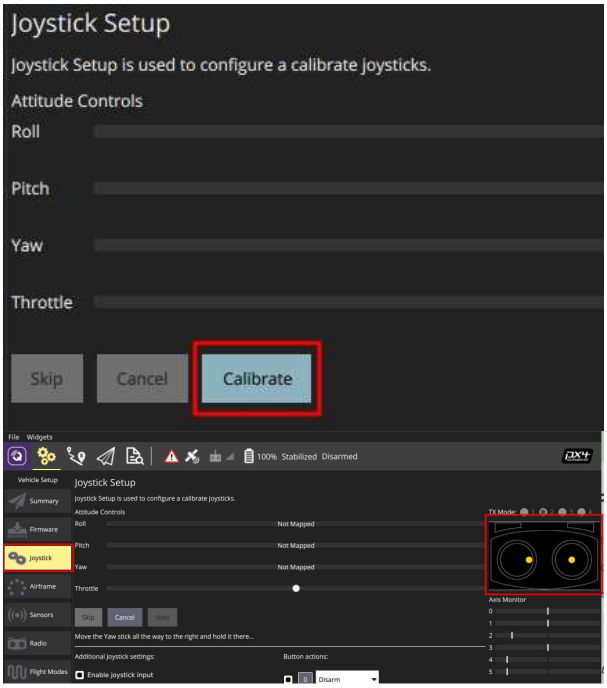
To do this, you can use the command `ip addr`. The result will be a numbered list of the active networks enabled on your device. Note the connection with the prefix *tun* and the IP address you specify; if it is present in your list, your copter is connected to the network.

Set up the GCS connection to your drone using the same protocol that is used for your VPN network. The steps are the same as in the Wi-Fi connection article. We recommend using *UDP* due to lower latency.

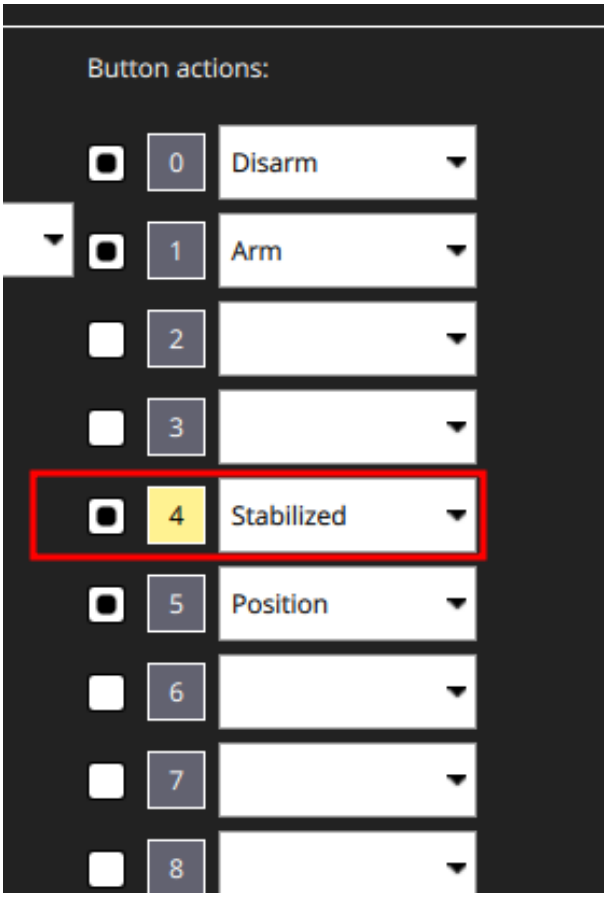
If you have a connection to your drone, connect some joystick to your PC. You may use an RC transmitter with a USB port, such as FlySky-i6X, Taranis x7, etc., as well as any analog joystick that is recognized by the system.

When the joystick is recognized by the system, the *Joystick* item will appear in the *Vehicle Setup* column. If it is highlighted in red, then calibration is required.

To calibrate the joystick, press the *Calibrate* button in the *Joystick* tab and follow the instructions for the sticks position on the left side of the window.



After successful calibration, flight modes must be set up. To do this, switch the required toggle switches several times. During switching, you will see the virtual channels on which the toggle switches operate. One of the channels will be highlighted in the active position.

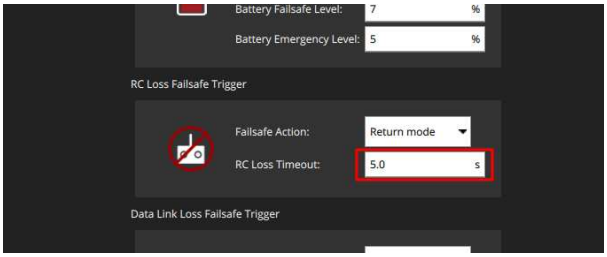


When selecting the joystick, check the number of working channels and its support in QGroundControl (which uses SDL2, so any joystick supported there should be fine). There are joysticks that support only 4 channels, which are not convenient for this type of control.

If changes to stick positions are reflected in the QGroundControl window, all you have to do is apply a parameter that specifies that the drone is controlled by the joystick, not by the RC:

`COM_RC_IN_MODE` - Joystick/No RC Checks

Since mobile communication is not always stable, it is recommended to increase the timeout for control signal loss to 5 seconds.



The drone is ready to fly!

If the copter does not arm when you move the left stick to the bottom right corner, set the Arm/Disarm command to one of the switches.

Streaming video from the camera to QGroundControl

You can stream video from almost any camera connected to your Raspberry Pi. You will need to install or build the *gst-rtsp-launch* package:

```
sudo apt update
sudo apt install gst-rtsp-launch
```

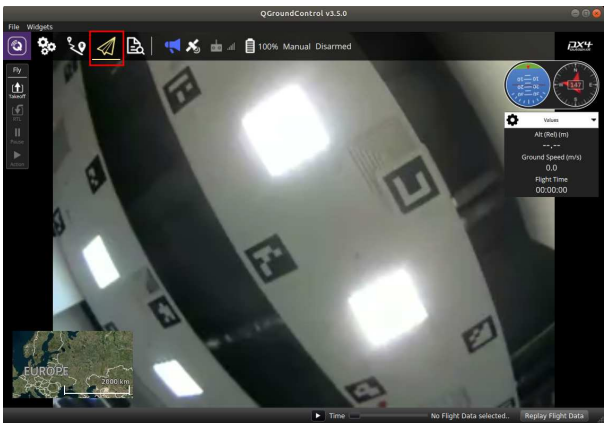
To start the transfer of images, you must enter the appropriate command line:

```
gst-rtsp-launch "( v4l2src device=/dev/video0 ! video/x-raw,framerate=30/1,widt
```

This command line contains the parameters of the video stream, such as the source video device, framerate, image height/width, encoding, etc. You can see more examples in the `gst-rtsp-launch` repository.

> A Raspberry Pi camera device `/dev/video0` could be used by `clover` systemd service. In this case `gst-rtsp-launch` will not get an access to the device. For stop the `clover` run command `sudo systemctl stop clover`. Also you could stream from a USB-camera, for this change the source video device to `/dev/video1`.

Make sure the stream by the address `rtsp://192.168.11.1:8554/video` (the IP-address of your Raspberry Pi could be different) is received and shown in QGroundControl.



Starting video stream automatically

Create a file and add your video stream command line:

```
nano script_name.sh
```

In order to run the file, you have to mark it as executable.

```
chmod a+x script_name.sh
```

You can use systemd to launch this script every time on system startup. Create the `qgc_video.service` file in the `/etc/systemd/system` directory:

```
sudo nano /etc/systemd/system/qgc_video.service
```

Put the following in this file:

```
[Unit]
Description=VideoStream

[Service]
ExecStart=/bin/bash /home/pi/script_name.sh

[Install]
WantedBy=multi-user.target
```

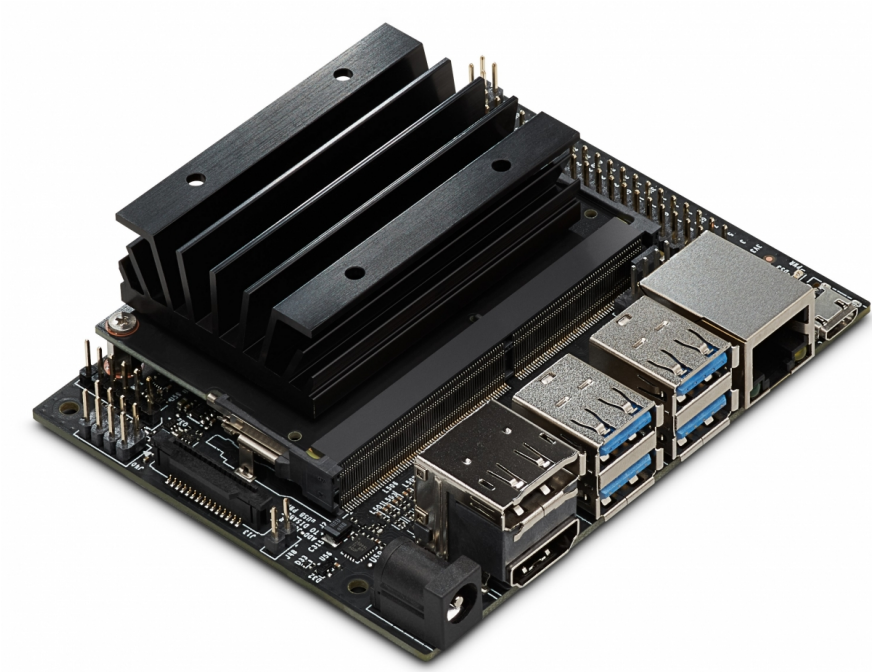
Enable the script at startup.

```
sudo systemctl enable qgc_video.service
```

Clover and Jetson Nano

Jetson Nano overview

Jetson Nano is a system-on-a-module by Nvidia. It is built on a Tegra X1 platform. With four ARM Cortex-A57 cores clocked at 1.4 GHz, 4 GB of RAM and a relatively powerful GPU, it is more capable than a Raspberry Pi 3 series of single-board computers.



Jetson Nano developer kits come with a carrier board that has USB 3.0, CSI and Ethernet ports, as well as a row of GPIO pins. The carrier board is only slightly larger than a Raspberry Pi computer, making it a viable option for an onboard computer.

> The default carrier board does not have a Wi-Fi chip installed. You can use a USB Wi-Fi adapter or install a Wi-Fi card in the M.2 slot on the carrier board. Be sure to check your adapter for compatibility with the Jetson Nano!

Setting up

Nvidia provides an SD card image with an operating system based on Ubuntu Linux 18.04 for Jetson Nano. This image is a good starting point for ROS and Clover installation.

Initial system setup

> Be sure to check the official Getting Started instructions for the Jetson Nano developer kit!

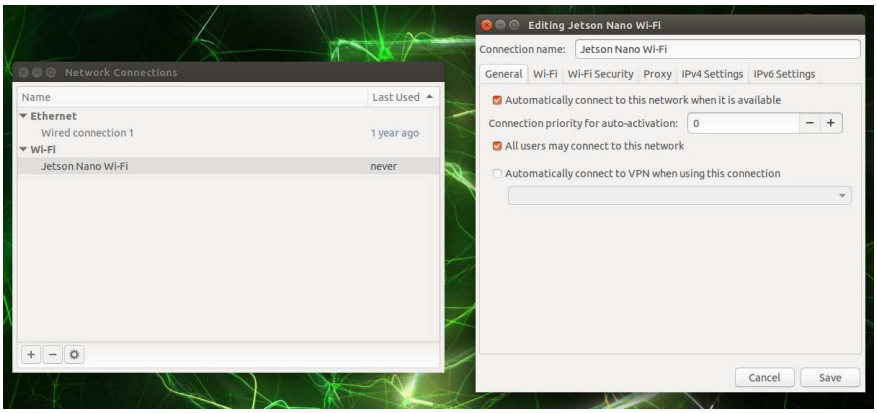
For the initial setup you'll need an HDMI or DisplayPort monitor, a keyboard and a mouse. Download the Jetson Nano developer kit image and flash it on a microSD card (a 32+ GB card is strongly recommended). Plug the card into the Jetson Nano module, connect your monitor, keyboard, and mouse to the carrier board, and power up the Jetson Nano.

> Jetson Nano can be powered by a microUSB cable, but we strongly suggest using a good power supply and a barrel jack connector. You'll need to put a jumper on the J48 pins (they are right next to the CSI connector on the carrier board).

Accept the Nvidia EULA and follow the installer prompts. The system will reboot after installation. Login with your username and password.

> We strongly recommend to choose the English system language/locale for Jetson Nano to avoid ROS compatibility issues!

If you've installed a Wi-Fi adapter, you may want to configure your Jetson Nano to connect to your Wi-Fi network automatically. Once the system is installed and booted up, click on the "wireless network" icon in the top bar, choose "Edit Connections..." in the drop-down menu, select your network name from the list and click on the gear icon at the bottom of the window.



Go to the "General" tab in the newly-opened window and check the "All users may connect to this network" checkbox. Press the "Save" button to close the window.

> You may want to make sure you're able to access your Jetson Nano over the network. The image already has SSH enabled, and it's more convenient to perform next steps using the remote shell.

Installing ROS

> Ubuntu 18.04 is officially supported as a base system for ROS Melodic. Be sure to check the official installation instructions!

Add OSRF keys and repositories to your system:

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E3
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) ma
sudo apt update
```

Install base ROS packages:

```
sudo apt install ros-melodic-ros-base
```

Enable your ROS environment and update your `rosdep` cache:

```
source /opt/ros/melodic/setup.bash
sudo rosdep init
rosdep update
```

> You may wish to put the `source /opt/ros/melodic/setup.bash` line at the end of your user's `.profile` file.

Install pip for Python 2 (while this is not technically a part of ROS, some dependencies are only installable using pip):

```
sudo apt install curl
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
sudo python ./get-pip.py
```

Building Clover nodes

Create a "workspace" directory in your home folder and populate it with Clover packages:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
git clone https://github.com/CopterExpress/clover
git clone https://github.com/CopterExpress/ros_led
git clone https://github.com/okalachev/vl53l1x_ros
```

Install dependencies using `rosdep`:

```
cd ~/catkin_ws
rosdep install --from-paths src --ignore-src -y
```

Install geographiclib datasets (they are required for mavros, but are not packaged with it):

```
curl https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts/ins
chmod a+x ./install_geographiclib_datasets.sh
sudo ./install_geographiclib_datasets.sh
```

Install development libraries for OpenCV 3.2 (recent Jetson Nano images have OpenCV 4.1.1 preinstalled; using this version will result in build failures):

```
sudo apt install libopencv-dev=3.2.0+dfsg-4ubuntu0.1
```

Finally, build the Clover nodes:

```
cd ~/catkin_ws
catkin_make
```

> You may also want to add udev rules for PX4 flight controllers. Copy the rules file to `/etc/udev/rules.d` and run `sudo udevadm control --reload-rules && sudo udevadm trigger`.

Running Clover nodes

Set up the workspace environment:

```
cd ~/catkin_ws
source devel/setup.bash
```

Configure the launch files to your taste and use `roslaunch` to launch the nodes:

```
roslaunch clover clover.launch
```

> You may want to start the Clover nodes automatically. This can be done with `systemd`: look at service files for `roscore` and `clover` that are used in our image and adjust them as necessary.

Caveats

CSI cameras

Jetson Nano currently does not support older Raspberry Pi v1 cameras (that are based on the Omnivision OV5647 sensor). Raspberry Pi v2 cameras (the ones that use Sony IMX219) are supported, but are not available as Video4Linux devices.

Fortunately, these cameras are available using GStreamer. You can try using the `gscam` ROS package or our `jetson_camera` node. The latter requires you to build OpenCV 3.4 from source with GStreamer support.

The GStreamer pipelines are available at JetsonHacksNano CSI camera repository.

You may also notice that the camera image has a red tint that is more pronounced near the edges. This can be fixed by image signal processor tuning. Generally this should be done by your camera manufacturer; here is a sample ISP

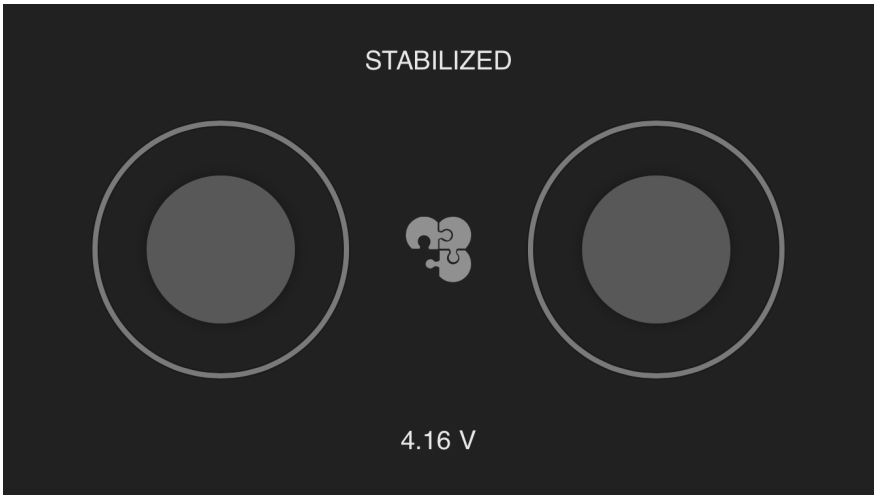
configuration from Adrucam

LED strip

Jetson Nano currently does not support LED strips over GPIO.

Controlling Clover from a smartphone

To control Clover from a smartphone via Wi-Fi, you have to install the appropriate application – iOS, Android (https://play.google.com/store/apps/details?id=express.copter.cleverrc).



> The mobile transmitter is mainly intended for indoor flights to the range not exceeding 10-15 m. Many Wi-Fi networks may also impair responsiveness and the range of the transmitter.

Control from a smartphone is also available in the mobile version of the app QGroundControl.

Configuring

> An open QGroundControl or rviz connection sends large amounts of data over Wi-Fi, which can adversely affect responsiveness of the mobile transmitter. It is recommended not to use these applications together with it.

Install Clover image on RPi. For running the application, settings `rosbridge` and `rc` in the launch file (`~/catkin_ws/src/clover/clover/launch/clover.launch`) should be enabled:

```
<arg name="rosbridge" default="true"/>
```

```
<arg name="rc" default="true"/>
```

After the launch-file is edited, restart package `clover` :

```
sudo systemctl restart clover
```

Also make sure that PX4-parameter `COM_RC_IN_MODE` is set to `0` (RC Transmitter).

Additional PX4 parameters:

- `COM_RC_LOSS_T` – timeout for detecting signal loss by the transmitter (mobile or physical). It is recommended to increase the timeout to several seconds.
- `NAV_RCL_ACT` – action upon loss of transmitter signal.

> The mobile transmitter conflicts with the real radio control equipment. When the mobile transmitter is used, it should be powered off.

Connection

Connect the smartphone to Clover Wi-Fi network (`clover-xxxx`). The application should connect to the copter automatically. Upon successful connection, the current mode and the battery charge level should be displayed.

The sticks on the screen of the application work just like real sticks. To arm the copter, hold the left stick in the bottom right corner for several seconds. To disarm — in the bottom left corner.

Malfunctions

- If the interface of the transmitter displays a surely incorrect voltage (e.g., > 5 V), check that the value of PX4 parameter `BAT_N_CELLS` matches the actual number of battery cells. If the displayed voltage is still incorrect, calibrate the power sensor.
- If instead of mode PX4, text "DISCONNECTED FROM FCU" is displayed, check Raspberry Pi connection to Pixhawk.

Configuring Wi-Fi

The Raspberry Pi Wi-Fi adapter has two main operating modes:

1. **Client mode** – RPi connects to an existing Wi-Fi network.
2. **Access point mode** – RPi creates a Wi-Fi network that you can connect to.

On our RPi image the Wi-Fi adapter is configured to use the access point mode by default.

Changing the password or SSID (of the network name)

1. Edit the `/etc/wpa_supplicant/wpa_supplicant.conf` file (using SSH connection):

   ```
   sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
   ```

   In order to change the name of the Wi-Fi network, change the value of the `ssid` parameter; to change the password, change the `psk` parameter. For example:

   ```
   network={
       ssid="my-super-ssid"
       psk="cloverwifi123"
       mode=2
       proto=RSN
       key_mgmt=WPA-PSK
       pairwise=CCMP
       group=CCMP
       auth_alg=OPEN
   }
   ```

2. Restart Raspberry Pi.

> The Wi-Fi network password should be **at least** 8 characters.
>
> If your `wpa_supplicant.conf` is not valid, Raspberry Pi will not allow Wi-Fi connections!

Switching adapter to the client mode

1. Disable the `dnsmasq` service.

   ```
   sudo systemctl stop dnsmasq
   sudo systemctl disable dnsmasq
   ```

2. Enable DHCP client on the wireless interface to obtain IP address. In order to do this, remove the following lines from the `etc/dhcpcd.conf` file:

```
interface wlan0
static ip_address=192.168.11.1/24
```

3. Configure `wpa_supplicant` to connect to an existing access point. Change your `/etc/wpa_supplicant/wpa_supplicant.conf` to contain the following:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB

network={
    ssid="SSID"
    psk="password"
}
```

   where `SSID` is the name of the network, and `password` is its password.

4. Restart the `dhcpcd` service.

```
sudo systemctl restart dhcpcd
```

Switching the adapter to the access point mode

1. Enable the static IP address in the wireless interface. Add the following lines to your `/etc/dhcpcd.conf` file:

```
interface wlan0
static ip_address=192.168.11.1/24
```

2. Configure wpa_supplicant to work in the access point mode. Change your `/etc/wpa_supplicant/wpa_supplicant.conf` file to contain the following:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB

network={
    ssid="clover-1234"
    psk="cloverwifi"
    mode=2
    proto=RSN
    key_mgmt=WPA-PSK
    pairwise=CCMP
    group=CCMP
    auth_alg=OPEN
}
```

   where `clover-1234` is the network name and `cloverwifi` is the password.

3. Enable the `dnsmasq` service.

```
sudo systemctl enable dnsmasq
sudo systemctl start dnsmasq
```

4. Restart the `dhcpcd` service.

```
sudo systemctl restart dhcpcd
```

Below you can read more about how RPi networking is organized.

RPi network organization

Network operation in the image is supported by two pre-installed services:

- **networking** — the service enables all network interfaces at startup [5].
- **dhcpcd** — the service ensures that configuration of addressing and routing on the interfaces is obtained dynamically or specified statically in the config file.

To work in the router (access point) mode, RPi requires a DHCP server. It is used to automatically send the settings of the current network to connected clients. `isc-dhcp-server` or `dnsmasq` may be used for this.

dhcpcd

Starting with Raspbian Jessie, network settings are no longer defined in the `/etc/network/interfaces` file. Now `dhcpcd` is used for sending addressing and routing settings[4].

By default, a dhcp client is enabled in all interfaces. Settings for network interfaces are changed in the `/etc/dhcpcd.conf` file. An access point should have a static IP address. To specify one, add the following lines to the end of the file:

```
interface wlan0
static ip_address=192.168.11.1/24
```

> If the interface is wireless (wlan), the `dhcpcd` service triggers `wpa_supplicant` [13], which in turn works directly with the Wi-Fi adapter, and sets it to the specified state.

wpa_supplicant

wpa_supplicant — the service configures the Wi-Fi adapter. The `wpa_supplicant` service does not run as a standalone service (although it exists as such), but is instead launched as a `dhcpcd` child process.

By default the config file is `/etc/wpa_supplicant/wpa_supplicant.conf`. An example of the configuration file:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB

network={
        ssid=\"my-clover\"
        psk=\"cloverwifi\"
        mode=2
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        group=CCMP
        auth_alg=OPEN
}
```

Inside the config file, general `wpa_supplicant` settings, and the settings for the adapter configuration are specified. The configuration file also contains `network` section with the basic settings of the Wi-Fi network, such as network SSID, password, adapter operating mode. There may be several `network` sections, but only the first valid one is used. For example, if the first section contains a connection to an unavailable network, the adapter will be configured according to a next valid section, if there is one. Read more about the syntax of `wpa_supplicant.conf` in the Debian manual page.

wpa_passphrase

`wpa_passphrase` — a utility for creating the `network` section.

```
wpa_passphrase SSID PASSWORD
```

After running the command, copy the resulting section to your config file. You may remove the commented field `psk`, and leave only the field with the password hash, or vice versa.

```
network={
    ssid="SSID"
    #psk="PASSWORD"
    psk=c2161655c6ba444d8df94cbbf4e9c5c4c61fc37702b9c66ed37aee1545a5a333
}
```

Multiple Wi-Fi adapters

The system may use multiple Wi-Fi adapters. If drivers are properly connected to them, they may be viewed by calling `ifconfig` (e.g. `wlan0` and `wlan1`).

If you have multiple adapters, the same working `network` section will be used for all of them. This is due to the fact that for each interface, `dhcpcd` separately creates a child `wpa_supplicant` process, which runs the same code (since the config is the same).

To make multiple adapters work with individual settings, the mechanism for running different configuration scripts is implemented in the called standard `dhcpcd` script. To use it, rename the standard config file as follows:

`wpa_supplicant-<interface name>.conf`, for example `wpa_supplicant-wlan0.conf`.

To apply the settings, restart the parent process — the `dhcpcd` service. This can be done by running the following command:

```
sudo systemctl restart dhcpcd
```

DHCP server

dnsmasq-base

`dnsmasq-base` — a command-line utility, which is not a service. To use dnsmasq as a service, install the `dnsmasq` package.

```
sudo apt install dnsmasq-base
```

```
# Calling dnsmasq-base
sudo dnsmasq --interface=wlan0 --address=/clover/coex/192.168.11.1 --no-daemon

# More about dnsmasq-base
dnsmasq --help

# or
man dnsmasq
```

dnsmasq

```
sudo apt install dnsmasq
```

```
cat << EOF | sudo tee -a /etc/dnsmasq.conf
interface=wlan0
address=/clover/coex/192.168.11.1
dhcp-range=192.168.11.100,192.168.11.200,12h
no-hosts
filterwin2k
bogus-priv
domain-needed
quiet-dhcp6

EOF
```

isc-dhcp-server

```
sudo apt install isc-dhcp-server
```

```
# https://www.shellhacks.com/ru/sed-find-replace-string-in-file/
sed -i 's/INTERFACESv4=\"\"/INTERFACESv4=\"wlan0\"/' /etc/default/isc-dhcp-serv
```

```
cat << EOF | sudo tee /etc/dhcp/dhcpd.conf
subnet 192.168.11.0 netmask 255.255.255.0 {
  range 192.168.11.11 192.168.11.254;
  #option domain-name-servers 8.8.8.8;
  #option domain-name "rpi.local";
  option routers 192.168.11.1;
  option broadcast-address 192.168.11.255;
  default-lease-time 600;
  max-lease-time 7200;
}

EOF
```

```
cat << EOF | sudo tee /etc/network/if-up.d/isc-dhcp-server && sudo chmod +x /et
#!/bin/sh
if [ "\$IFACE" = "--all" ];
then sleep 10 && systemctl start isc-dhcp-server.service &
fi

EOF
```

Links

1. habr.com Linux WiFi from the command line with wpa_supplicant
2. wiki.archlinux.org WPA supplicant (Russian)
3. blog.hoxnox.com: WiFi access point with wpa_supplicant
4. dmitrysnotes.ru: Raspberry Pi 3. Assigning a static IP addresses
5. thegeekdiary.com: Linux OS Service ‘network’
6. frillip.com: Using your new Raspberry Pi 3 as a Wi-Fi access point with hostapt (it also contains instructions for setting up forwarding for using RPi as an Internet gateway)
7. habr.com: Configuring a ddns server on a GNU/Linux Debian 6 (Good article on configuring a ddns server based on `bind` and `isc-dhcp-server`)
8. pro-gram.ru to: Setting up and configuring a DHCP server in Ubuntu 16.04. (setup isc-dhcp-server)
9. expert-orda.ru: Configuring a DHCP server in Ubuntu (setup isc-dhcp-server)
10. academicfox.com: A Raspberry Pi wireless access point (WiFi access point) (setting the routes, hostapd, isc-dhcp-server)
11. weworkweplay.com: Automatically connect a Raspberry Pi to a Wifi network (Contains settings for creating an open access point)
12. wiki.archlinux.org: WPA supplicant
13. wiki.archlinux.org: dhcpcd (dhcpcd hook wpa_supplicant)

UART interface

UART is an asynchronous serial interface for data transfer that is used in many devices. For example, GPS antennas, Wi-Fi routers, or Pixhawk.

The interface usually contains two lines: TX for data transmission, and RX for data reception. It usually uses the 5-volt logics.

To connect two devices, you have to feed the TX line of the first device to the RX line of the other one. A similar manipulation is required on the other end for ensuring two-way data transmission.

It is also necessary to synchronize the voltages – connect the ground on two devices.

Read more about the interface and the Protocol in this article.

Linux TTY

In Linux, there is the concept of POSIX Terminal Interface (read more here. It is an abstraction over the serial or virtual interface that allows several agents to work with the device simultaneously.

An example of such abstraction in Raspbian may be `/dev/tty1` – the device for text output to the screen connected via HDMI.

UART on Raspberry Pi 3

Raspberry Pi 3 has two hardware UART interfaces:

1. Mini UART (`/dev/ttyAMA0`) – uses the timing of the RPi graphics core, and therefore limits its frequency.
2. PL011 (`/dev/ttyS0`) – the full-fledged UART interface on a separate chip of the microcontroller.

Read more about UART on Raspberry Pi in the official article.

Using microcontroller valves, these interfaces may be switched between two physical outputs:

1. UART connector on GPIO;
2. RPi Bluetooth module.

By default, Raspberry Pi 3 PL011 is connected to the Bluetooth module. And Mini UART is disabled with the value of directive `enable_uart` , which is `0` by default.

> One should understand that directive `enable_uart` changes its default value, depending on which UART is connected to the RPi Bluetooth module with directive `dtoverlay=pi3-miniuart-bt` .

For the sake of convenience of working with these outputs, aliases exist in Raspbian:

- `/dev/serial0` – always points to the TTY device that is connected to the GPIO ports.
- `/dev/serial1` – always points to the TTY device that is connected to the Bluetooth module.

Configuration of UART on Raspberry Pi 3

To configure UART, there are directives located in `/boot/config.txt`.

To enable the UART interface on GPIO:

```
enable_uart=1
```

To disconnect the UART interface from the Bluetooth module:

```
dtoverlay=pi3-disable-bt
```

To connect Mini UART to the Bluetooth module:

```
dtoverlay=pi3-miniuart-bt
```

If the Bluetooth module is disabled, one should disable the hciuart service:

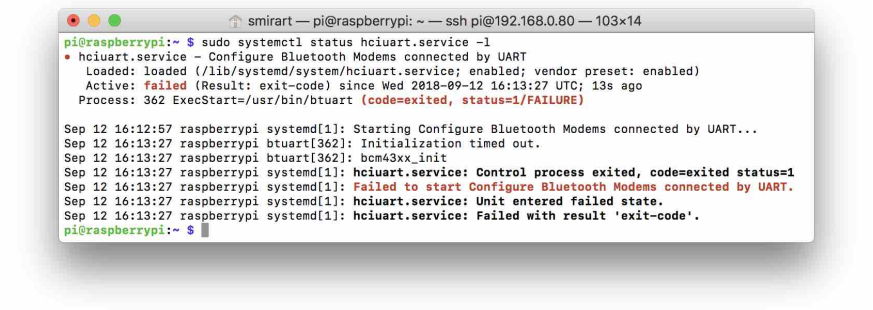
```
sudo systemctl disable hciuart.service
```

Default image configuration

In the Clover image for RPi, we initially disabled Mini UART and the Bluetooth module.

Bugs

If you use the Mini UART connection to Bluetooth, `hciuart` crashes with the following error:



In case of Bluetooth disconnection

```
/dev/serial0 -\> ttyAMA0
/dev/serial1 -\> ttyS0
```

PX4 Parameters

Full documentation on PX4 parameters:
https://docs.px4.io/master/en/advanced_config/parameter_reference.html.

For changing PX4 parameters, use QGroundControl software, connect to Clover over Wi-Fi or USB. Go to *Vehicle Setup* panel (click on the QGroundControl logo in the top-left corner) and choose *Parameters* menu.

Recommended values

Common parameters

Parameter	Value	Comment
`SENS_FLOW_ROT`	0 (*No rotation*)	If using *PX4Flow* hardware, keep the default value
`SENS_FLOW_MINHGT`	0.0	For VL53L1X rangefinder
`SENS_FLOW_MAXHGT`	4.0	For VL53L1X rangefinder
`SENS_FLOW_MAXR`	10.0	
`SYS_HAS_MAG`	0	If impossible to run the magnetometer (*No mags found* error)

Estimator subsystem parameters

In case of using LPE (COEX patched firmware):

Parameter	Value	Comment
LPE_FUSION	86	Checkboxes: *flow* + *vis* + *land detector* + *gyro comp*. If flying over horizontal floor *pub agl as lpos down* checkbox is allowed. Details: Optical Flow, ArUco markers, GPS.
LPE_VIS_DELAY	0.0	
LPE_VIS_Z	0.1	
LPE_FLW_SCALE	1.0	
LPE_FLW_R	0.2	
LPE_FLW_RR	0.0	
LPE_FLW_QMIN	10	
ATT_W_EXT_HDG	0.5	Enabling usage of external yaw angle (when navigating using markers map)
ATT_EXT_HDG_M	1 (*Vision*)	
ATT_W_MAG	0	Disabling usage of the magnetometer (when navigating indoor)

In case of using EKF2 (official firmware):

Parameter	Value	Comment
EKF2_AID_MASK *	26	Checkboxes: *flow* + *vision position* + *vision yaw*. Details: Optical Flow, ArUco markers, GPS.
EKF2_OF_DELAY	0	
EKF2_OF_QMIN	10	
EKF2_OF_N_MIN	0.05	
EKF2_OF_N_MAX	0.2	
EKF2_HGT_MODE *	3 (*Vision*)	If the rangefinder is present and flying over horizontal floor – 2 (*Range sensor*)
EKF2_EVA_NOISE	0.1 rad or 5 deg	
EKF2_EVP_NOISE	0.1	
EKF2_EV_DELAY	0	
EKF2_MAG_TYPE	5 (*None*)	Disabling usage of the magnetometer (when navigating indoor)

* — starting from PX4 version 1.14, the parameters marked with an asterisk are replaced with the following:

Parameter	Value	Comment
`EKF2_EV_CTRL`	11	Checkboxes: *Horizontal position* + *Vertical position* + *Yaw*
`EKF2_GPS_CTRL`	0	All checkboxes are disabled
`EKF2_BARO_CTRL`	0 (*Disabled*)	Barometer is disabled
`EKF2_OF_CTRL`	1 (*Enabled*)	Optical flow is enabled
`EKF2_HGT_REF`	3 (*Vision*)	If the rangefinder is present and flying over horizontal floor – 2 (*Range sensor*)
`EKF2_RNG_CTRL`	2 (*Enabled*)	Range sensor is enabled

See also: list of default parameters of the Clover simulator: https://github.com/CopterExpress/clover/blob/master/clover_simulation/airframes/4500_clover.

Additional information

The `SYS_MC_EST_GROUP` parameter defines the estimator subsystem to use.

Estimator subsystem is a group of modules that calculates the current state of the copter using readings from the sensors. The copter state includes:

- Angle rate of the copter – roll_rate, pitch_rate, yaw_rate;
- Copter orientation (in the local coordinate system) – roll, pitch, yaw (one of presentations);
- Copter position (in the local coordinate system) – x, y, z;
- Copter speed (in the local coordinate system) – vx, vy, vz;
- Global coordinates of the copter – latitude, longitude, altitude;
- Altitude above the surface;
- Other parameters (the drift of gyroscopes, wind speed, etc.).

`SYS_AUTOCONFIG` — resets all parameters (sets to 1).

EKF2

`EKF2_AID_MASK` — selects sensors that are used by EKF2 to calculate the copter state.

`EKF2_HGT_MODE` is the main source of height data (z in the local coordinate system):

- 0 – pressure reading on the barometer.
- 1 – GPS.
- 2 – distance meter (for example, vl53l1x).
- 3 – data from VPE.

Variant 2 is the most accurate; however, it is correct to use it only if the surface the copter flies over is flat. Otherwise, the Z axis origin will move up and down with the altitude of the surface.

Multicopter Position Control

These parameters adjust the flight of the copter by position (POSCTL, OFFBOARD, AUTO modes).

`MPC_THR_HOVER` — hovering throttle. This option is to set to the approximate percentage of throttle needed to make the copter maintain its altitude. If copter has a tendency to gain or lose altitude during the hovering mode, reduce or increase this value.

`MPC_XY_P` – position factor *P* of the ESC. This parameter affects how sharply the copter will react to the position commands. A too high value may cause overshoots.

`MPC_XY_VEL_P` – speed factor *P* of the ESC. This parameter also affects the accuracy and sharpness of copter execution of the given position. A too high value may cause overshoots.

`MPC_XY_VEL_MAX` — the maximum horizontal speed in POSCTL, OFFBOARD, AUTO modes.

`MPC_Z_P` , `MPC_Z_VEL_P` – vertical position and speed factors *P* of the ESCs they determine the copter's ability to maintain the desired altitude.

`MPC_LAND_SPEED` is the vertical velocity of landing in the LAND mode.

LPE + Q attitude estimator

These parameters configure the behavior of the `lpe` and `q` modules, which compute the state (orientation, position) of the copter. These parameters apply **only** if the `SYS_MC_EST_GROUP` parameter is set to `1` (local_position_estimator, attitude_estimator_q).

Commander

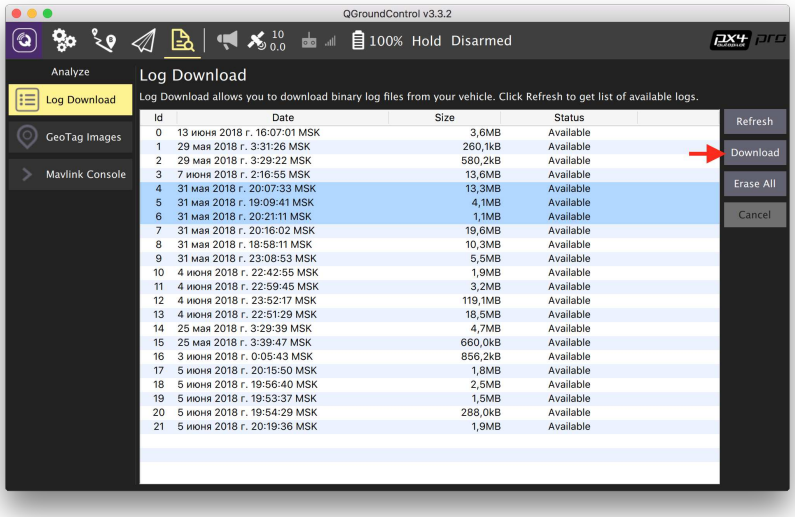
Prearm checks, switching the modes and states of the copter.

Sensors

Enabling, disabling and configuring various sensors.

PX4 Logs and Topics

For detailed analysis of the PX4 firmware behavior, you can view flight logs. Flight logs are messages in uORB topics written to a file with extension `.ulg`. The log file can be downloaded using QGroundControl via Wi-Fi or USB in the *Download Log* tab:

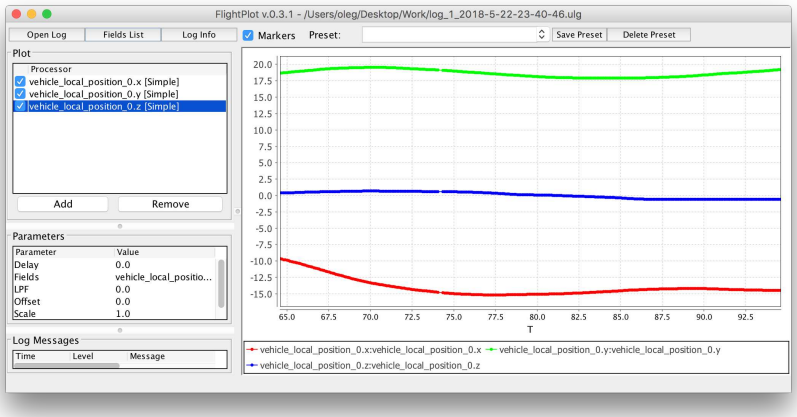


The required `.ulg` files may also be copied directly from the MicroSD card in the flight controller.

Analysis

The log file can be analyzed using the FlightPlot application. The current version of the application is available for downloading from GitHub.

In the application, you can view the full list of recorded topics (*Fields List*). In the list, you will have to select the required topics, after which they will appear on the chart:



Main topics in PX4

uORB is a pubsub mechanism similar to ROS topics, but greatly simplified and suitable for an embedded environment.

A complete list of topics may be found in the source code of the project in the `msg` directory.

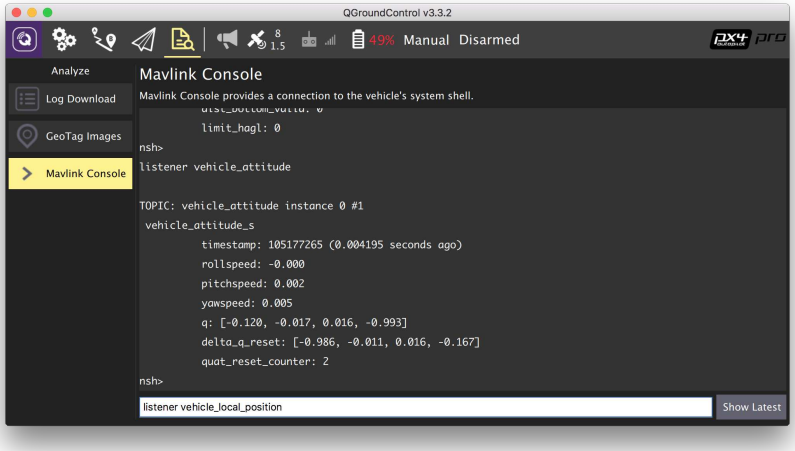
Here are some topics:

- **vehicle_status** — quadcopter status (mode, etc.).
- **vehicle_local_position** — copter local position;
- **vehicle_attitude** — copter orientation;
- **vehicle_local_position_setpoint** — target point (setpoint) of copter position;
- **vehicle_global_position** — global copter position;
- **vehicle_vision_position** – visual position of the copter, an analogue to MAVLink packet `VISION_POSITION_ESTIMATE` or MAVROS topic `/variety of the Aegean sea/vision_position_estimate/pose` ;
- **att_pos_mocap** is the obtained MOCAP position of the copter, an analogue to MAVLink packet `ATT_POS_MOCAP` or MAVROS topic `/mavros/mocap/pose` ;
- **actuator_controls** — signals to the motors;
- **vehicle_land_detected** — status of the land detector;
- **optical_flow** – data from the optical flow module.

Monitoring the topics in real time

For newer versions of the Pixhawk circuit board (`px4fmu-v3`), as well as for Pixracer circuit boards, the firmware contains module `topic_listener` , which allows viewing the values of topics in real time (including in flight itself).

To use it, select tab MAVLink Console in QGroundControl:



Command `list_topics` displays a list of topics available for viewing (included only in SITL).

Command `listener <topic name>` displays the current value in the topic. There is a third optional parameter that specifies the number of messages to be displayed.

Examples of commands:

```
listener vehicle_local_position
```

```
listener vehicle_attitude 5
```

Pixhawk / Pixracer firmware flashing

Pixhawk, Pixracer or COEX Pix firmware may be flashed using QGroundControl or command line utilities.

Modified firmware for Clover

It is advisable to use a specialized build of PX4 with the necessary fixes and better defaults for the Clover drone. Use the latest stable release in our GitHub repository with the word `clover`, for example, `v1.8.2-clover.5`.

> If you are using the firmware version older than *v1.10* (for example, `v1.8.2-clover.13`), then in order to avoid configuration errors, use QGroundControl version *v4.2.0* (or older). See detailed information about changes in the firmware parameters that cause errors in newer versions of QGroundControl.

QGroundControl

Open the Firmware section in QGroundControl. Then, connect your flight controller via USB.

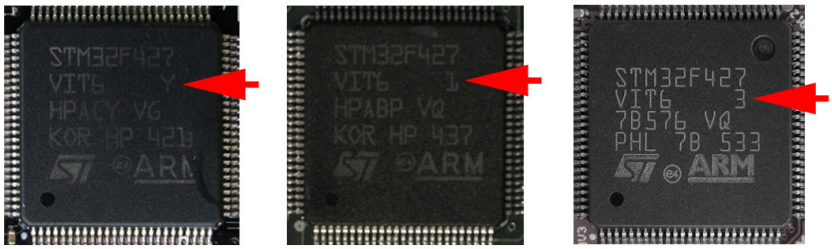
Choose PX4 Flight Stack. If you wish to install the official firmware (with EKF2 for Pixhawk), choose "Standard version". In order to flash custom firmware, choose "Custom firmware file..." and click OK.

> Do not unplug your flight controller from USB during flashing!

Firmware variants

The name of the firmware file contains information about the target flight controller and build variant. For example:

- `px4fmu-v4_default.px4` — firmware for COEX Pix and Pixracer with EKF2 and LPE (**Clover 3** / **Clover 4**).
- `px4fmu-v2_lpe.px4` — firmware for Pixhawk with LPE (**Clover 2**).
- `px4fmu-v2_default.px4` — firmware for Pixhawk with EKF2.
- `px4fmu-v3_default.px4` — firmware for newer Pixhawk versions (rev. 3 chip, see Fig. + Bootloader v5) with EKF2 and LPE.



> In order to flash the `px4fmu-v3_default.px4` file, you may need to use the `force_upload` command in the command prompt.

Command prompt

PX4 may be compiled from the source and automatically flashed to the flight controller from the command prompt.

To do this, clone the PX4 repository:

```
git clone https://github.com/PX4/PX4-Autopilot.git
```

Select the appropriate version (tag) using `git checkout`. Then compile and upload the firmware:

```
make px4_fmu-v4_default upload
```

Where `px4_fmu-v4_default` is the required firmware variant.

In order to upload the `v3` firmware to Pixhawk, you may need to use the `force_upload` option:

```
make px4_fmu-v3_default force-upload
```

MAVLink

Basic documentation: https://mavlink.io/en/.

MAVLink is a communication protocol between autonomous aircraft and vehicle systems (drones, planes, vehicles). The MAVLink protocol lies at the base of interaction between Pixhawk and Raspberry Pi.

Clover contains two wrappers for this protocol: MAVROS and simple_offboard.

The code for sending an arbitrary MAVLink message may be found in the examples.

The main concerts

Communication channel

The MAVLink protocol may be used on top of the following communication channels:

- connection in series (UART, USB, etc.);
- UDP (Wi-Fi, Ethernet, 3G, LTE);
- TCP (Wi-Fi, Ethernet, 3G, LTE).

Message

A MAVLink message is an individual "portion" of data transmitted between devices. An individual MAVLink message contains information about the state of the drone (or another device) or a command for the drone.

Examples of MAVLink messages:

- `ATTITUDE`, `ATTITUDE_QUATERNION` – the quadcopter orientation in the space;
- `LOCAL_POSITION_NED` – local position of the quadcopter;
- `GLOBAL_POSITION_INT` – global position of the quadcopter (latitude/longitude/altitude);
- `COMMAND_LONG` – a command to the quadcopter (take off, land, toggle modes, etc).

A complete list of MAVLink messages is available in MAVLink documentation.

System, system component

Each device (a drone, a base station, etc.) has an ID in the MAVLink network. In PX4 MAVLink, ID is changed using parameter `MAV_SYS_ID`. Each MAVLink message contains a field with the ID of the originating system. Besides, some messages (for example, `COMMAND_LONG`) also contain the ID of the target system.

In addition to IDs of the systems, the messages may contain IDs of the originating component and the target component. Examples of the system components: a flight controller, an external camera, a controlling onboard computer (Raspberry Pi in case of Clover), etc.

An example of a package

An example of a MAVLink package structure with message `COMMAND_LONG`:

	Field	Length	Name	Comme
Header	`magic`	1 byte	Start tag	0xFD for MAVLink 2
	`len`	1 byte	Data size	
	`incompat_flags`	1 byte	Reversely incompatible flags	Currently unused
	`compat_flags`	1 byte	Reversely compatible flags	Currently unused
	`seq`	1 byte	Message sequence number	
	`sysid`	1 byte	Originating system ID	
	`compid`	1 byte	Originating component ID	
	`msgid`	3 bytes	Message ID	
Data (example)	`target_system`	1 byte	Target system ID	
	`target_component`	1 byte	Target component ID	
	`command`	2 bytes	Command ID	
	`confirmation`	1 byte	Number for confirmation	
	`param1`	4 bytes	Parameter 1	A single-precision floating po number
	`param2`	4 bytes	Parameter 2	
	`param3`	4 bytes	Parameter 3	
	`param4`	4 bytes	Parameter 4	
	`param5`	4 bytes	Parameter 5	
	`param6`	4 bytes	Parameter 6	
	`param7`	4 bytes	Parameter 7	
	`checksum`	2 bytes	Checksum	
	`signature`	13 bytes	Signature (optional)	Allows checking t the packa has not be compromi Usually unused.

Yellow is used for highlighting the data fields(payload). An individual set of such fields exists for every message type.

How to use a multimeter?

Check the circuits (continuity test)

> The tested object should be disconnected from the power supply (de-energized)!

Using a multimeter, check the absence of a short circuit (check the loop):

- Set the multimeter to the loop check mode.
- Test the multimeter by shorting the probes. A unit that operated properly makes a distinctive sound.
- The red probe is connected to the “+ ”pin, the black probe — to the “-” / ”GND” pin. If the circuit is short, a sound is heard.

 Loop check mode

1. Check OPEN CONDITION of the following circuits (absence of the multimeter sound signal):

- “BAT+” and “BAT-”
- “12V” and “GND”
- “5V” and “GND”

2. Check CLOSED CONDITION of the following circuits (presence of the multimeter sound signal):

- “BAT-” with every contact marked “-” and “GND”
- “BAT+”, with every contact marked “+”

Checking for voltage

Using a multimeter, you need to make sure that the voltage converters located on the power distribution board are working properly and provide the voltage of 5V and 12V, respectively.

- Switch the multimeter to the "Measuring DC voltage" mode
- Select the upper limit of the measured voltage (in our case, not more than 20)
- Make sure the battery is connected
- Make the following measurements:
    1. Measure the battery voltage (between BAT+ and BAT-). It should be between 14.0 V to 16.8 V
    2. Measure the voltage at the 5V output. It should not exceed 5.5 V
    3. Measure the voltage at the 12 V output. It should not exceed 12.5 V

After measurement:

- disconnect the battery
- turn off the multimeter

Possible radio failures

The remote is blocked

If the remote is blocked, the LCD will read: "Warning. Place all switches in their up position and lower the throttle".

To unlock the controller, place all sticks and switches to the initial position, namely:

1. The left stick (1) to the central bottom position.
2. Switches A, B, C, D (2) to the position “away from yourself”.
3. The right stick (3) to the center.



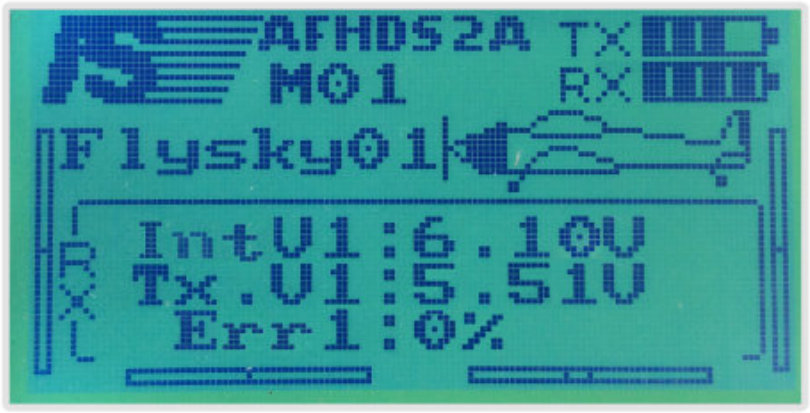
No communication with the receiver

To test the remote connection with the receiver, turn on the remote and watch the readouts on the LCD Screen.

1. Communication with the receiver is absent:



2. Communication with the receiver established:

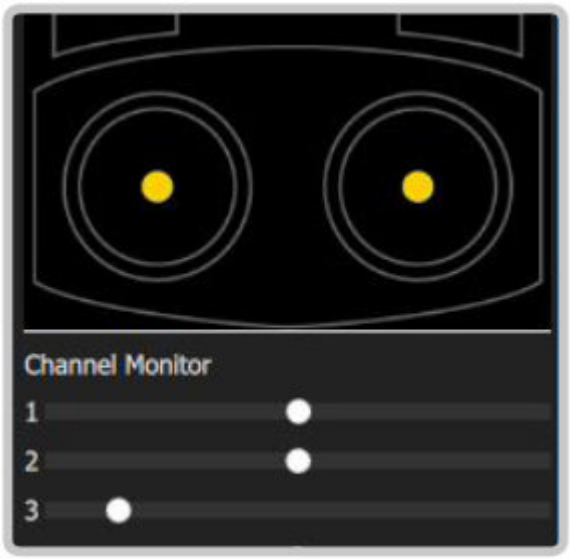


If there is no communication:

1. Make sure the receiver is enabled (the red LED is blinking). If the LED remains constantly ON, it means that communication is established with another remote.
2. Pair the remote and the receiver.

No communication with the flight controller

If there is no communication with the flight controller, the screen of the computer monitor in the Channel Monitor window will not display changes in the position of the sliders when the sticks of your remote are shifted.



1. Go to MENU (by holding down the “OK” button)
2. Select menu “System setup” (Up/Down Button to navigate, OK button - to confirms the choice).
3. Select “RX setup” > “PPM OUTPUT” > “On”
4. Save changes (hold pressed the “CANCEL” button).

Flashing ESCs using BLHeliSuite

A good article that explains the principle of ESCs (Electric speed controller) operation: http://www.avmodels.ru/engines/electric/esc.html

Why reflash?

Sometimes, it is necessary to change one of ESC parameters, such as the direction of motor rotation, the minimum and the maximum duty cycle of the PPM signal at ESC input, the volume of audio signals emitted by the motor, or the time after which the ESC should start reminding that it is engaged.

An application for flashing ESCs

For flashing various ESC, the BLHeliSuite application is used (for Windows).

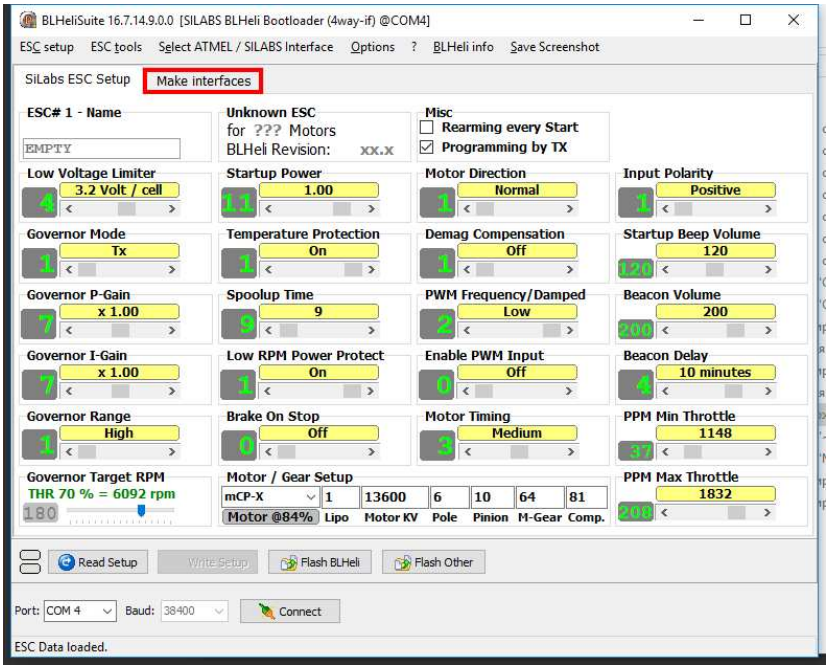
To start the (BLHeliSuite.exe) application, unpack archives BLHeliAtmelHEX.zip and BLHeliSilabsHEX.zip into the folder with the application.

A programming unit for flashing ESCs

To flash an ESC, you need a programming unit that can handle an ESC controller via the 1-wire protocol. One of the ways of obtaining the programming unit - is flashing special firmware to an Arduino device. BLHeliSuite contains a tool for creating interfaces for programming units.

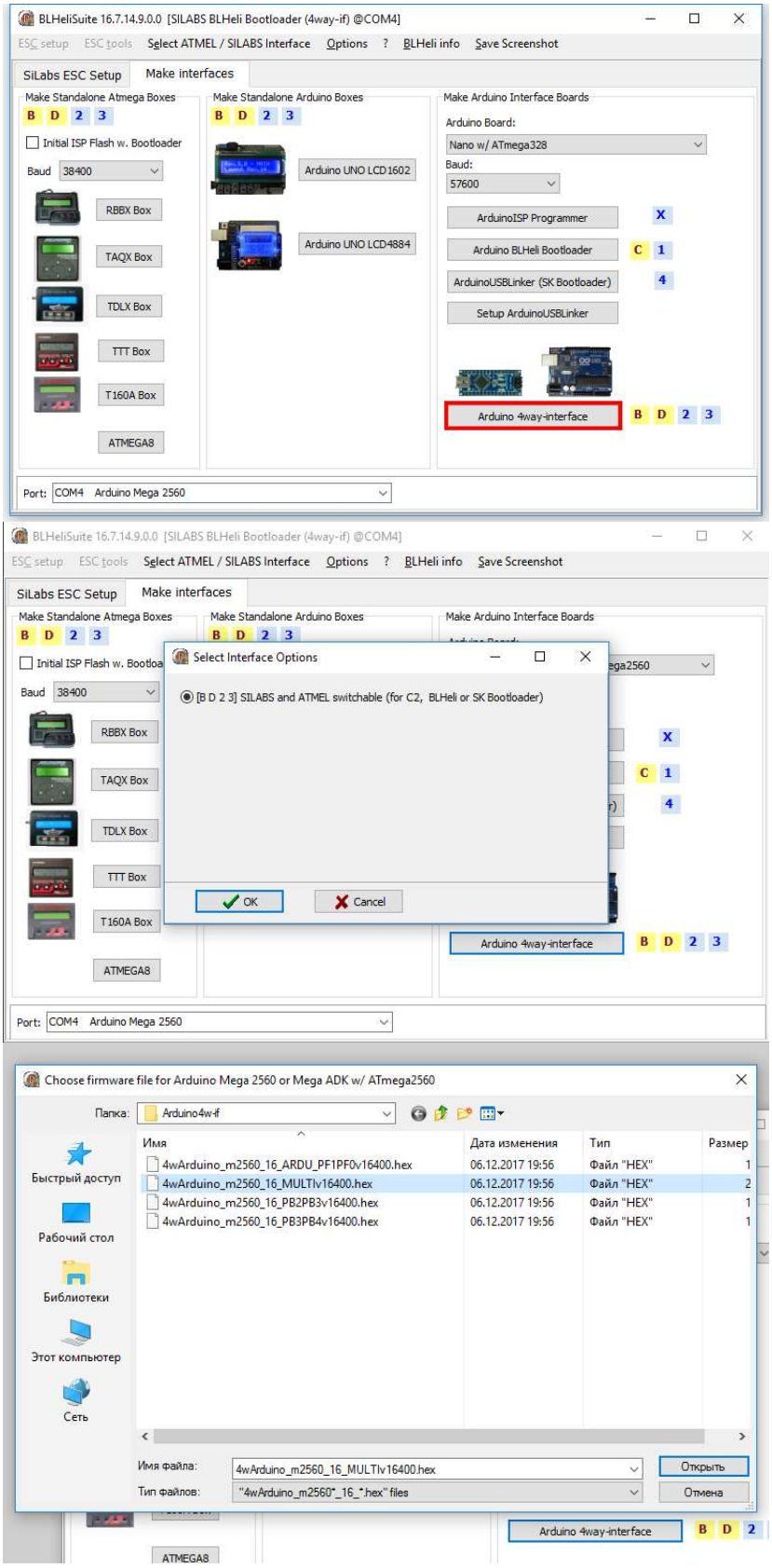
Creating a programming unit on the example of Arduino Mega.

1. Start BLHeliSuite and select the Make interfaces tab.

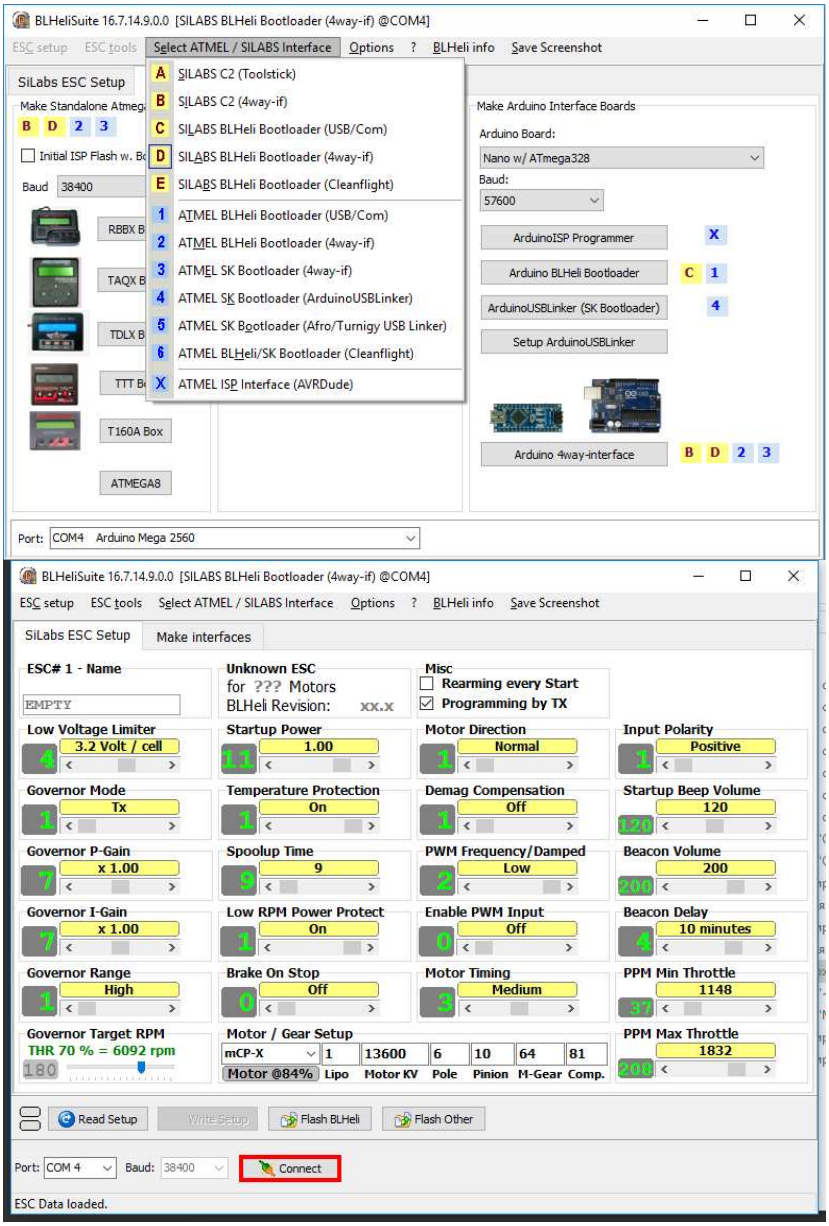


2. Connect the Arduino to a computer, if necessary, check the number of the COM port to which the circuit board is connected, in the Device Manager.

3. Click on Arduino 4way-interface in tab Make Arduino Interface Boards, and select the firmware file. After the file is selected, the flashing of the controller will start.

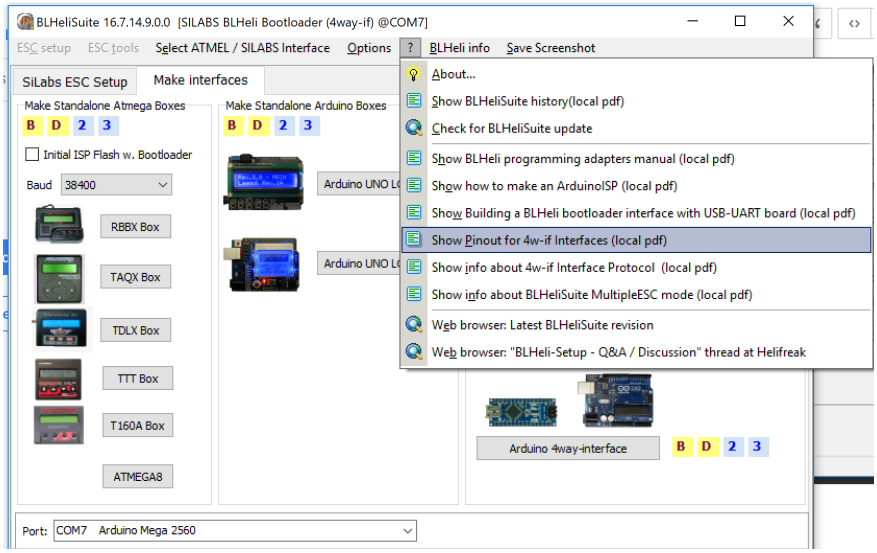


4. After flashing the Arduino, return to tap Silabs ESC Setup and connect to Arduino, having selected the 4way-if interface of the programming device and the Arduino COM port.



Connecting ESCs to Arduino

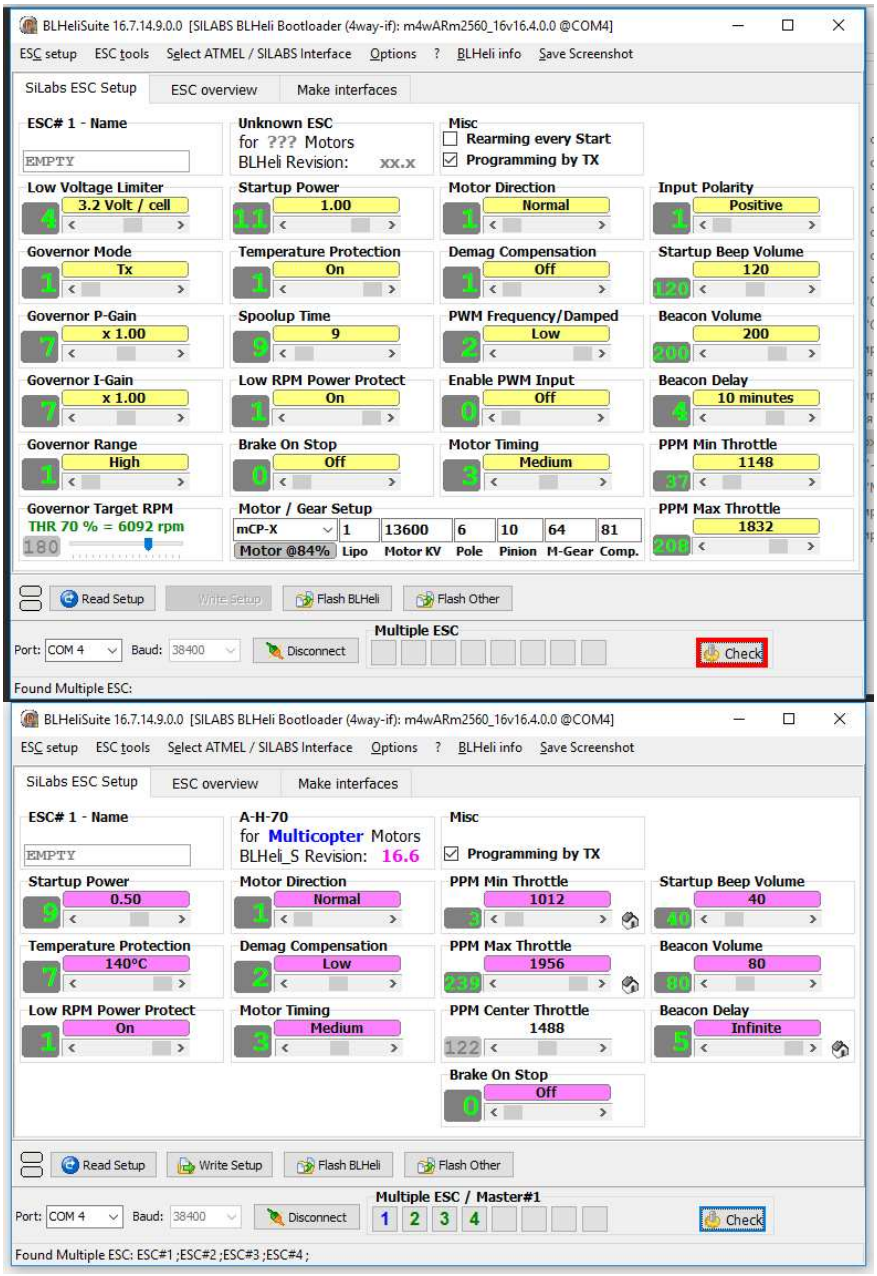
For flashing or readjusting ESCs, connect signal ports (usually white) of ESCs to Arduino ports, after checking in the manual (see the figure below), which ports are used for connecting to ESCs. You should also connect GND of Arduino with the ground of one of the ESCs (usually black). The ESCs should be connected to power, and, if electric motors are connected to ESCs, **they should not have propellers**.



In the case of Arduino Mega, signal ports of the ECSs are connected to ports D43-D49 and D51.

Changing ESC settings

To download information about firmware version and ESC settings, click on Check.

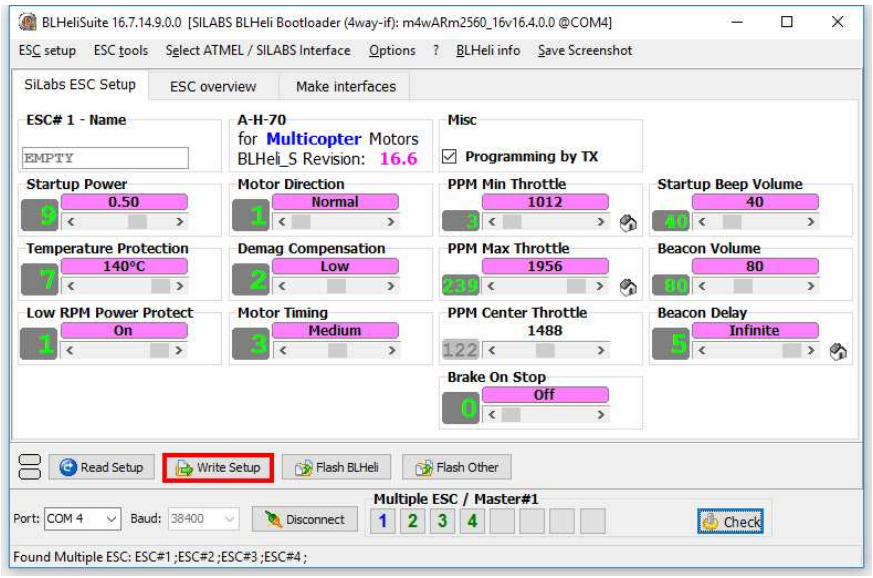


The main parameters that we are interested in are:

- Motor Direction (Normal or Reversed) - sets motor rotation direction. Convenient, if you do not wish to reconnect an incorrectly connected motor.
- PPM Min and Max Throttle - sets the minimum and the maximum throttle signal
- Startup Beep Volume - set the signal volume on startup. In firmware16.65, an ability to change the startup melody has been added. More information is available [here] (https://github.com/cleanflight/blheli-multishot/releases). For example, you can set the Imperial March from the Star Wars or the main theme from the Game of the Thrones as the startup melody
- Beacon Volume - sets the volume of the detecting signal. When the motors have not been rotating for some time, and the ESC is not used, it starts reminding about itself by motor squeaks.

- Beacon Delay - sets the duration of inactivity, after which the detecting signal is enabled. During development, it may become boring, therefore it may be set to infinity.

The leftmost motor in the list of motors (Multiple ESC) is considered the (master) motor. by clicking on motor numbers, you can enable or disable the possibility of writing their settings. After changing the necessary parameters, you can write settings to respective motors by clicking on Write Setup.



To display the settings of all ESCs simultaneously, you can use the ESC Overview tab.

Flashing ESCs

ESC firmware files are located here.

To flesh ESCs, click on button Flash BLHeli and choose the firmware file with the type of the controller, the name of which is indicated in the firmware name frame on top of the screen in tab Silabs ESC Setup (for the controller that is used in Clover 2, it is A-H-70).

To re-flash an individual ESC, disable all other ESCs.

Video guide to flashing ESCs

For a better understanding of the things written in the article, we recommend watching a video guide about connecting electronics and flashing ESCs in English on youtube.

Controlling the copter from Arduino

For interaction with ROS topics and services on a Raspberry Pi, you can use the rosserial_arduino library. This library is pre-installed on a Raspberry Pi image.

The main tutorial for rosserial: http://wiki.ros.org/rosserial_arduino/Tutorials

Arduino is to be installed on Clover and connected via a USB port.

Configuring Arduino IDE

To work with ROS and Arduino, you should understand the format of installed packages' messages. For this purpose on Raspberry Pi, build the ROS messages library:

```
rosrun rosserial_arduino make_libraries.py.
```

The obtained folder `ros_lib` is to be copied to `<sketches folder>/libraries` on a computer with Arduino IDE.

Configuring Raspberry Pi

To run the program on Arduino once, you can use command:

```
roslaunch clover arduino.launch
```

To start the link with Arduino at the startup automatically, set argument `arduino` in the Clover launch file (`~/catkin_ws/src/clover/clover/launch/clover.launch`):

```
<arg name="arduino" default="true"/>
```

After the launch file is edited, restart the `clover` service:

```
sudo systemctl restart clover
```

Delays

When `rosserial_arduino` is used, the Arduino microcontroller should not be blocked for more than a few seconds (for example, using the `delay` function); otherwise communication between Raspberry Pi and Arduino will be broken.

During implementation of long `while` cycles, ensure periodic calling the `hn.spinOnce` function:

```
while(/* condition */) {
  // ... Perform required actions
  nh.spinOnce();
}
```

To organize long delays, use the delays in a loop with periodic calling of the `hn.spinOnce()` function:

```
// 8 second delay
for(int i=0; i<8; i++) {
  delay(1000);
  nh.spinOnce();
}
```

Working with Clover

The set of services and topics is similar to the regular set in simple_offboard and mavros.

An example of a program that controls the copter by position using the `navigate` and `set_mode` services:

```
// Connecting libraries for working with rosserial
#include <ros.h>

// Connecting Clover and MAVROS package message header files
#include <clover/Navigate.h>
#include <mavros_msgs/SetMode.h>

using namespace clover;
using namespace mavros_msgs;

ros::NodeHandle nh;

// Declaring services
ros::ServiceClient<Navigate::Request, Navigate::Response> navigate("/navigate"
ros::ServiceClient<SetMode::Request, SetMode::Response> setMode("/mavros/set_mo

void setup()
{
  // Initializing rosserial
  nh.initNode();

  // Initializing services
  nh.serviceClient(navigate);
  nh.serviceClient(setMode);

  // Waiting for connection to Raspberry Pi
  while(!nh.connected()) nh.spinOnce();
  nh.loginfo("Startup complete");

  // Custom settings
  // <...>

  // Test program
  Navigate::Request nav_req;
  Navigate::Response nav_res;
  SetMode::Request sm_req;
  SetMode::Response sm_res;

  // Ascending to 2 meters:
  nh.loginfo("Take off");
  nav_req.auto_arm = false;
  nav_req.x = 0;
  nav_req.y = 0;
  nav_req.z = 2;
  nav_req.frame_id = "body";
  nav_req.speed = 0.5;
  navigate.call(nav_req, nav_res);

  // Waiting for 5 seconds
  for(int i=0; i<5; i++) {
      delay(1000);
      nh.spinOnce();
  }

  nav_req.auto_arm = false;

  // Flying forward 3 meters:
  nh.loginfo("Fly forward");
  nav_req.auto_arm = true;
  nav_req.x = 3;
  nav_req.y = 0;
  nav_req.z = 0;
  nav_req.frame_id = "body";
  nav_req.speed = 0.8;
  navigate.call(nav_req, nav_res);
```

```
  // Waiting for 5 seconds
  for(int i=0; i<5; i++) {
    delay(1000);
    nh.spinOnce();
  }

  // Flying to point 1:0:2 in the marker field
  nh.loginfo("Fly on point");
  nav_req.auto_arm = false;
  nav_req.x = 1;
  nav_req.y = 0;
  nav_req.z = 2;
  nav_req.frame_id = "aruco_map";
  nav_req.speed = 0.8;
  navigate.call(nav_req, nav_res);

  // Waiting for 5 seconds
  for(int i=0; i<5; i++) {
    delay(1000);
    nh.spinOnce();
  }

  // Landing
  nh.loginfo("Land");
  sm_req.custom_mode = "AUTO.LAND";
  setMode.call(sm_req, sm_res);
}

void loop()
{
}
```

Getting telemetry

With Arduino, you can use the `get_telemetry` service. To do so, declare it similar to the `navigate` and `set_mode` services:

```
#include <ros.h>

// ...

#include <clover/GetTelemetry.h>

// ...

ros::ServiceClient<GetTelemetry::Request, GetTelemetry::Response> getTelemetry(

// ...

nh.serviceClient(getTelemetry);

// ...

GetTelemetry::Request gt_req;
GetTelemetry::Response gt_res;


// ...

gt_req.frame_id = "aruco_map"; // frame id for x, y, z
getTelemetry.call(gt_req, gt_res);

// gt_res.x is copter position on the x axis
// gt_res.y is copter position on the y axis
// gt_res.z is copter position on the z axis
```

Problem

When using Arduino Nano, RAM may be insufficient. In this case, messages will appear in the Arduino IDE like:

```
Global variables use 1837 bytes (89%) of the dynamic memory, leaving 211 bytes
Not enough memory, the program may be unstable.
```

You can reduce RAM usage by reducing the size of the buffers allocated for sending and receiving messages. To do this, place the following line **at the beginning** the program:

```
#define __AVR_ATmega168__ 1
```

You can reduce the amount of used memory even more, if you manually configure the number publishers and subscribers, as well as the size of memory buffers allocated for messages, for example:

```
#include <ros.h>

// ...

typedef ros::NodeHandle_<ArduinoHardware, 3, 3, 100, 100> NodeHandle;

// ...
NodeHandle nh;
```

Connecting GPS

Upon connecting GPS, the following possibilities appear:

- The copter can remain at the same point when flying outside
- Autonomous missions may be programmed in the QGroundControl application
- Flying may be performed by global points in standalone mode using the simple_offboard module.

Useful links:

- https://docs.px4.io/en/assembly/quick_start_pixhawk.html
- http://ardupilot.org/copter/docs/common-pixhawk-wiring-and-quick-start.html
- http://ardupilot.org/copter/docs/common-installing-3dr-ublox-gps-compass-module.html

Connection

The GPS module is connected to "GPS" and "I2C" (compass) connectors of the flight controller.

If GPS is connected, magnetometers are to be re-calibrated in the QGroundControl application via a Wi-Fi or USB connection.

Next, GPS is to be enabled in parameter `EKF2_AID_MASK` (when EKF2 is used) or `LPE_FUSION` (when LPE is used). When using LPE the weight of the magnetometer should be more 0 (`ATT_W_MAG` = 0.1).

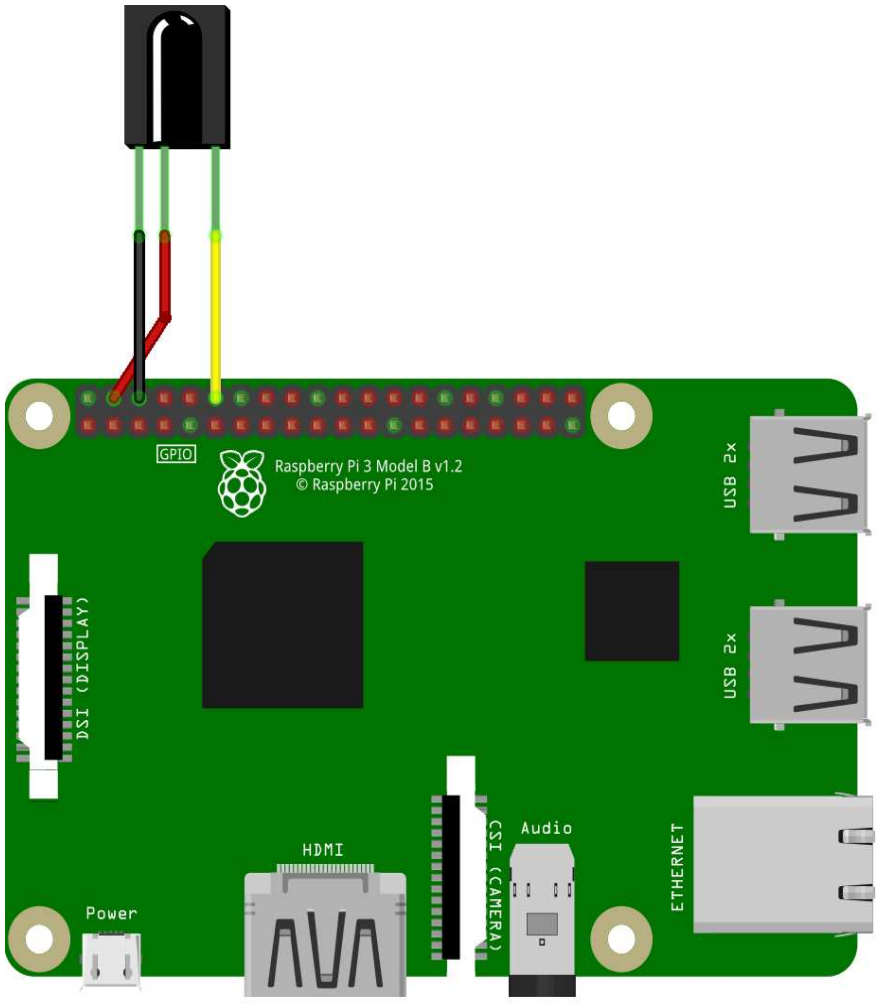
Working with IR sensors on Raspberry Pi 3

> This article is not relevant for the latest versions of the *clover* image and works only on the versions *clover_v0.16-clover_v0.17*.

Infrared sensors are a convenient tool for transmitting any commands to the copter. They are flexible in configuration, and interaction with them is possible in Python.

Connecting the IR receiver

Most IR receivers operate and are connected the same way. Such receivers have 3 pins for connecting: G/GND — ground V/VCC — 5V power, S/OUT — signal.



> The signal port doesn't have to be connected to port GPIO 17; this pin may be changed during the in/out port settings.

Configuring the IR receiver to work with the LIRC module

LIRC (Linux Infrared Remote Control) is a stable and time-proven open source library, which allows sending and receiving commands via an infrared port. LIRC is supported by Raspbian.

To install LIRC and related modules, connect your Raspberry Pi to the Internet and run the console command:

```
sudo apt-get update
sudo apt-get install lirc
sudo apt-get install python-lirc
pip install py-irsend
```

> To correctly edit the system files, superuser privileges are required; when calling a text editor, use `sudo`.

After installing the module, edit file `/etc/modules` and add line:

```
lirc_dev
lirc_rpi gpio_in_pin=18 gpio_out_pin=17
```

Where:

- `gpio_in_pin` is the input pin from the receiver
- `gpio_out_pin` is the transmitter output pin

Update the following line in file `/boot/config.txt`:

```
dtoverlay=lirc-rpi,gpio_in_pin=18,gpio_out_pin=17
```

Add the following lines to file `/etc/lirc/hardware.conf`. Is this file does not exist, create it yourself.

```
LIRCD_ARGS="--uinput --listen"
LOAD_MODULES=true
DRIVER="default"
DEVICE="/dev/lirc0"
MODULES="lirc_rpi"
```

Update the following lines in file `/etc/lirc/lirc_options.conf`

```
driver    = default
device    = /dev/lirc0
```

All required settings are made, you now have to restart your Raspberry Pi device to complete the installation. To do so, run:

```
sudo reboot
```

After rebooting, check its status by calling command:

```
sudo /etc/init.d/lircd status
```

If everything has been done correctly, the status should be `active`. To check whether the installed module LIRC is running, disable daemon `lircd`, and call the appropriate command:

```
sudo /etc/init.d/lircd stop
mode2 -d /dev/lirc0
```

Now point the IR transmitter on your device and tap a few keys. You should see something like this:

```
space 402351
pulse 135
space 7085
pulse 85
space 2903
pulse 560
space 1706
pulse 535
```

> If you are using an IR transmitter (a TV remote, an air conditioner remote, etc. and you are not getting the signal when checking, your remote is evidently using another signal frequency. When using receivers such as TSOP 22XX, the operating frequency of the signal reception will be in the range between 30 and 50 kHz.

Write your configuration of the IR transmitter

If you want to use your own IR transmitter, you will have to write its specific settings using the supplied module `irrecord`. For this purpose, disable daemon `lircd`, and call the appropriate command. During transmitter calibration, stick to all written instructions.

> Please note that the last step of the calibration will be specifying the names of the keys that you will want to decode programmatically. To view the list of available names, call command `irrecord --list-namespace`.

```
irrecord -d /dev/lirc0 ~/lircd.conf
```

If you have managed to successfully write the configuration of your transmitter, file `your-name.lircd.conf` should appear in folder `/home/pi/`. Now you need to move the written configuration file to working folder `lirc`, and restart the daemon:

```
sudo cp ~/your-name.lircd.conf /etc/lirc/lircd.conf
sudo /etc/init.d/lircd restart
```

To check whether the written configuration is recognized, call the appropriate module. Now when you tap the keys that you have specified in the previously created configuration, the terminal will show debug information about which key has been pressed.

```
irw
```

> when working with some transmitters, there are situations where the bit descriptions of keys are redundant; in this case, command `irw` may fail. To correct this error, open file `etc/lirc/lircd.conf` and check what the description of your keys looks like; if it looks like `KEY_1 0x00FF6897 0x7EE0CF2C` and in all lines the second digits match, you have to remove it, so that lines with keys assignment looks like `KEY_1 0x00FF6897` and all digits in them are unique. After completing these steps, close the file and restart the daemon.

If you did everything correctly, upon tapping a key, you will see the output similar to:

```
0000000000ff6897 00 KEY_1 pult
0000000000ff6897 01 KEY_1 pult
0000000000ff9867 00 KEY_2 pult
0000000000ff9867 01 KEY_2 pult
```

This means that your configuration is correctly detected by the program, and now you can program the desired action for tapping appropriate keys.

Working with IR sensors in Python

To be able to use signals from the IR receiver in Python programs, you'll need package `python-lirc`. Install it, if necessary.

For correct obtaining information, create file `lircrc` in your own script, which will store settings of your keys and the program response upon calling.

This file is created in the folder from which your script will be called, `/home/pi/` by default.

To create the required file, use any text editor:

```
sudo nano .lircrc
```

The format of this file should be something like this:

```
begin
prog = myprogram
button = KEY_1
config = one
end

begin
prog = myprogram
button = KEY_2
config = two
end
```

Where:

- `prog` is the name of the program that you will call from your script
- `key` is the name of the key that you entered during transmitter setup
- `config` is the information to be passed to your program upon tapping a specified key

All settings are now made, and you can proceed directly to programming IR signals.

For this purpose, create a Python script that will accept the values of the keys pressed and perform required action s accordingly. An example of such a script:

```
import lirc
import fly_module

# ...

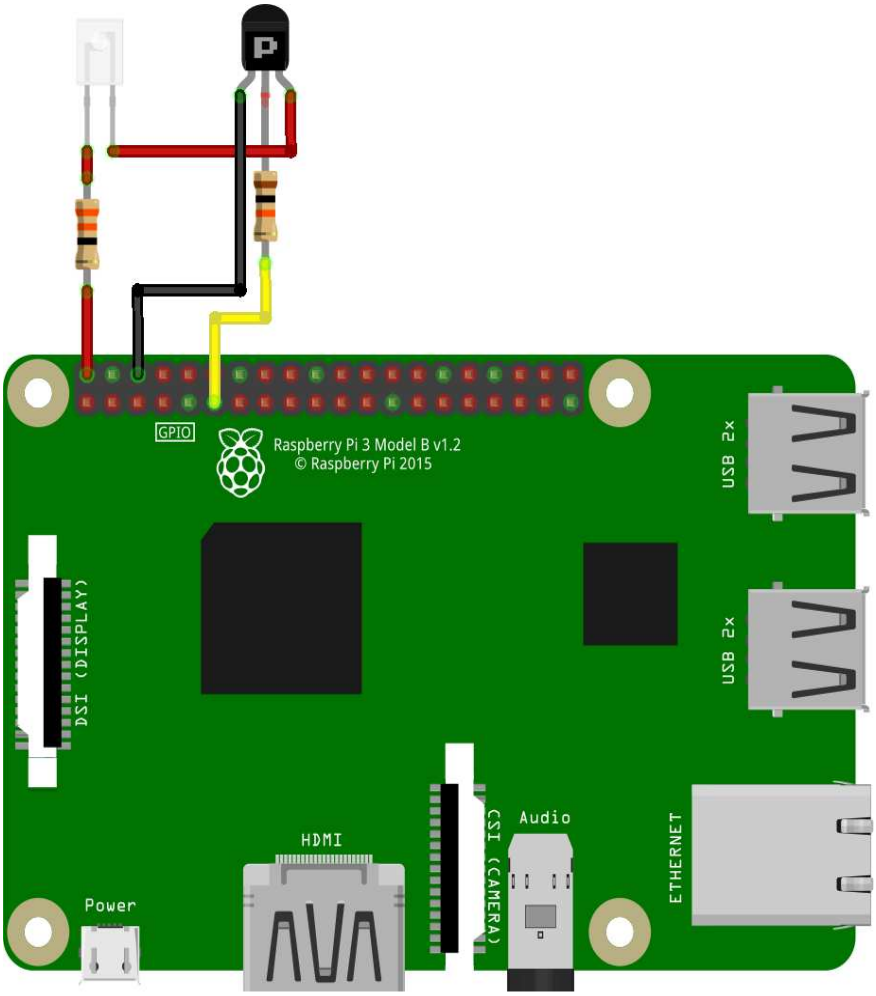
sockid = lirc.init('myprogram')

inf = lirc.nextcode()
if inf[0] == 1:
    print('You pressed key 1')
elif inf[0] == 2:
    print('You pressed key 2')

lirc.deinit()
```

Working with the IR transmitter

To work with the IR transmitter, connect it to the ports specified during setup.





> if you are using a ready IR transmitter board, connect it to required pins of Raspberry in accordance with pins marking, in the same way as with the receiver.

If everything has been properly connected, you will be able to send signals specified in transmitter settings using the command:

```
irsend SEND_ONCE deviceName keyName
```

Where:

- SEND_ONCE is the parameter responsible for sending a single signal, or sending a signal from a depressed and held down key
- deviceName is the name of the transmitter specified during setup
- keyName is the name of one of the keys specified during transmitter configuration

To work with `irsend` inside your script, you'll need module `python-irsend`; if necessary, install it.

To use `irsend`, import the library and call the appropriate command:

```
from py_irsend import irsend


irsend.send_once('YourRemote', ['YourKey'])
```

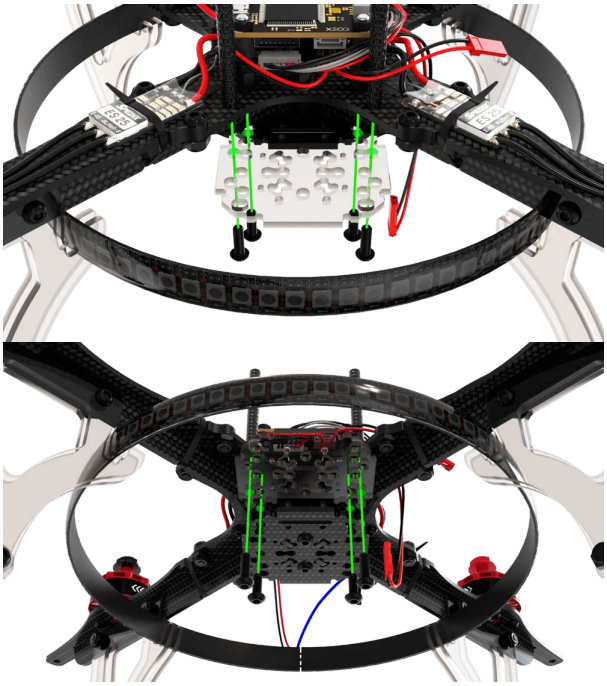
Where:

- YourRemote is the name of your transmitter specified during setup
- YourKey is the name of one of the buttons specified during setup

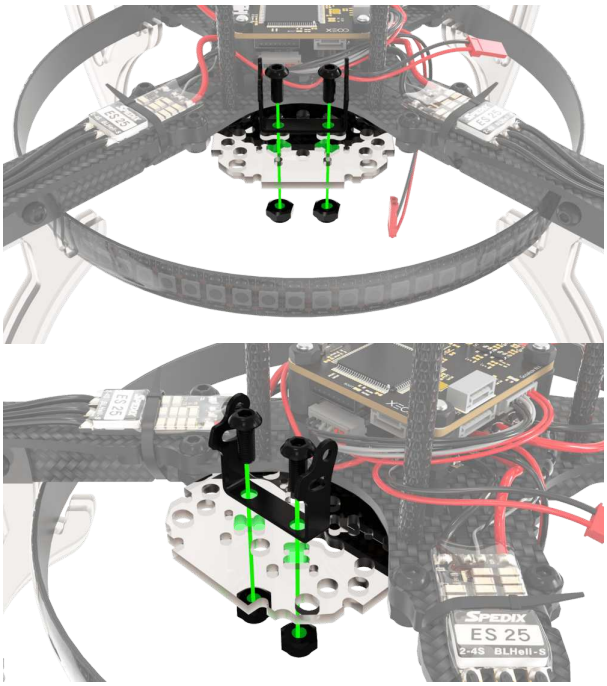
Installing and configuring FPV equipment

Preparing and installing the FPV camera and transmitter

1. Install the small mounting deck onto the main frame.



2. Install the camera mount bracket into the corresponding holes.



3. Cut the three-pin supplied camera cable.





4. Tin the wires



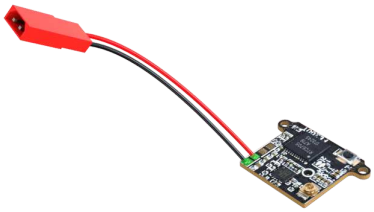
5. Solder the JST-male connector to the power wires of the camera.

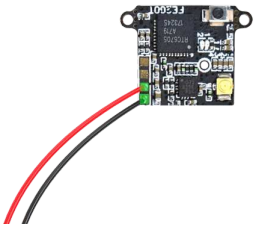




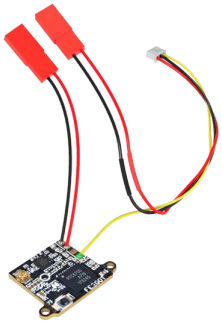
> Check what you are wearing shrink tubes before soldering the wires.

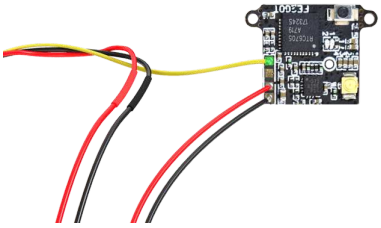
6. Solder the JST male connector to the transmitter.



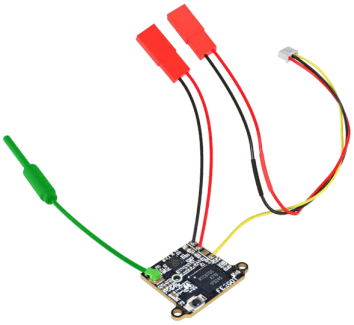


7. Solder the yellow camera signal cable to the transmitter.



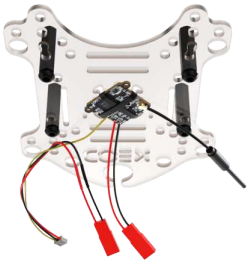


8. Connect the antenna to the transmitter.

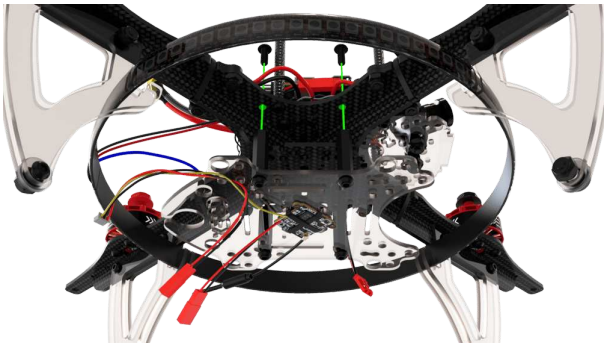


> If voltage is applied to a transmitter without an antenna, there is a high probability that it will burn out.

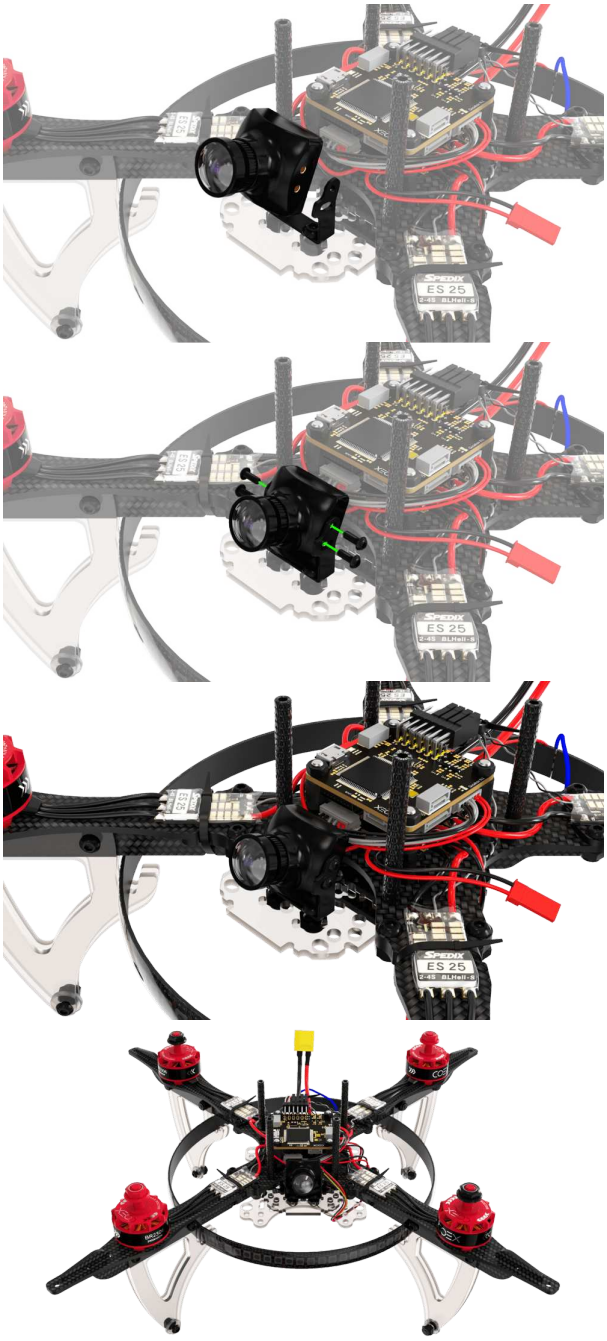
9. Place the receiver onto the mounting deck, securing it with ties.



10. Place the mounting deck with the receiver on the bottom of the aircraft.



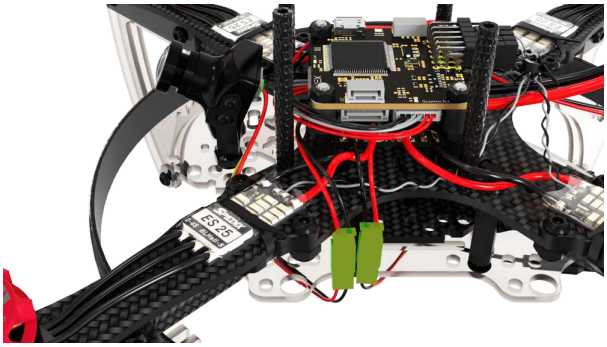
11. Place the camera in the bracket and secure it with the 4 attached bolts. The camera should be at an angle of 15°-20° relative to the plane of the aircraft.



12. Connect the signal cable to the camera.



13. Connect the camera's power cable to the power JST soldered to the *BAT+* and *GND* pads on the power distribution board.
14. Connect the transmitter power cable to JST at 5V.



Setting up and connecting FPV goggles

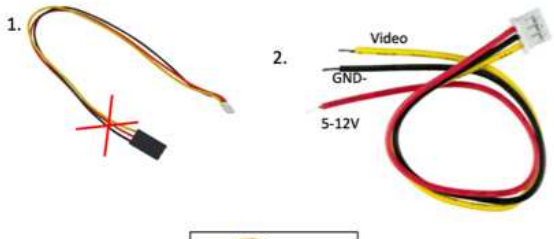
1. Install the two supplied antennas on the glasses.
2. Turn on the glasses by holding the power button for 3-4 seconds.
3. Turn on the aircraft and make sure the transmitter LED is blue.
4. Press the *Auto Search* button on the glasses to automatically search for an available radio channel.

Installation of FPV

Preparation of the FPV camera

1. Take the connector wire from the camera and cut off the BLACK side of the 3-pin connector.
2. Prepare the wire leads to be connected:
    i. Shorten the wires to the desired length *.
    ii. Strip (remove 2 mm of insulation from the end of the wire without damaging the strands).
    iii. Twist the wires.
    iv. Blanch using tweezers.

* The distance between the power distribution board and the estimated location of the camera should be determined in advance!





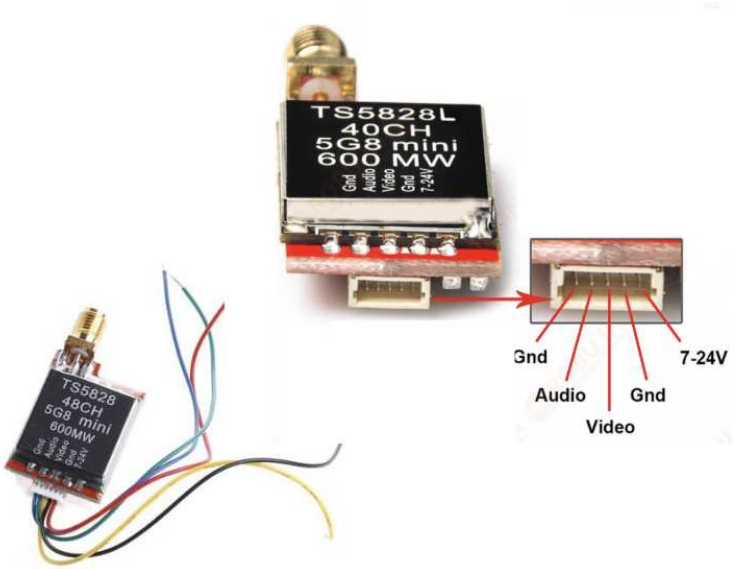
Course camera
• Power: DCS-12V
• Video Output: 1.0 V p-p, 75 Ohms

Preparation of the transmitter

The same procedure applies here:

1. Take the connector wire from the transmitter and cut off the BLACK side of the 5-pin connector.
2. Prepare the wire leads to be connected:
    i. Shorten the wires to the desired length *.
    ii. Strip (remove 2 mm of insulation from the end of the wire without damaging the strands).
    iii. Twist the wires.
    iv. Blanch using tweezers.

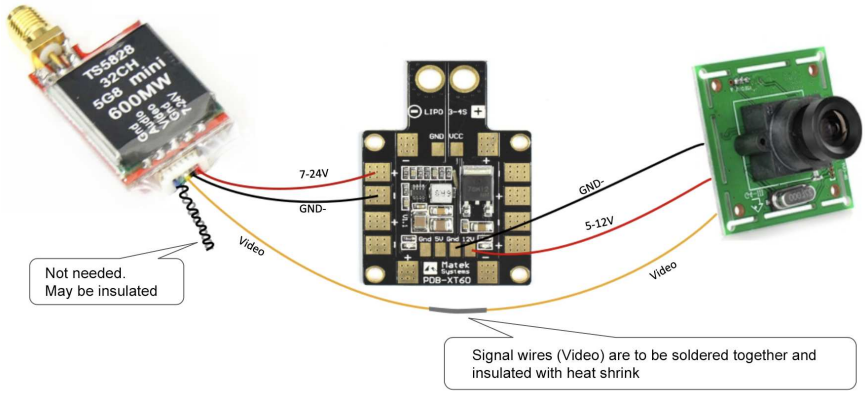
* The distance between the power distribution board and the estimated location of the transmitter should be determined in advance!



Transmitter power supply: DC7-24V

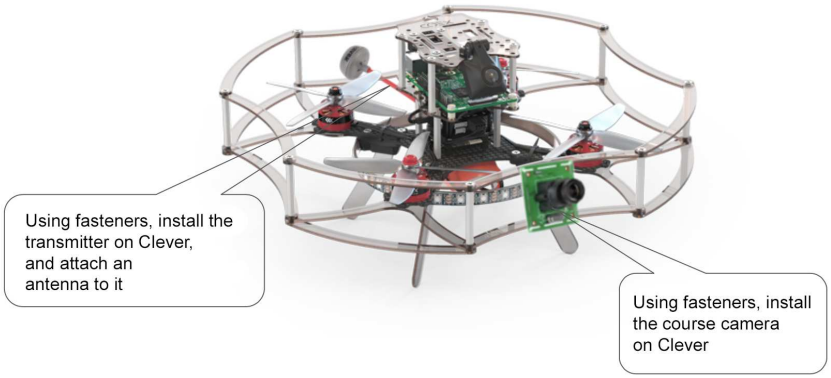
Connection of FPV

Prepared connectors are to be inserted into appropriate sockets, and power wires are to be soldered to the power distribution board according to the circuit diagram:



> In this circuit diagram, the camera is powered from 12 V (however, it is possible to use 5 V). The transmitter is powered from the ESC power (however, it is possible to use 12 V).

Installing FPV components

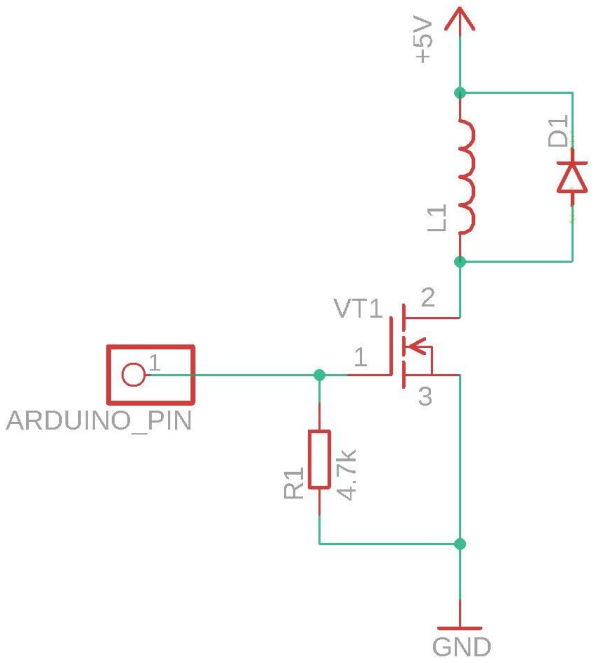


The following may be used as fastening materials:

1. Hot-melt glue;
2. electrical tape;
3. zip-ties (clamps);
4. double-sided adhesive tape.

Assembling and setting up the electromagnetic gripper

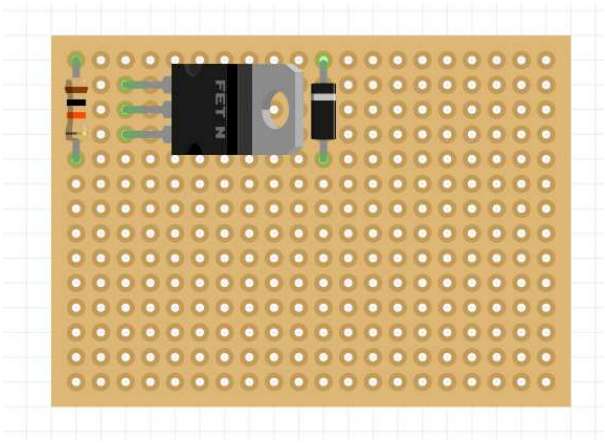
The magnetic gripper can be assembled in various ways according to the wiring diagram.



The following is an example of assembling an electromagnetic capture circuit on a breadboard.

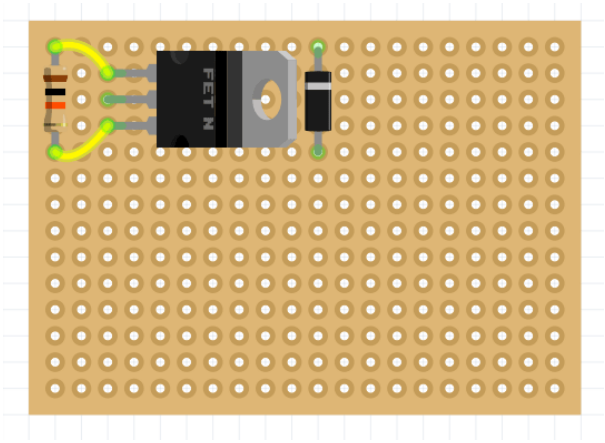
> It is recommended to lay the wiring between the elements on the back side of the board (in the following images, the wiring is done over the diagram for illustrative purpose).

1. Place the Schottky diode, 10K resistor, and transistor on the soldering board.

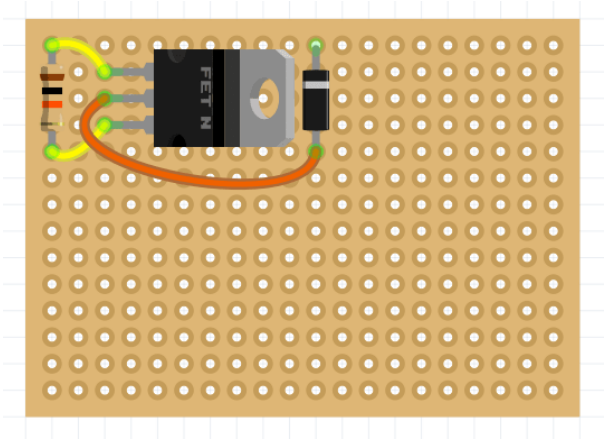


2. Solder the contacts on the other side of the board and bite off the remaining element legs.

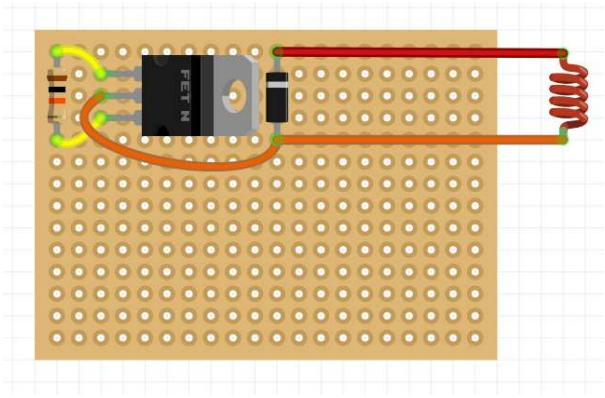
3. Connect the pins of the resistor and the two outer legs of the transistor.



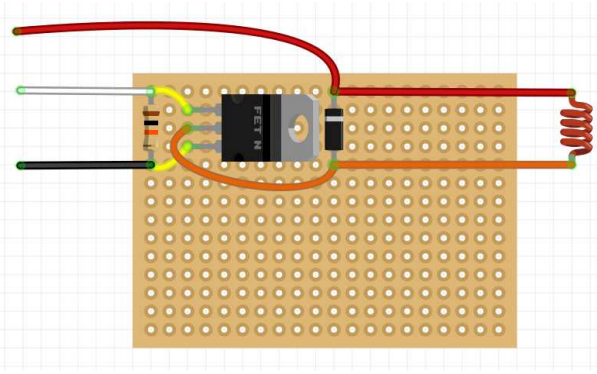
4. Connect the center leg of the transistor and the leg of the Schottky diode (opposite to the gray marking strip).



5. Cut the required amount of magnetic grab wire and solder it to the pins of the Schottky diode.

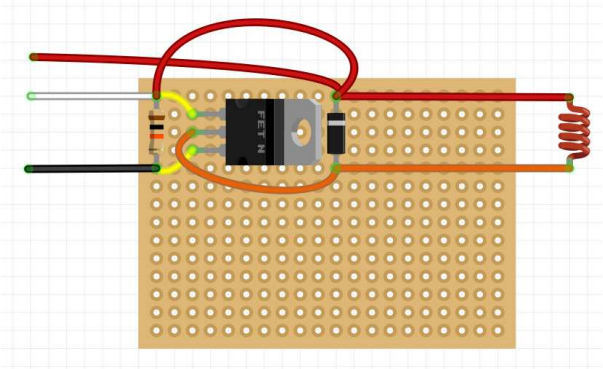


6. Solder the *Dupont* - male wires to the transistor and diode leg (red, black wires), and the *Dupont* - fmale wire to the opposite transistor leg (white wire).



Checking the operation of the electromagnetic gripper

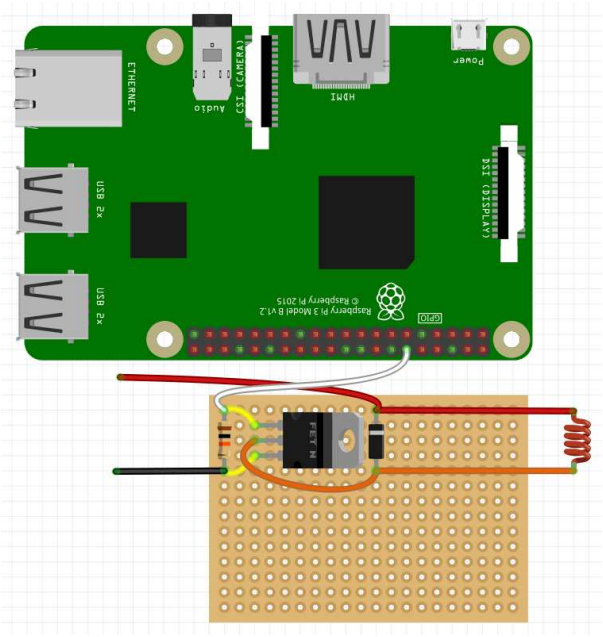
In order to check the operation of the gripper, apply a voltage of 5V to the signal wire. You can use the *Dupont* dad-dad wire for that.



After applying voltage, the magnet should turn on.

Connecting to Raspberry Pi

Connect the magnetic gripper to a Raspberry Pi for software activation.

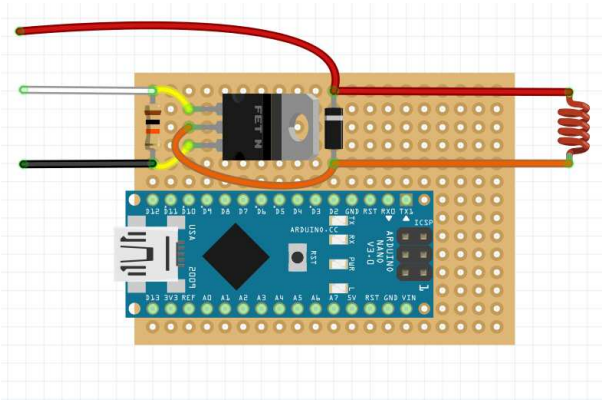


An example of the code activating the magnetic gripper can be found here.

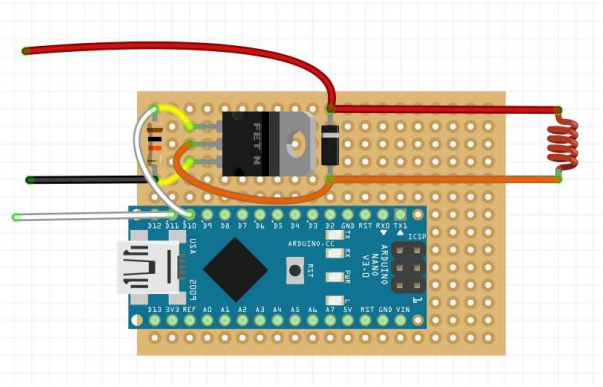
Connecting to Arduino

Connect the gripper to the Arduino Nano board in order to control it manually.

It is convenient to place it on the same soldering board — insert it into the appropriate holes and solder it from the back to the board.



Then connect the signal output of the circuit to the selected port and solder the *Dupont* female wire to the selected signal port on the board.



Installation of electromagnetic gripper

1. Install an electromagnet into the center hole on the gripper deck.
2. Use a zip tie to pull the assembled circuit to the back of the deck.
3. Plug the Arduino *D11* signal pin into one of the *AUX* pins on the flight controller.
4. Plug the power wire of the electromagnetic gripper to JST 5V.

Setting up electromagnetic gripper

To control the magnet through Arduino Nano, use the following code:

```
void setup() {
  pinMode(11, INPUT);
  pinMode(13, OUTPUT);
}

void loop() {
  if (int duration = pulseIn(11, HIGH) > 1200) {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

To monitor the status of the electromagnetic gripper, you can connect the *ws281x* LED strip (included to Clover kit). Connect it to power +5v – 5v, ground GND – GND, and signal wire DIN – Arduino D12.

To control the magnet and monitor it using the LED strip, use the following code:

```
#include <Adafruit_NeoPixel.h>
#define NUMPIXELS 72
#define PIN 12
int pin = 11;
int led = 13;

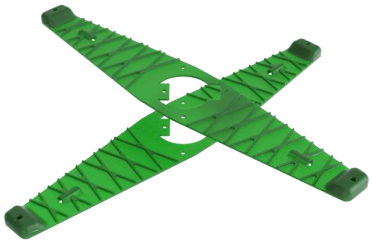
unsigned long duration;
Adafruit_NeoPixel strip (NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
  strip.setBrightness(10);
  Serial.begin(9600);
  pinMode(pin, INPUT);
  pinMode(led, OUTPUT);
}

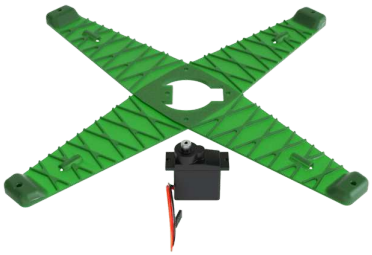
void loop() {
  duration = pulseIn(pin, HIGH);
  Serial.println(duration);
  delay(100);
  if (duration >= 1500) {
    digitalWrite(led, HIGH);
    for (int i = -1; i < NUMPIXELS; i++) {
      strip.setPixelColor(i, strip.Color(255, 0, 0));
      strip.show();
    }
  } else {
    digitalWrite(led, LOW);
    for (int i = -1; i < NUMPIXELS; i++) {
      strip.setPixelColor(i, strip.Color(0, 255, 0));
      strip.show();
    }
  }
}
```

Assembling and setting up a mechanical gripper

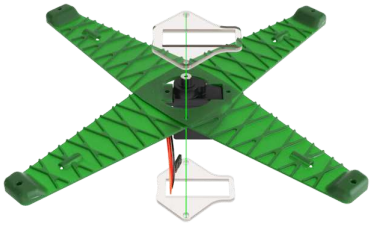
1. Combine the main gripper plates.



2. Install the servo in the appropriate groove in the center of the plates with the axle gear in the middle.



3. Hold down the gripper plates with the small spacers.

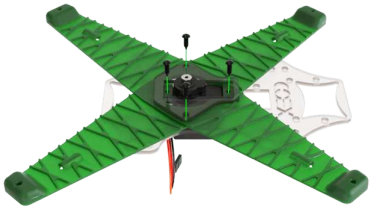




4. Install the gripper deck so that the mounting holes in the grapple correspond with the self-tapping holes in the plate.



5. Fix the gripper structure with self-tapping screws.



6. Turn the servo gear to the end position.

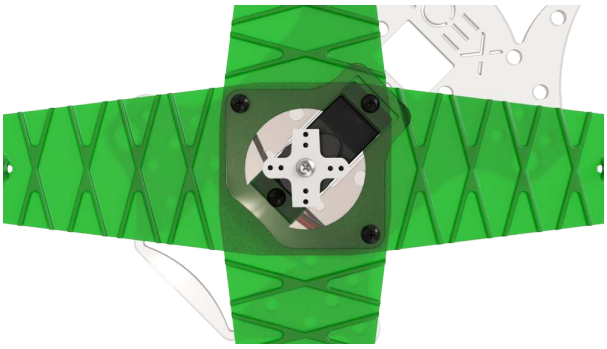


7. Install the cruciform mount onto the gear and secure it using the screw attached to the servo.



8. Cut the cruciform mount.





9. Tie the servo thread so that there is a 2 to 3 cm margin.





10. Thread the servo thread into the corresponding tensioning slots.

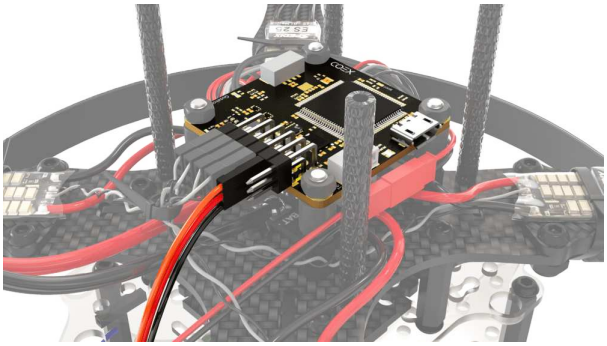


11. Fasten the grip claws with small self-tapping screws so that their angle is 25°–40°.

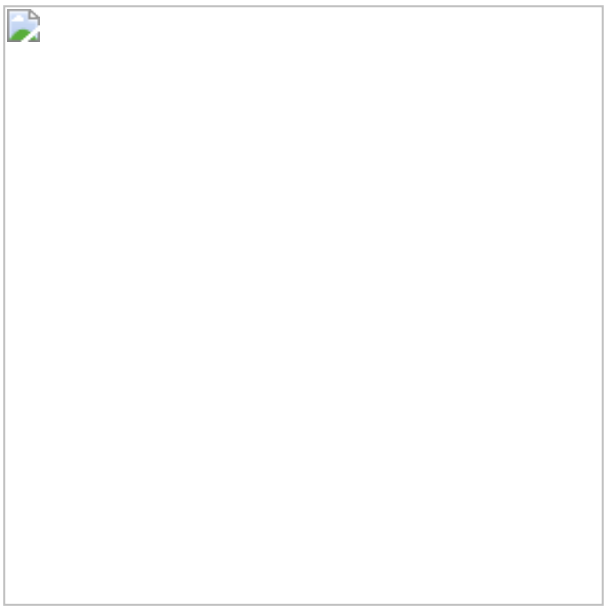


12. Install the assembled grip onto the aircraft from below.





13. Insert the servo cable into the *AUX* 1-2 output on the flight controller.



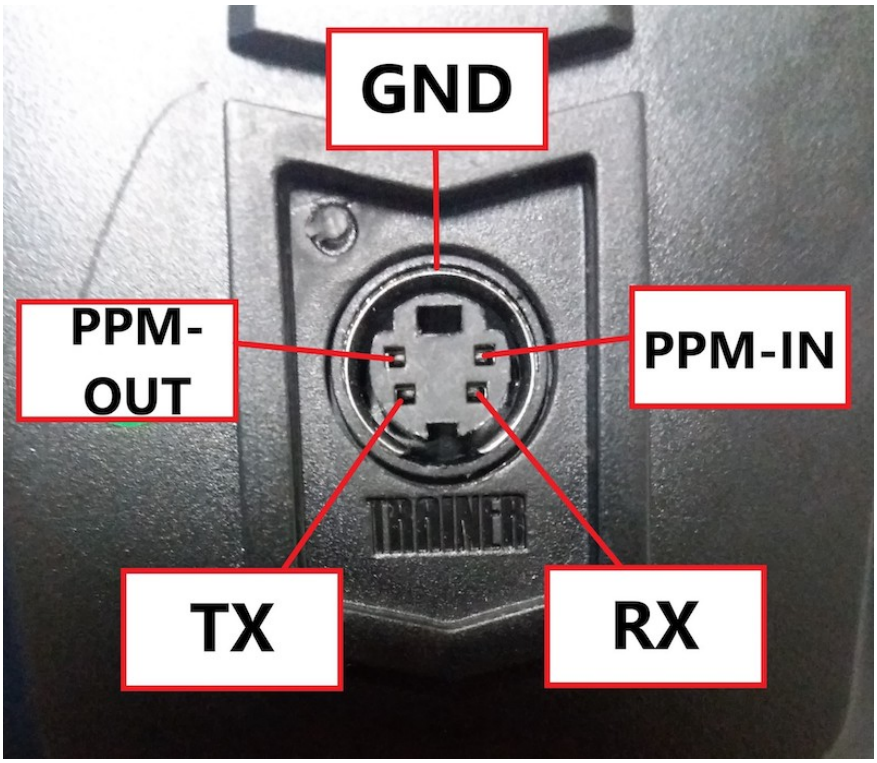
14. Go to the *Radio* tab to control capture with the remote control.
15. In the *AUX 1/2 Passthrough RC channel* parameter, select the desired channel.
16. Now, when you switch the toggle switch of the corresponding channel, the capture will be closed or opened.

FlySky Trainer mode settings

The trainer mode is used for teaching students how to fly a drone. In trainer mode the experienced pilot is ready to take control of the drone in case of emergency situation during the flight.

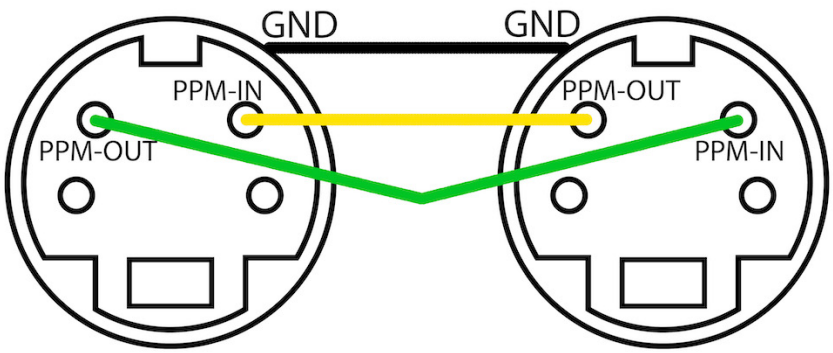
For that we connect two remote controllers. The first controller (Slave) is for a student and the second (Trainer) is for a teacher.

Wire pinout



To create a link between remote controllers, a connector is used at the back of the case (S-Video). Three contacts in the connector are used for receiving, transmitting information and for ground. The PPM-OUT (transmit) contact must be connected to the PPM-IN (receive) and vice versa. To avoid environmental interference, ground contacts must be interconnected.

This is how we need to solder the wires to the connector.



Trainer’s settings

Go to settings (hold OK button). Then to the System setup and look for (Up / Down) Trainer mode.

To activate mode, the Mode line should be ‘ON’. Use the Up/Down buttons to change the parameter. To save the parameter, click OK.

Now select the switch for taking the control:

Do it in the Trainer mode menu. In the Switch line, select (you can change using Up/Down) any convenient switch (SwA, SwB, SwC, SwD). Trainer now will take control of the drone after toggling the switching.

To save the settings hold Cancel.

> Both teacher’s and student’s remote controllers must be in the same flight mode.

Student’s settings

Go to the settings, System setup and select the student mode (Student mode). Then click OK and use Up\Down to select ‘Yes’. Hold Cancel to save the settings. If everything is configured correctly, the letter S will appear on the main screen.

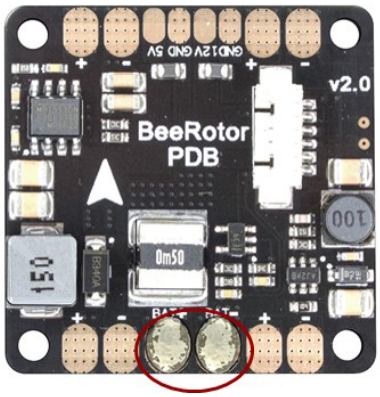
Blanching

> Before soldering and blanching, make sure that the wires and the circuit-boards are disconnected from the power supply (de-energized)!

Blanching contact pads

Blanching a contact pad means doing the following:

1. Apply flux on the contact pad
2. Cover the contact pad with solder



1. Blanch contact pads BAT+ and BAT-
2. Blanch other contact pads

Blanching wires

Blanching a wire means doing the following:

1. Strip the wire to remove the insulation layer
2. Twist stripped wires
3. Apply flux to the twisted stripped wires
4. Apply a layer of solder.



Types of power connectors

XT-60

It is one of the most reliable power connectors that they try to use on power batteries. These specific connectors are used on Li-Po batteries for copters.



T-plug

An analog for XT-60. It has various options for simplifying disconnection.



JST-XH or a balancing connector

Connectors of this type are often used for balancing individual elements in an assembly of several lithium-polymer (Li-Pol), lithium-ion (Li-ion), or lithium-phosphate (LiFePO4) batteries. Similar connectors with various numbers of pins are installed in most modern chargers for balancing the lithium cells during charging. This connector may be used together with a buzzer (beeper) for monitoring battery charge.



Gold bullet connectors, or bananas

There is a great variety of Gold bullet pin connectors. Connectors of this type may have different diameters and size. The most widespread connectors are those with the diameter of 2 mm, 3 mm, and 4 mm. They are often used for solderless connections on PDB and motors.



Connecting 4 in 1 ESCs

4 in 1 ESC circuit board pin-out

Appropriate phase wires and their control signal (Fig. 1b) are marked with the same color (Fig. 1a).

For example, orange color -> bottom-right motor -> S1 - orange wire.

Pixracer flight controller pin-out

Fig. 2a shows the pin-out of the terminal strip:

- **SIGNAL** — ESC connection. Every pin has its own signal. Pins 5 and 6 can receive a PWM signal (for example, a servo may be connected).
- **GND** is the ground of the flight controller. A common bus on all GND pins (marked in black).
- 1, 2, 3, 4 are ports for connecting ESCs.
- 1, 2 are ports for expanding the output PWM signal (can be setup in QGroundControl, can also can be used to control the hexacopter).

  Fig. 2b shows motor numbering of the Pixracer flight controller.

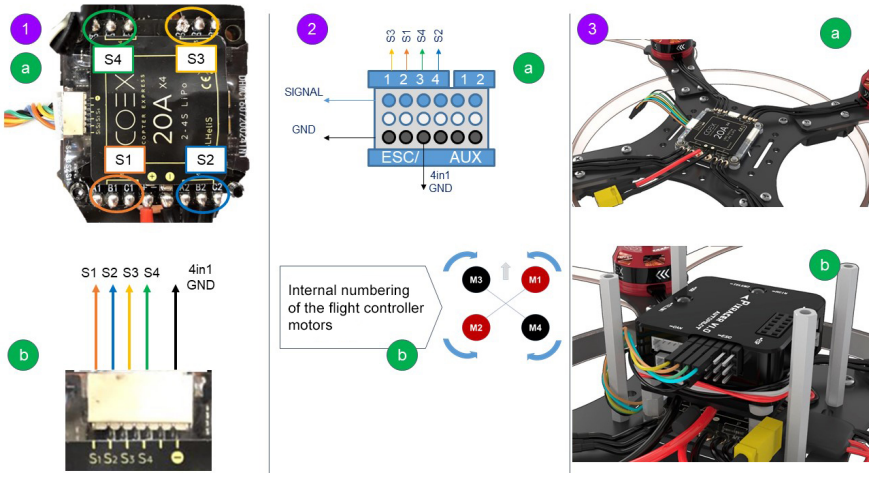
- The arrow is the flight controller orientation.

- Black M3, M4 are the motors that rotate clockwise.
- Red M1, M2 are the motors that rotate counter-clockwise.

Picture of the connection, based on the current orientation of the 4 in 1 ESC board

Using Fig. 1a, 1b, 2a, 2b, map its own control signal to each motor, and connect in accordance with the Pixracer motor numbering order.

For example, motor M3 that rotates counter-clockwise (top left corner) is controlled by signal S4 (green wire). It is connected to port 3.


1
a
S4
S3
S1
S2
20A
2
S3 S1 S4 S2
1 2 3 4 1 2
SIGNAL
GND
ESC/ AUX
4in1
GND
a
3
a
b
S1 S2 S3 S4
4in1
GND
S1 S2 S3 S4
Internal numbering
of the flight controller
motors
M3
M1
M2
M4
b
b

Soldering safety

All work involving soldering and blanching should be performed on specially equipped and prepared premises. There must be a ventilation system.

Before you start, do the following:

1. Tidy up your workplace, nothing should interfere with the process. The workplace should be well illuminated.
2. The working soldering iron is to be placed in the area of local exhaust ventilation, in a special holder.
3. Before work, put on a protective gown, goggles, and gloves, if necessary.



When soldering:

1. The soldering iron should be held only by the handle since the tip is hot.



2. Items are to only be moved using special tools (tweezers, pliers or other tools) that ensure safety when soldering.
3. To avoid burns by molten solder when soldering, do not pull the soldered wires out abruptly with great effort.
4. When soldering small and mobile items use a special holder.



5. Carry the soldering iron by the handle, rather than the cable or the working part. During breaks, the soldering iron is to be disconnected from the mains.

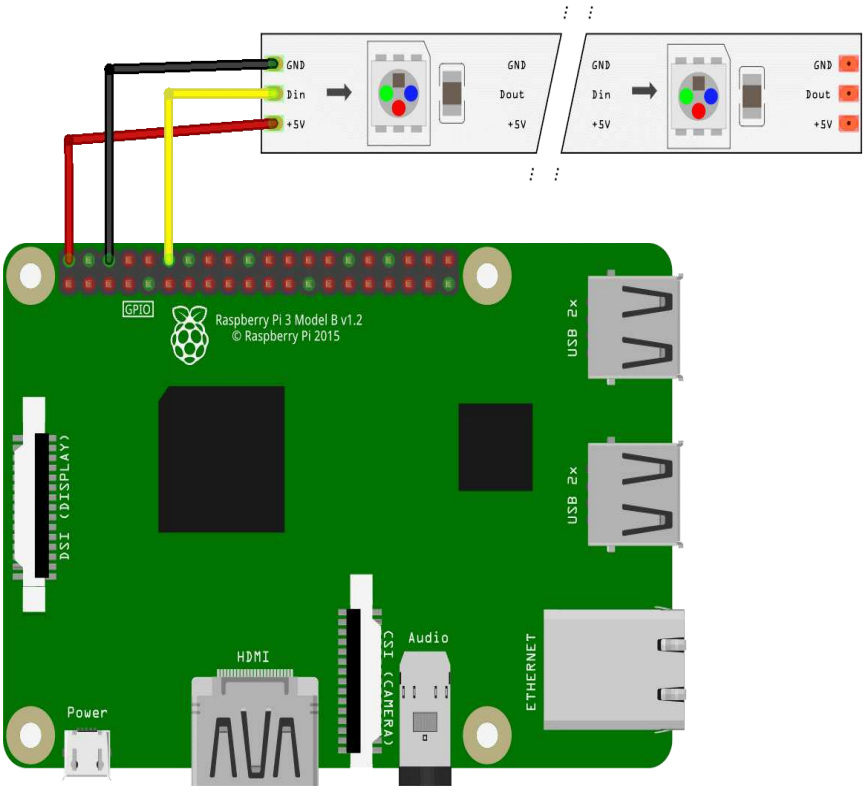
> In case of the soldering iron malfunction or fire, disconnect it from the mains.

Working with a LED strip on Raspberry 3

Connecting and determining the type of the strip

> The following is applicable to image versions 0.14 and up. For versions 0.13 and older see an older revision of this article

Connect the +5v and GND leads of your LED to a power source and the DIN (data in) lead to GPIO18 or **GPIO21** (by default).



> LED strip can consume a lot of power! Powering it from a Raspberry Pi may overload the computer's power circuitry. Consider using a separate BEC as a power source.

> If you are using GPIO along with the LED strip, connect the strip to GPIO21. Otherwise you may experience unintended strip behavior.

ROS and Python compatibility

LED strip library requires you to run your Python scripts with `sudo`. In order to make it work with ROS nodes you have to add the following lines to your `/etc/sudoers` file on the Raspberry Pi:

```
Defaults        env_keep += "PYTHONPATH"
Defaults        env_keep += "PATH"
Defaults        env_keep += "ROS_ROOT"
Defaults        env_keep += "ROS_MASTER_URI"
Defaults        env_keep += "ROS_PACKAGE_PATH"
Defaults        env_keep += "ROS_LOCATIONS"
Defaults        env_keep += "ROS_HOME"
Defaults        env_keep += "ROS_LOG_DIR"
```

Sample program for the LED strip

The following code lights up the first 10 LEDs on the LED strip. You may use it to check whether your LED strip works correctly:

```
import time

from rpi_ws281x import Adafruit_NeoPixel
from rpi_ws281x import Color


LED_COUNT      = 10      # Number of LED pixels
LED_PIN        = 21      # GPIO pin for the strip
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN transistor
LED_CHANNEL    = 0       # Set to '1' for GPIOs 13, 19, 41, 45 or 53

strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT)

strip.begin()


def colorWipe(strip, color, wait_ms=50):
    """Wipe color across strip a pixel at a time."""
    for i in range(strip.numPixels()):
        strip.setPixelColor(i, color)
        strip.show()
        time.sleep(wait_ms/1000.0)


print('Color wipe animations.')
colorWipe(strip, Color(255, 0, 0), wait_ms=100)  # Red wipe
colorWipe(strip, Color(0, 255, 0), wait_ms=100)  # Blue wipe
colorWipe(strip, Color(0, 0, 255), wait_ms=100)  # Green wipe
colorWipe(strip, Color(0, 0, 0), wait_ms=100)    # Turn LEDs off
```

> You may also want to use additional test scripts from the LED library repository.

Save the script and run it as root:

```
sudo python led_test.py
```

Basic LED library functions

You'll need to import `Adafruit_NeoPixel` class and `Color` function into your program to interact with the LED strip. Additionally, you'll want the `time` module to add delays to your animations:

```
from rpi_ws281x import Adafruit_NeoPixel
from rpi_ws281x import Color
import time
```

Instantiate the `Adafruit_NeoPixel` object and call its `begin()` method to start working with the strip:

```
# Strip object instantiation (parameter description is provided in a code snipp
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT)
strip.begin()
```

Main strip control methods:

- `numPixels()` returns the number of pixels in the strip. Convenient for whole strip operations.
- `setPixelColor(pos, color)` sets the pixel color at `pos` to `color`. Color should be a 24-bit value, where the first 8 bits are for the red channel, the next 8 bits are for the green channel, and the last 8 bits are for the blue channel. You may use the `Color(red, green, blue)` convenience function to convert colors to this format. Each color value should be an integer in the [0..255] range, where 0 means zero brightness and 255 means full brightness.
- `SetPixelColorRGB(pos, red, green, blue)` sets the pixel at `pos` to the color value with components `red`, `green` and `blue`. Each component value should be an integer in the [0..255] range, where 0 means zero brightness and 255 means full brightness.
- `show()` updates the strip state. Any changes to the strip state are only pushed to the actual strip after calling this method.

Does it have to be this way?

The LED strip type used in the Clover kits use the following protocol: a data source (a Raspberry Pi, for example) sends a bit stream, 24 bits per LED. Each LED reads the first 24 bits from the stream and sets its color accordingly while passing the rest of the stream to the next LED. Zeroes and ones are encoded by different pulse lengths.

This LED strip is supported by the rpi_ws281x library. The library uses the DMA (direct memory access) module of the Raspberry CPU and can utilize one of the three periphery channels: PWM, PCM or SPI. This allows the library to drive the strip consistently in a multitasking environment.

Each channel has its caveats. Using the PWM prevents you from using the builtin Raspberry audio subsystem; using the PCM channel will prevent you from adding I2S (digital audio) devices, although the analog audio will work. SPI requires you to change your GPU and buffer size and prevents you from using any SPI devices.

Some DMA channels are reserved for system use. DMA channel 5 is used for SD card reads and writes, and setting LED_DMA to 5 will corrupt your filesystem. DMA channel 10 is considered to be safe.

You have the following options for the LED strip:

1. If you don't need onboard audio, you may use the PWM channel and connect the LED strip to one of the following GPIO pins: 12, 18, 40 or 52 for PWM0 and 13, 19, 41, 45 or 53 for PWM1.
2. If you don't care about SPI devices, you may use the SPI channel for the LED with GPIO pins 10 or 38. You'll have to perform the following adjustments:
    - increase the SPI device buffer by adding `spidev.bufsiz=32768` option to `/boot/cmdline.txt`;
    - set the GPU frequency to 250 MHz by adding `core_freq=250` to `/boot/cmdline.txt`;
    - reboot your Raspberry Pi: `sudo reboot`.
3. If you care about audio and SPI devices, you may want to use the PCM channel (GPIO 21 or 31). You don't have to reconfigure your Raspberry.

The default option is 3, because it allows the builtin audio system to work and does not require any modifications to the boot sequence.

Contribution to Clover

Clover is mostly an open source and open hardware project aimed at lowering the entry threshold to development of the projects related to flying robotics. You can contribute to the project by offering fixes and improvements for Clover documentation and software.

> To offer changes to Clover documentation or SW, you should have an account at GitHub.

Markdown

All Clover documentation is written in the widespread Markdown format. There are many Markdown guides on the Internet.

In Russian: https://guides.hexlet.io/markdown/.

In English: https://www.markdownguide.org/getting-started, https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet.

For the ease of editing texts, you may use text editors with Markdown support: Typora, Dillinger (web), VSCode with the Markdown Editor plugin.

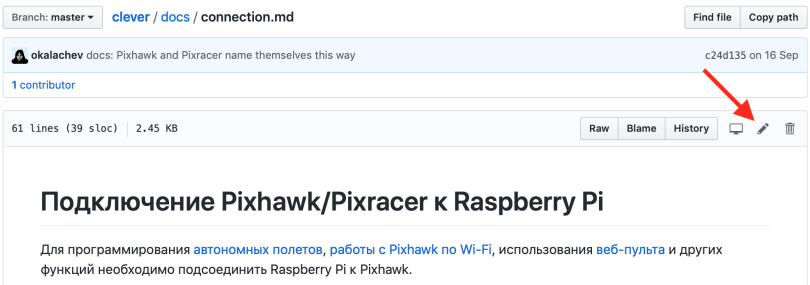
We also recommend using the Code Spell Checker VScode plugin.

For a local build of a static documentation website, use the `gitbook-cli` utility.

Fixing documentation errors

If you have found an error in the documentation or if you want to improve it, use the **Pull Request** mechanism.

1. Find a file with the article you want in the repository – https://github.com/CopterExpress/clover/tree/master/docs.
2. Click "Edit".



3. Make the necessary changes.
4. Click "Propose file change".
5. Describe the change you have made, and click "Create a Pull Request".
6. Wait for your changes to be approved :)

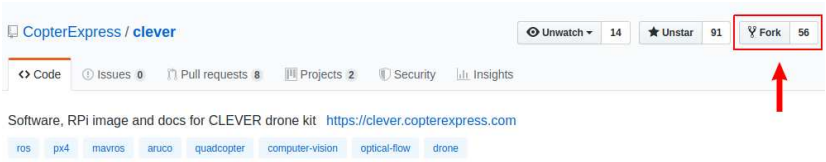
More information about Pull Requests is available at GitHub (English) or in GIT documentation (Russian).

Contributing a new article

> If you've made your own project based on Clover, you can add an article about it to the "Clover-based projects" section.

Prepare your article and send it as a pull request to the Clover repository.

1. Fork the Clover repository:



2. Check out the freshly-forked repository on your computer:

```
git clone https://github.com/<USERNAME>/clover.git
```

3. Open the directory with the source code checkout and create a new branch for your article (for example, `new-article`):

```
git checkout -b new-article
```

4. Write a new article in the Markdown format and save it in the `docs/ru` or `docs/en` folder (for example, `docs/en/new_article.md`). Don't forget to give you contacts (e-mail / Telegram / ...) in articles on your projects.
5. Place additional visual assets in the `docs/assets` folder and add them to your article.
6. Add a link to your article to the appropriate section in the `SUMMARY.md` file (in the same folder as in the fourth step):

```
...
* Supplementary materials
  * [COEX Pix](coex_pix.md)
  * [Contribution guidelines](contributing.md)
  * [New article](new_article.md)
  * [RC troubleshooting](radioerrors.md)
  * [Flashing ESCs](esc_firmware.md)
...
```

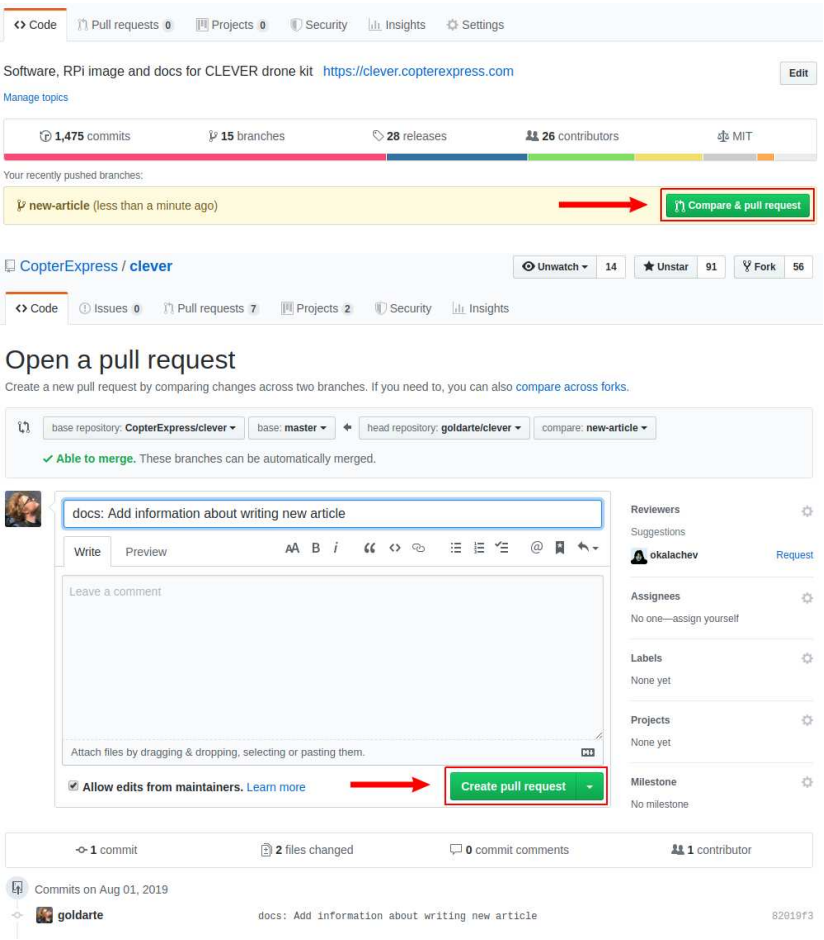
7. Commit your changes locally:

```
git add docs/
git commit -m "Add new article for Clover"
```

8. Upload your branch to your forked repository on GitHub:

```
git push -u origin new-article
```

9. Open your repository on GitHub and send a `pull request` from your branch to Clover:



10. Wait for the review, be ready to make changes if needed.
11. Look at your new and useful article at https://clovercoex.tech !

Easy way

If the above instructions are too difficult for you, send your fixes and new articles by e-mail (okalachev@gmail.com) or in Telegram messenger (user @okalachev).

Publishing packages

You also can publish a package, that extends Clover functionality, into the official COEX Debian repository.

COEX packages repository

COEX provides an open Debian-repository with ROS Noetic related prebuilt binary pacakges for `armhf` architecture.

> Repository URL: http://packages.coex.tech.

The repository is already addedd in RPi image and may be used for simple installation of additional ROS packages.

Packages publishing

You can make a Pull Request in a git repository with packages, adding or updating your package (a file with `.deb` extension), that relates to Clover or ROS. After merging your package will be available for installation with `apt` utility:

```
sudo apt install ros-noetic-clover-some-feature
```

Packages, that extend Clover functionality are recommended to be named with `clover_` prefix, e. g. `clover_some_feature`.

Using on a normal Raspberry Pi OS

On a normal Raspberry Pi OS, the repository may be added to the sources list, this way:

```
wget -O - 'http://packages.coex.tech/key.asc' | apt-key add -
echo 'deb http://packages.coex.tech buster main' >> /etc/apt/sources.list
sudo apt update
```

Migration to version 0.20

Image version v0.20 includes significant changes in comparison with the version 0.19. When transitioning please note the changes presented below.

ROS package `clever` is renamed to `clover`

All the imports in Python scripts should be changed.

Before:

```
import rospy
from clever import srv
from std_srvs.srv import Trigger

rospy.init_node('flight')

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)

# Take off 1 m
navigate(x=0, y=0, z=1, frame_id='body', auto_arm=True)
```

After:

```
import rospy
from clover import srv
from std_srvs.srv import Trigger

rospy.init_node('flight')

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)

# Take off 1 m
navigate(x=0, y=0, z=1, frame_id='body', auto_arm=True)
```

systemd service `clever` is renamed to `clover`

For restarting the platform instead of:

```
sudo systemctl restart clever
```

use command:

```
sudo systemctl restart clover
```

Path to platform's files changed

The `~/catkin_ws/src/clever/` directory is renamed to `~/catkin_ws/src/clover`. Thus, configuration files (`*.launch`) are to be edited using the new path.

For example, `~/catkin_ws/src/clever/clever/launch/clever.launch` file is now `~/catkin_ws/src/clover/clover/launch/clover.launch`.

Wi-Fi network configuration

Wi-Fi networks' SSID is changed to `clover-XXXX` (where X is a random number), password is changed to `cloverwifi`.

The camera orientation configuration changed

See details in the "Camera setup" article.

Migration to version 0.22

Python 3 transition

Python 2 is deprecated since January 1st, 2020. The Clover platform moves to Python 3.

For running flight script instead of `python` command:

```
python flight.py
```

use `python3` command:

```
python3 flight.py
```

Python 3 has certain syntax differences in comparison with the old version. Instead of `print` *operator*:

```
print 'Clover is the best'  # this won't work
```

use `print` *function*:

```
print('Clover is the best')
```

The division operator operates floating points by default (instead of integer). Python 2:

```
>>> 10 / 4
2
```

Python 3:

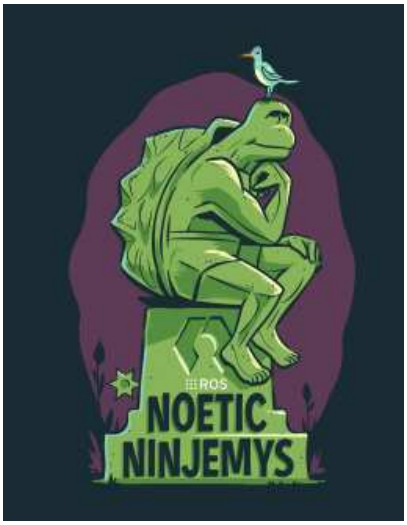
```
>>> 10 / 4
2.5
```

For strings `unicode` type is used by default (instead of `str` type).

Encoding specification (`# coding: utf8`) is not necessary any more.

More details on all the language changes see in appropriate article.

Move to ROS Noetic



ROS Melodic version was updated to ROS Noetic. See the full list of changes in the ROS official documentation.

Changes in launch-files

Configuration of ArUco-markers navigation is simplified. See details in markers navigation and markers map navigation articles.

Events

Clover is being used in a lot of educational events and competitions, such as WorldSkills, NTI Olympics, Copter Hack, Innopolis Open Robotics, etc.

This section contains articles written specifically for a particular event.

CopterHack 2023

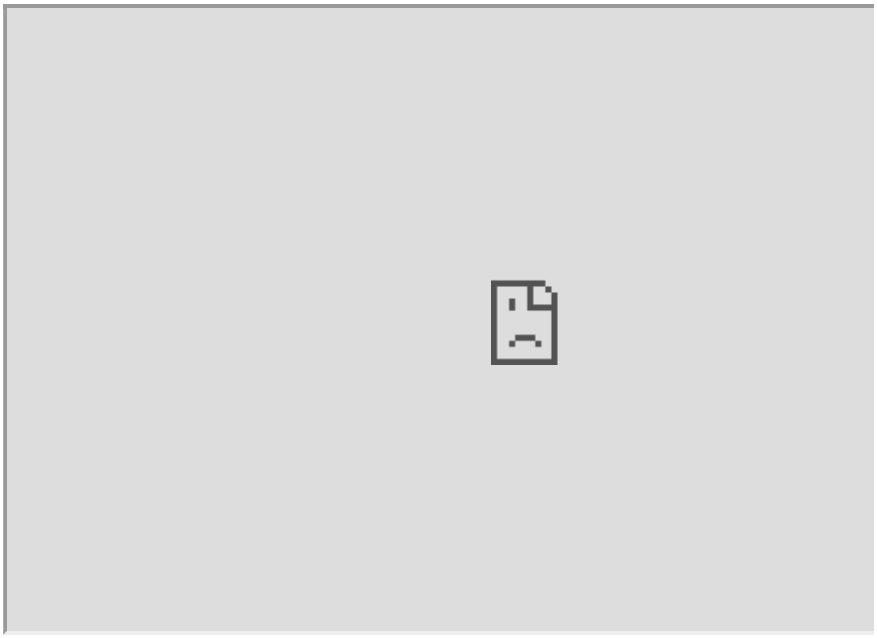
CopterHack 2023 is an international open-source projects competition on aerial robotics. The main direction of the CopterHack is team competition with a free choice of the project topic. The competition's official language is English.

To learn more about the articles of the CopterHack finalist teams follow the links CopterHack 2021, CopterHack 2022.

The proposed projects are supposed to be open-source and be compatible with the Clover quadcopter platform. Teams-participants are supposed to work on their projects throughout the competition, bringing them closer to the state of the finished product while being assisted by industry experts through lectures and regular feedback.

Final of the CopterHack 2022 was held on May 27, 2023. The winner team was the team 🇷🇺 **Clover Cloud Platform**.

Full stream of the final



Projects of the contest's participants

Place	Team	Project	Points
1	🇷🇺 Clover Cloud Team	Clover Cloud Platform	21.7
2	🇧🇾 FTL	Advanced Clover 2	21
3	🇨🇦 Clover with Motion Capture System	Clover with Motion Capture System	20.5
4	🇧🇷 Atena	Swarm in Blocks 2	20.3
5	C305	Система радио-навигации	17.5
6	🇮🇳 DJS PHOENIX	Autonomous Racing Drone	14.6
7	Lyceum №128	Network of Clover charging stations	13.7
×	🇰🇬 Zavarka	Система обмена грузами с помощью конвейера	
×	FSOTM	Drone Interceptor	
×	🇰🇬 Homelesses	Trash Collector	
×	Digital otters	Digital otters	
×	Light Flight	Сопровождение БПЛА при посадке	
×	🇰🇬 LiveSavers	LiveSavers	
×	XenCOM	Bound by fate	
×	Ava_Clover	DoubleClover	
×	TPU_1	Совместная транспортировка груза	
×	🇷🇺 TPU_2	Алгоритм полета сквозь лесную местность	

See all points by criteria in the full table.

CopterHack 2023 stages

The qualifying and project development stages will be held in an online format, however, the final round will be in a hybrid mode (offline + online). The competition involves monthly updates from the teams with regular feedback from the jury. All teams are required to prepare a final video and presentation on the project's results to participate in the final stage.

1. Qualifying stage. Applications are accepted on the deadline date until October 31, 2022.
2. Projects development stage. This stage includes monthly updates and mentorship of participants. Starting date - November 1, 2022. Deadline date - February 28, 2023.

3. All teams-participants are required to make the final video to proceed to the final round. Final videos are required to be uploaded until March 31, 2023.
4. The final round. Projects presentation takes place April 23, 2023.

Conditions and criteria for evaluation the final result

General project requirements:

1. Open-source.
2. Compatibility with the Clover platform.

Judging criteria for the jury at the final:

1. Readiness and the article (max. 10 points): the degree of readiness of the project; an accessible and understandable description of the project in the article; a link to the code with comments, diagrams, drawings. It should be possible to reproduce the project and get the result according to the article.
2. Amount of work done (max. 6 points): the amount of work done by the team in the framework within of CopterHack, its complexity, and the technical level.
3. Usefulness for Clover (max. 6 points): the relevance to the Clover and PX4 platform application in practice, the potential level of demand from other Clover users.
4. Presentation at the final (max. 3 points): quality and entertainment points of the final presentation; completeness of the project coverage; demonstration; answers to the jury's questions.

Prize fund

Basing on the results of the evaluation of projects at the final round, the jury will select the winners with the following prizes.

- 1st place: $3000 (USD).
- 2nd place: $2000 (USD).
- 3rd place: $1000 (USD).
- 4th place: $500 (USD).
- 5th place: $500 (USD).

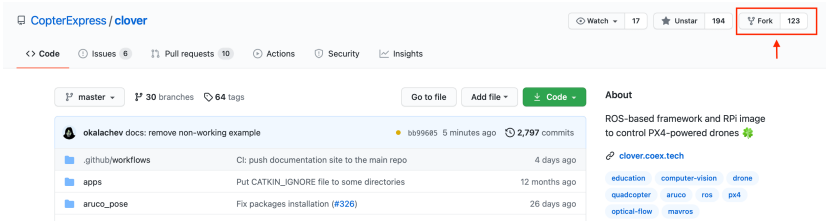
The competition partners can reward the teams according to additional criteria identified during the evaluation of projects during the final round.

How to apply?

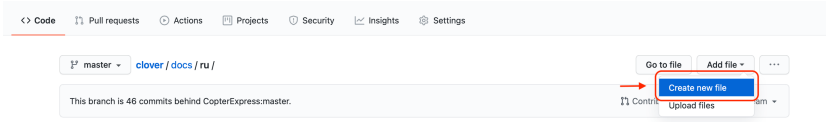
> In order to be able to apply, you must have an account on GitHub.

Prepare your application and send it as a Draft Pull Request to Clover repository

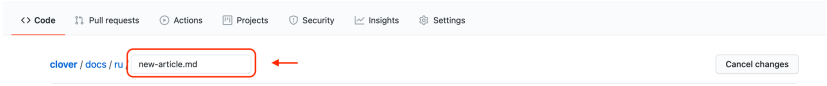
1. Fork the Clover repository:



2. On the web page of your fork, go to the `docs/en` section and create a new file in the Markdown format:



3. Enter the title of your article. For example, `new-article.md`



4. Fill in your application by the recommended template:

```
# Project name

[CopterHack-2023](copterhack2023.md), team **Team name**.

## Team information

The list of team members:

(Describe the team: full name, contacts (Telegram username), role in the t

* Alexander Sokolov, @aleksandrsokolov111, engineer.
* Elena Smirnova, @elenasmirnova111, programmer.

## Project description

### Project idea

Briefly describe the idea and stage of the project.

### The potential outcomes

Describe how you see the project result.

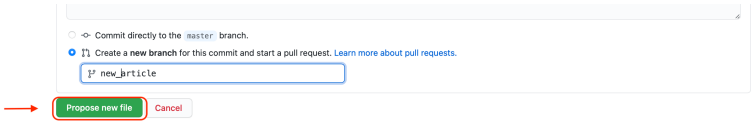
### Using Clover platform

Describe how the Clover platform will be used in your project.

### Additional information at the request of participants

For example, information about the team's experience while working on proj
```

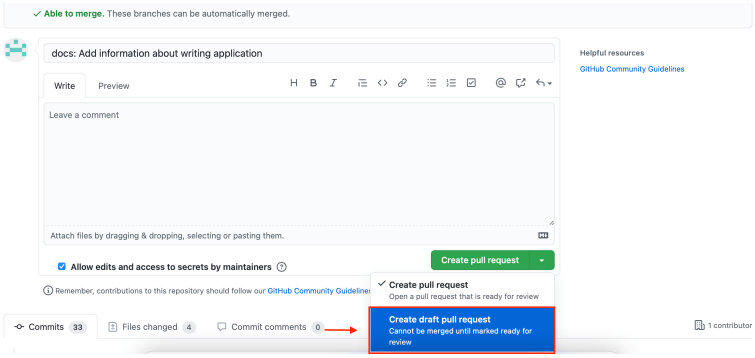
5. Go to the bottom of the page and create a new branch with the title of your article:



> Don't commit changes directly to the `master` branch, create a new branch.

6. If necessary, place additional visual assets in the `docs/assets` folder and add them to your article.

7. Send a Draft Pull Request from your branch to Clover:



8. In the Pull Request comments, you will be given feedback on the application.

9. Please note, in the *Checks* block the *Documentation* field should contain a tick, id cross appeared, click *Details* link to see the list of issues in you article found by markdownlint. If you need to change added files, edit them in you branch – changes will appear in the Pull Request automatically. **Do not open a new Pull Request for the same application**.

10. During the contest, you will work on this document, bringing it closer to the finished state. By the end of the contest you are expected to publish your article which is supposed to be the result of your work in CopterHack.

Teams-participants are supposed to be added to the special Telegram group, where one can send the project's updates and get feedback from the Jury. For all participating teams, COEX will provide a 40% discount on the Clover drone kit.

> There are no restrictions on the age, education, and number of people in a team.

CopterHack 2023 projects' papers contest

Our participants have been engaged in advanced projects in the field of aerial robotics for already two years. This year we are planning to launch a new type of contest stimulating participants to present the research results running within the whole contest, at high -level international conferences as well as to publish them in Russian and international magazines in thematic areas.

Original articles are accepted in following nominations:

- $2000 (USD) for an article in a magazine of first quartile (Q1), indexed in Scopus, Web of Science.
- $1000 (USD) for an article in a magazine, indexed in Scopus, Web of Science.
- $500 (USD) for an article, published in Compendium (Conference Proceedings), indexed in Scopus, Web of Science.

> Easy way to find quartiles for journals in Web of Science and Scopus.

Requirements:

1. The article is required to reflect the results of the project, developed within CopterHack 2023.
2. The article is required to be accepted for publication by the moment of application for the Contest.
3. It is required to indicate in the acknowledgement area that work is accomplished within the Contest.

Applications deadline: December 10, 2023. The application for the contest should be submitted through the Google Form.

Results announcement: December 24, 2023.

For all questions: CopterHack in Telegram.

> Please contact Oleg Ponfilenok in Telegram if you are interested in becoming the contest's partner or jury member.

CopterHack 2022

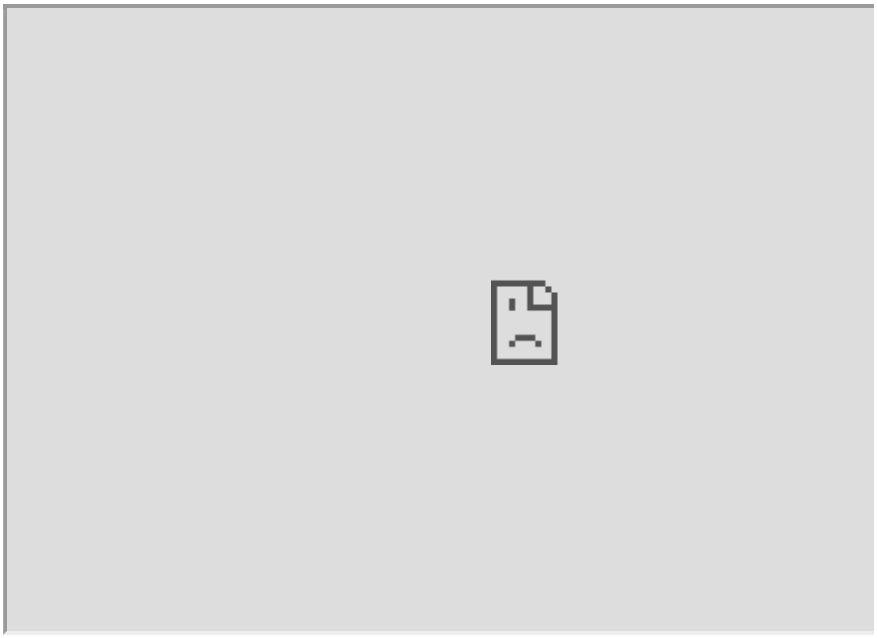
CopterHack 2022 is an international open-source projects competition on aerial robotics. The mainstream of the CopterHack 2022 is team competition with a free choice of the project topic. In addition, this year we organized a second category, company cases. The competition's main language is English.

You can see the articles of the CopterHack 2021 finalist teams by the link CopterHack 2021.

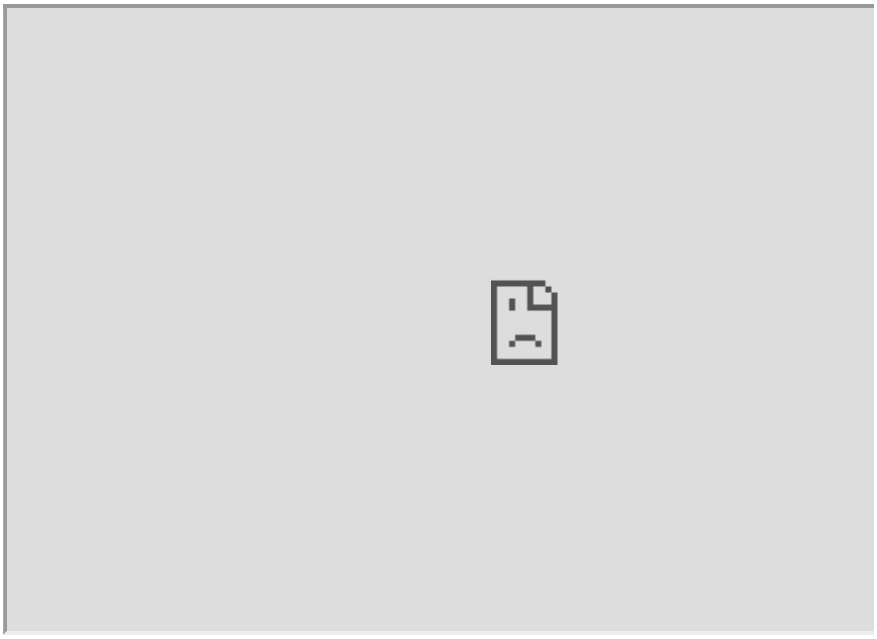
The proposed projects have to be open-source and be compatible with the Clover quadcopter platform. Teams will work on their projects throughout the competition, bringing them closer to the state of the finished product. Industry experts will assist the participants through lectures and regular feedback.

Final of the CopterHack 2022 was held on April 23, 2022. The winner team was the team 🇧🇷 **Atena - Grupo SEMEAR**.

Full stream of the final



Aftermovie



Projects of the contest's participants

Place	Team	Project	Points
1	🇧🇷 Atena - Grupo SEMEAR	Swarm in Blocks	21.6
2	🇧🇾 FTL	Advanced Clover 2	19.9
3	🇷🇺 Clover Rescue Team	Rescue Clover	17.7
4	C305	Система мониторинга воздуха	17.3
5	Space clowns	Copter For Space	16.2
6	CopterCat	CopterCat	16.1
7	Stereo	Neural obstacle avoidance	15.85
8	🇮🇳 DJS Phoenix	Autonomous valet parking drone assistance	11.7
×	R.S.	Drone Hawk	
×	🇲🇾 Moopt	IoT Water Monitoring & Optimization	
×	Дрой Ронов	Clover Swarm	
×	SPECTRE	SPECTRE	
×	🇰🇬 Alatoo University Team	Облачная платформа для симулятора Клевера	
×	Clevertron	Clevertron	
×	🇵🇱 Edgenoon	Neural and vision-based landing method	
×	🇩🇪 Inondro	Inondro Pix	
×	SolidEye	Разработка лидара без движущихся частей	
×	🇰🇬 AI_U_CLOVER	AIU_CLOVER	
×	🇻🇳 Dragon&Tanker	Dragon&Tanker	
×	V-NAV	Visual Navigation	
×	Джедаи 1581	Ретранслятор на базе Клевера	
×	Lucky flight	Swarm of Improved Clover	
×	🇺🇸 EnviroFleet	EnviroFleet	
×	Бизнес-гуси	Drone Rover Climbing System	
×	🇷🇺 fuall	Доставка дронами	

Place	Team	Project	Points
×	🇷🇺 Scout_Drone	Создание поисково-спасательного беспилотного летательного аппарата	

× – teams which haven't qualified for the Final.

See all points by criteria in the full table.

Company case competition

Teams are welcome to dive into the development of the following company cases:

1. Develop the Pixhawk FMUv6U flight controller board with the dimensions 55x40 mm and the compatibility of a Raspberry Pi CM 4 installation.
2. Cloud platform for the Clover simulator similar as to/based on ROS Development Studio.

The list of cases may be expanded in future.

CopterHack 2022 stages

The qualifying and project development stages will be held in an online format. The final round will be in a hybrid mode (offline + online). The competition involves monthly updates from the teams with regular feedback from the Jury. All teams have to prepare a final video and presentation about the project's results to participate in the final stage.

1. Qualifying stage. Applications are welcome due October 31, 2021.
2. Projects development stage. This stage includes monthly updates and mentorship of participants, starts at June 10, 2021, and continuous until February 28, 2022.
3. All participating teams should shoot the final video to proceed to the final round. Final videos have to be submitted from March 1 up to March 31, 2022.
4. The final round. Projects presentation at April 9–10, 2022.

Conditions and criteria for evaluating the final result

General project requirements:

1. Open-source.
2. Compatibility with the Clover platform.

Criteria for judging the jury at the final:

1. Readiness and the article (max. 10 points): the degree of readiness of the project; an accessible and understandable description of the project in the article; a link to the code with comments, diagrams, drawings. It should be possible to reproduce the project and get the result according to the article.

2. Amount of work done (max. 6 points): the amount of work done by the team in the framework of CopterHack 2022, its complexity, and the technical level.
3. Usefulness for Clover (max. 6 points): the relevance to the Clover and PX4 platform application in practice, the potential level of demand from other Clover users.
4. Presentation at the final (max. 3 points): quality and entertainment of the final presentation; completeness of the project coverage; demonstration; answers to the jury's questions.

Prize fund

The mainstream of the CopterHack 2022 involves the following prizes from COEX based on the results of the jury's evaluation of projects at the final round:

- 1st place: $3000 (USD).
- 2nd place: $2000 (USD).
- 3rd place: $1000 (USD).
- 4th place: $500 (USD).
- 5th place: $500 (USD).

The competition partners can reward the teams according to additional criteria identified due to the evaluation of projects during the final round.

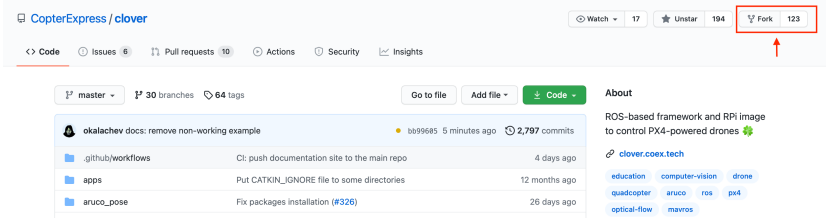
The company case competition provides a prize of $2500 (USD) from COEX for further project development for the best teams in each cases.

How to apply?

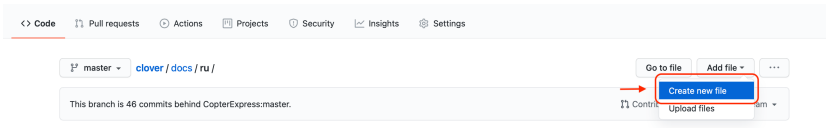
> To apply, you should have an account on GitHub.

Prepare your application and send it as a Draft Pull Request to Clover repository

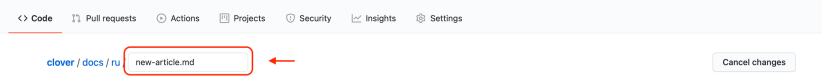
1. Fork the Clover repository:



2. On the web page of your fork, go to the `docs/en` section and create a new file in the Markdown format:



3. Enter the title of your article. For example, `new-article.md`



4. Fill out your application by the recommended template:

```
# Project name

[CopterHack-2022](copterhack2022.md), team **Team name**.

## Team information

The list of team members:

(Describe the team: full name, contacts (e-mail/Telegram username), role i

* Alexander Sokolov, @aleksandrsokolov111, engineer.
* Elena Smirnova, @elenasmirnova111, programmer.

## Project description

### Project idea

Briefly describe the idea and stage of the project.

### The potential outcomes

Describe how you see the project result.

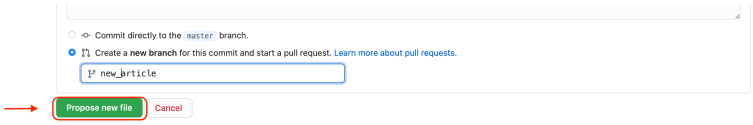
### Using Clover platform

Describe how the Clover platform will be used in your project.

### Additional information at the request of participants

For example, information about the team's experience working on projects,
```

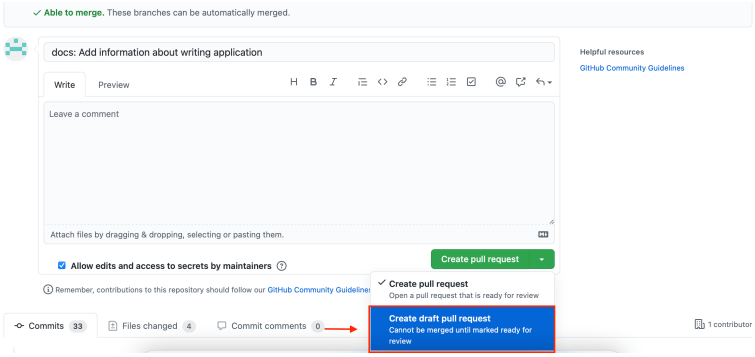
5. Go to the bottom of the page and create a new branch with the title of your article:



> Don't commit changes directly to the `master` branch, create a new branch.

6. If necessary, place additional visual assets in the `docs/assets` folder and add them to your article.

7. Send a Draft Pull Request from your branch to Clover:



8. In the Pull Request comments, you will be given feedback on the application. On the contest page, in the section "Projects of the contest participants", a link to your application in your fork will be published.

9. Note the *Checks* block at the bottom, a check mark should appear in the *Documentation* field. If a cross appeared, click *Details* link to see the list of issues in you article found by markdownlint. If you need to change added files, edit them in you branch – changes will appear in the Pull Request automatically. **Do not open a new Pull Request for the same application**.

10. During the contest, you will work on this document, bringing it closer to the state of the finished article. By the end of the contest, you will publish your article, which will be the result of your work in CopterHack 2022.

As soon as the link to the application is added to this page in the section "Projects of the contest's participants", your team has become an official participant of the CopterHack 2022!

Contest participants will be added to the special Telegram group, where one can send the project's updates and get feedback from the Jury. For all participating teams, COEX will provide a 50% discount on the Clover drone kit.

> There are no restrictions on the age, education, and number of people in the team.

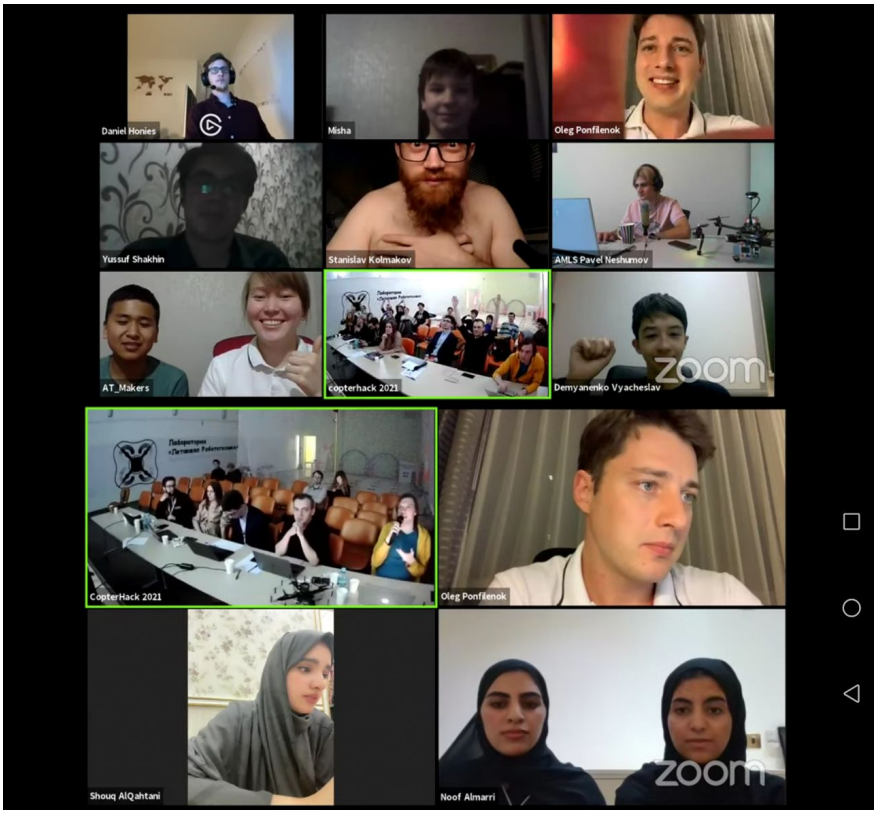
For all questions: CopterHack 2022.

CopterHack 2021

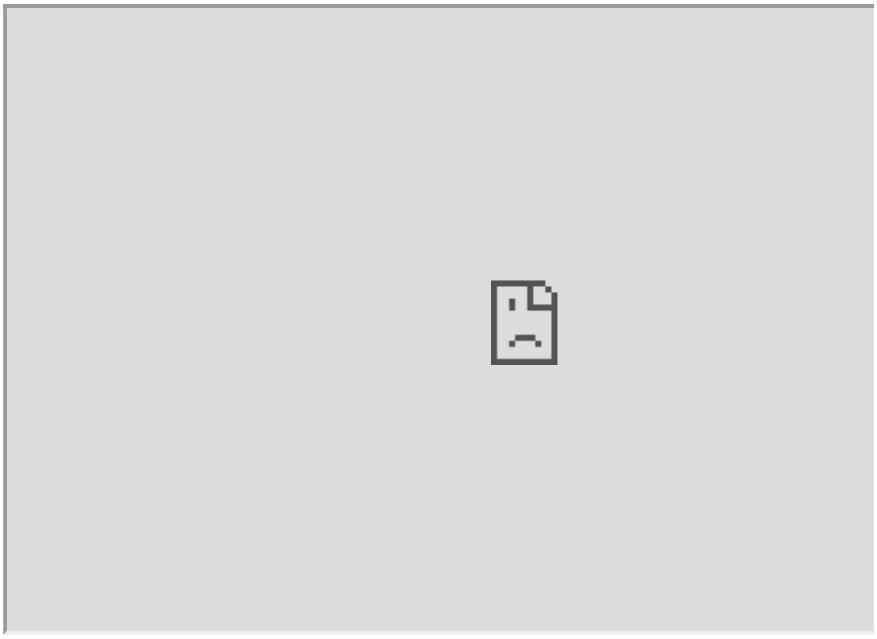
CopterHack 2021 is a team competition for the development of open source projects for the Clover quadcopter platform. Fifty-four teams from 12 countries took part in the competition.

All information about the event can be found on the official website: https://coex.tech/copterhack.

Full stream of the final: https://www.youtube.com/watch?v=Z06vxuAHmuE.



Aftermovie



Participating teams articles

Place	Team	Project	Points
1	🇧🇾 FTL	AdvancedClover	18.8
2	🇷🇺 EasyToFly	EasyToFly	18.5
3	🇩🇪 ADDI	3D-printed generative design frame	17.8
4	🇰🇬 AT Makers	D-drone Graffiti-copter	16.7
5	🇶🇦 DroMap	The Indoor Mapping Drone	16.5
6	MINIONS	Seed spreading quadcopter	15.5
7	🇷🇺 Hardaton	Хардатон Квиддич	15.48
8	Atomic Ferrets	Система засечки для дронов	15
9	Drones to fight Corona	Drones to fight Corona	14.6
10	AMLS	Autonomous Multirotor Landing System	12.8
11	PaD30DЖ	Октокоптер со специфичным расположением пропеллеров	11.6
12	Zaural Viking	Программируемый летающий автомобиль	11.4
13	🇨🇦 Bennie and the Jetson TX2	Retail Drone	9.8
14	🇳🇱 Blue Jay Eindhoven	Designing a drone and a path planning algorithm	9.6
15	ProCleVeR	Разработка системы для управления БПЛА с помощью шлема виртуальной реальности	8.5
16	🇷🇺 Quadrotor	Дрон-Агроном	7.7

See points by criteria in the full table.

Copter Hack 2019

The Copter Hack 2019 hackathon took place on the 11th to 13th of October in the "Moscow" Technopolis.

Event page: https://coex.tech/copterhack.

Hackathon chat: https://t.me/CopterHack.

Timepad event page: https://copterexpress.timepad.ru/event/1017592/.

Information for participants

COEX Pix specifics

Be sure to set the *Autopilot orientation* parameter to `ROTATION_ROLL_180_YAW_90` if you're using the *COEX Pix* flight controller. This parameter should be applied during calibration of each sensor.



This parameter is used for *IMU* orientation correction.

Suggested image versions

Raspberry Pi versions 3B+ and lower: v0.18

Raspberry Pi version 4: v0.19-alpha.1

Camera orientation

Some drones have the camera mounted with the cable going forward. You should set this orientation in the `main_camera.launch` file in the `clever` package.

Further reading: Camera orientation

Using Optical Flow

In order to enable optical flow set `optical_flow` and `rangefinder_vl53l1x` parameters to `true` in `clever.launch`.

Enable `pub agl as lpos down` in `LPE_FUSION` parameter using QGroundControl.

Make sure the rangefinder is mounted correctly and is working (see Interfacing with a laser rangefinder).

Further reading: Optical Flow.

Using ArUco map

Use the `cmit.txt` map. See instructions.

Drone batteries

The battery indicator should be connected to the battery at all times. The organizers will not replace your damaged batteries!

Flight videos

Be sure to record **ALL** your flights on video! If your drone fails before your presentation, you'll be able to at least show your videos.

Yaw problem

The v1.8.2-clever.7 FCU firmware has a potential bug that manifests during VPE (marker-based) flights. If your drone does not correct its yaw when using ArUco markers, try using an older firmware version (v1.8.2-clever.6, available from https://github.com/CopterExpress/Firmware/releases/tag/v1.8.2-clever.6). Download `px4fmu-v4_default.px4` for COEX Pix.

`navigate` service problem

The 0.18 Raspberry Pi image has a potential bug that makes the drone fly through waypoints too fast. Try setting the `nav_from_sp` parameter to `false` in `~/catkin_ws/src/clever/clever/launch/clever.launch` if you are affected by it:

```
<!-- simplified offboard control -->
<node name="simple_offboard" pkg="clever" type="simple_offboard" output="screen
    <param name="reference_frames/body" value="map"/>
    <param name="reference_frames/base_link" value="map"/>
    <param name="reference_frames/navigate_target" value="map"/>
    <param name="reference_frames/navigate_target" value="map"/>
    <param name="nav_from_sp" value="false"/>
</node>
```

Lectures (in Russian)

Lecture 1: Introduction – https://www.youtube.com/watch?v=cjtmZNuq7z0.

Lecture 2: FCU setup – https://www.youtube.com/watch?v=PJNDYFPZQms.

Lecture 3: PX4 architecture – https://www.youtube.com/watch?v=_jl7FImq3jk.

Lecture 4: Autonomous flights – https://www.youtube.com/watch?v=ThXiNG1IzvI.

Be sure to check out other videos on the COEX YouTube channel: https://www.youtube.com/channel/UCeCu93sLBkcgbIkIC7Jaauw/featured.

Results

Winners:

1. Bulbolet – potato delivery using a smart hoist.
2. Copter don't hurt me – controlling drone using a neural interface.
3. import torch – active track using neural networks.
4. Autobot – freeze light through a VK bot.
5. Stardust Crusaders – AR drone simulation.

Copter Hack 2018

Hackathon Copter Hack 2018 was held on October 19 – 21 at the Moscow Technopolis.

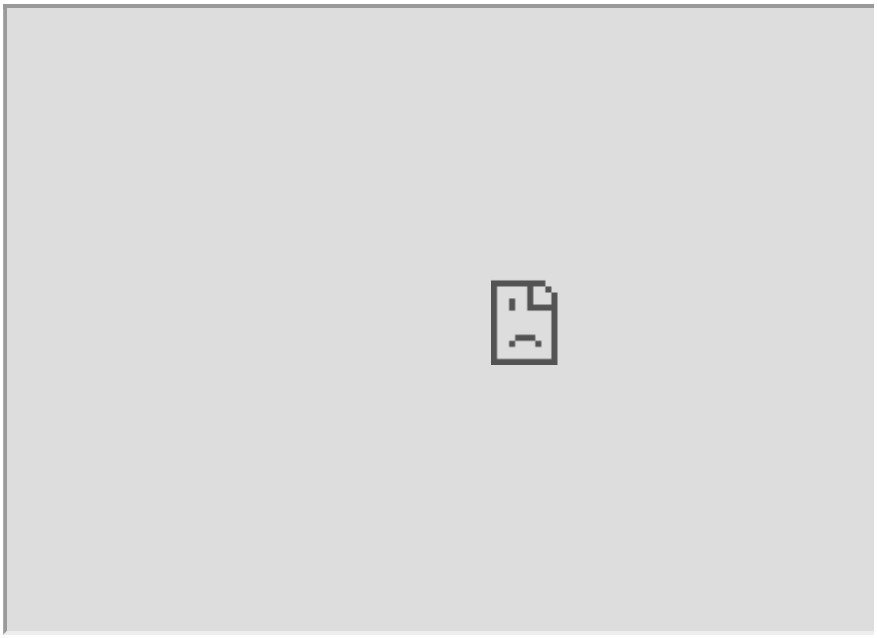


Hackathon chat: https://t.me/CopterHack.

Hackathon stream: https://www.youtube.com/watch?v=nIo5HSqlt6I.

Hackathon photos: https://drive.google.com/open?id=1ozdXol4rhKwhHbsrnfxrp3CqazBRm-3W.

Videos



Lectures

Lecture 1: assembly — https://www.youtube.com/watch?v=gEs-w7BRPM8.

Lecture 2: setup — https://www.youtube.com/watch?v=sPqSCCmgdG0.

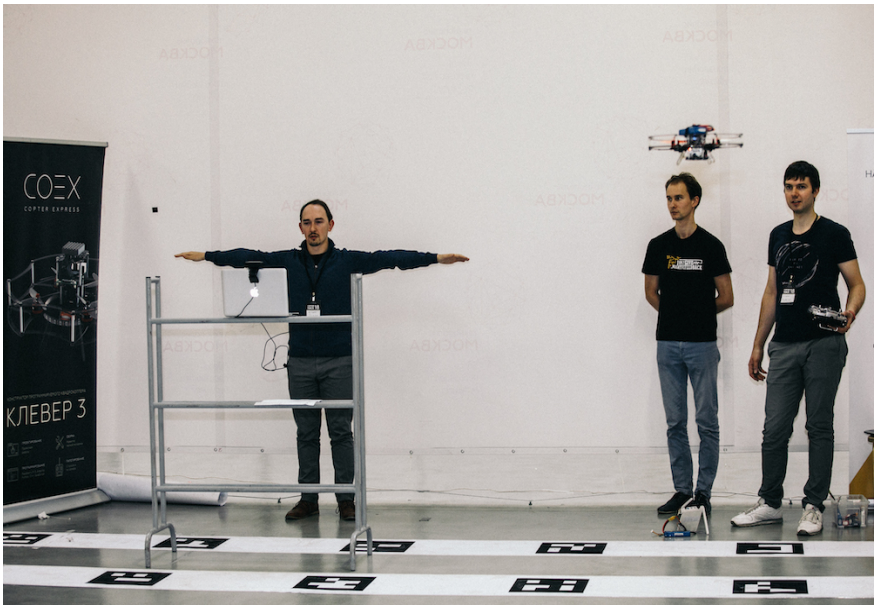
Lecture 3: PX4 firmware — https://www.youtube.com/watch?v=WFnZAIypgMQ.

Lecture 4: Autonomous flights — https://www.youtube.com/watch?v=gD6a7aSEf9M.

Results

Winning teams:

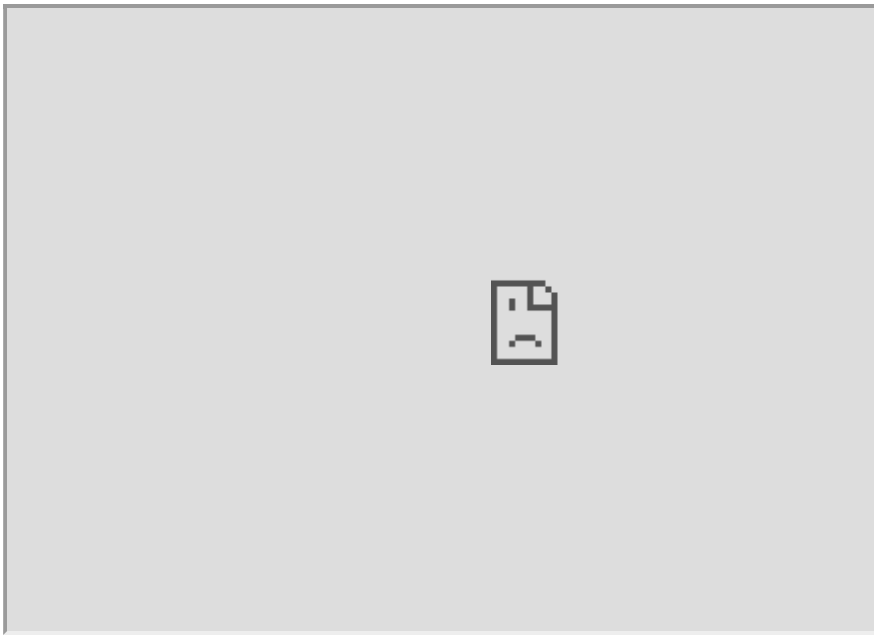
1. Starshine (Moscow) — controlling the drone using a "smart" glove.
2. Alcopter (Moscow) — controlling the drone with gestures and pose change.
3. Merry copter (Samara) — a Vkontakte bot for controlling the copter, a joint flight of "Zhuzha" and "Clover 3".
4. International Post (Novosibirsk) — automatic scattering leaflets from the drone.
5. LAMAR (Yekaterinburg) — an automatic quadcopter battery replacement station.


COEX
COPTER EXPRESS
КЛЕВЕР 3

Copter Hack 2017

On July 28 – 30, 2017, Copter Express held a hackathon named "Copter Hack 2017", where the objective was to program a Clover to dance-fly autonomously to random music.

The team "Pangolins" became the winners.



Video lectures are available at https://copterexpress.timepad.ru/event/510375/.

Modules

- Navigation in the marker field and simplified copter control: https://github.com/CopterExpress/marker_navigator (installed on the flash drive)
- ROS module for communication with the music server https://github.com/CopterExpress/copter_hack_music (installed on a flash drive)
- the source of the music server https://github.com/CopterExpress/copter_hack_music_server

Viewing video from the camera

To run on Raspberry:

```
rosrun web_video_server web_video_server
```

In the browser, open webpage `http://<ip raspberry>:8080`.

Attention: Video stream distribution greatly reduces the performance of marker recognition for the flight.

SSID Wi-Fi

To change the SSID of distributed Wi-Fi you should change the SSID parameter in any way in file `/etc/hostapd/hostapd.conf` .

The list of recognized markers

```
rostopic echo /marker_data
```

Helpful articles

- Copter setup
- Flight modes
- MAVRos package
- A good introductory article: https://habrahabr.ru/post/227425/
- Signals used in drones: https://geektimes.ru/post/258186/
- A good article about PIDs: https://habrahabr.ru/company/technoworks/blog/216437/
- Video lectures are available at https://copterexpress.timepad.ru/event/510375/.
- Aubio, a library for sound (music) analysis
- Packages for Python for working with music: https://wiki.python.org/moin/PythonInMusic

Contest for the best educational video on assembly and configuration

Requirements:

- the video contains the entire process of assembling and configuring the Clover 4.2 drone kit: from opening the box with components to flying the copter in Position mode using ArUco markers;
- the video is uploaded to YouTube and is public accessible;
- the video contains voice-over in English;
- the video lasts from 6 to 60 minutes.

Dates of the contest: February 12 – December 13, 2021.

Prizes

- 🥇 1st place: $500 (USD).
- 🥈 2nd place: $300 (USD).
- 🥉 3rd place: $200 (USD).

Results

Place	Participant	Link to the video
1	🇷🇺 Philipp Batalin	https://www.youtube.com/watch?v=f0rpdulOSEk
2	🇮🇹 Sara Pettinari	https://www.youtube.com/watch?v=PxxfyVH6RRA
3	🇲🇾 Kai Feng Chew	https://www.youtube.com/watch?v=skgSwFle6Ms
3	🇰🇿 Nikita Lobanov	https://www.youtube.com/watch?v=93b1epEM3SQ

Educational contests

1. Contest for the best educational lecture

The Copter Express company organizes a contest for the best educational lecture with COEX Clover 4 quadcopter kit application.

The main goal of the contest is aerial robotics popularization and community development.

Lecture requirements

- The topic of the lecture is of free choice. Programmable quadcopter kit COEX Clover 4 and/or The Clover simulation environment should be used as the main tool in the lecture.

  > *The version of COEX Clover is not earlier than version 4. The virtual machine image is not earlier than version 1.0.

- The video is uploaded on YouTube or another public platform and is public accessible.
- The language of the lecture is any. The video contains subtitles in English in case the language is made neither of English nor Russian.
- The duration of the lecture is limited from 15 min. to 3 hours.

Requirements for the participants

- The participant must be the author of the lesson.
- Third parties can provide technical support for recording a lecture.
- The status of the participant is unlimited (student, representative of a general education institution, representative of the industry, amateur).

Applications deadline: November 30, 2022.

How to apply?

The application to the contest is performed via the Google Form where the link to the video lecture should be attached.

Participants who are the authors of the lecture are allowed to participate in the competition.

Prizes

Based on the results of the submitted application, the jury selects the winners of the competition. The quality of the video, it is content, and audience engagement are assessed.

- 1st place: $500.
- 2nd place: $400.
- 3rd place: $300.
- 4th place: $200.
- 5th place: $100.

2. Contest for the best school lesson

The Copter Express company organizes a contest for the best school lesson with COEX Clover 4 quadcopter kit application.

The main goal of the contest is aerial robotics popularization and community development.

Lesson requirements

- Programmable quadcopter kit COEX Clover 4 should be used as the main tool for the lesson.
  > *The version of COEX Clover is not earlier than version 4.
- Integration of the quadcopter into any of the general education disciplines (physics, mathematics, computer science, etc.).
- Practical use of the main tool in the lesson.
- Grade - no restrictions (primary, high school).
- Lesson duration is 30-45 minutes.
- Lesson format - offline.
- The video of the lesson was filmed in the classroom of a general education institution.

Requirements for the participants

- The participant must be the author of the lesson.
- The participant must be a teacher of a general education institution

How to apply?

The application to the contest is performed via the Google Form.

Applications deadline: November 30, 2022.

Prizes

Based on the results of the submitted application, the jury selects the winners of the competition. The video and material quality are assessed.

- 1st place: $500.
- 2nd place: $400.
- 3rd place: $300.
- 4th place: $200.
- 5th place: $100.

3. Contest for the best online course

The Copter Express company organizes a contest for the best online course with COEX Clover 4 quadcopter kit application.

The main goal of the contest is aerial robotics popularization and community development.

The course is evaluated according to a separate, publicly available lesson submitted for the contest.

Course requirements

- The course is related to the direction of Aerial robotics.
- Programmable quadcopter kit COEX Clover 4 and/or The Clover simulation environment should be used as the main tool in the course;

  > *The version of COEX Clover is not earlier than version 4. The virtual machine image is not earlier than version 1.0.

- The course is located on a public platform (e.g., Coursera).
- The course can be either paid or free of charge. One public lesson from the course is submitted for the competition;
- The lesson submitted for the contest should be publicly accessible.
- The language of the lesson is any. The video contains subtitles in English in case the language is made neither of English nor Russian (if there is a video in the lesson).
- The duration of the course and lesson is not limited.

Requirements for the participants

- The participant must be the author of the course.
- Third parties can provide technical support for preparing a course.
- The status of the participant is unlimited (student, representative of a general education institution, representative of the industry, amateur).

How to apply?

The application to the contest is performed via the Google Form where the link to the video course should be attached.

Applications deadline: November 30, 2022.

Prizes

Based on the results of the submitted application, the members of the Commission select the winners of the competition. The quality of the material, the format of the presentation of the material, the total volume and content of the course are assessed.

- 1st place: $1000.
- 2nd place: $800.

- 3rd place: $600.
- 4th place: $400.
- 5th place: $200.

Clover-based projects

Clover drone kit is widely used in design activities. This section contains user articles describing the implemented projects.

Clover Cloud Platform

CopterHack-2023, team **Clover Cloud Team**.

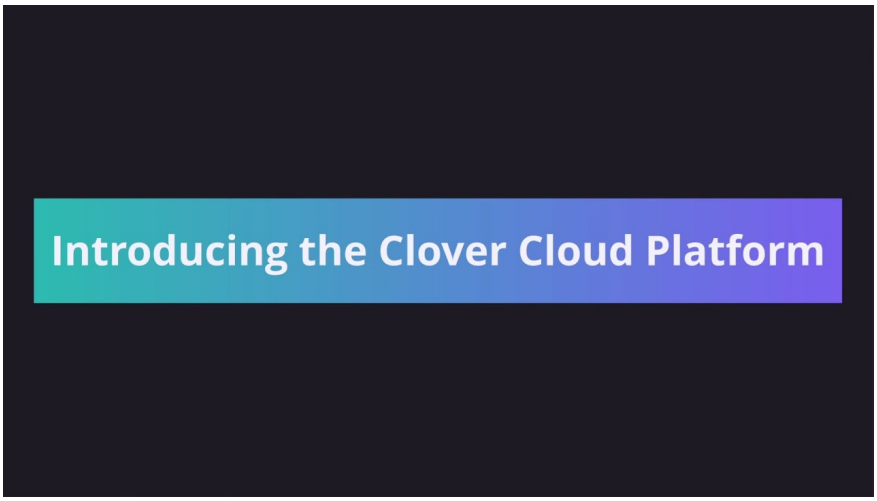
The list of our team members:

- Кирилл Лещинский / Kirill Leshchinskiy, @k_leshchinskiy - Team Lead.
- Кузнецов Михаил / Mikhail Kuznetsov, @bruhfloppa - Frontend Developer.
- Даниил Валишин / Daniil Valishin, @Astel_1 - Backend Developer.

Table of contents

- Introduction
- Usability
- How to work with our platform?
- About the development of the platform
- Conclusion

Video demonstration



Introduction

Clover Cloud Platform is an innovative platform that enables users to access COEX Clover drone simulation online, without the need to download any programs or virtual machines.

> Visit our documentation to learn all about the platform, its development and how to use it.

Unleash Your Coding Power: Develop Autonomous Flight Code at Lightning Speed on Clover Cloud Platform

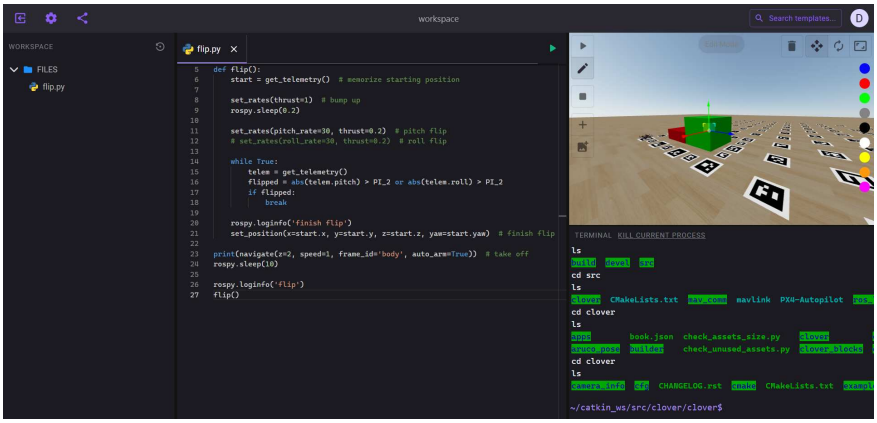
If you're a developer working on autonomous flight projects, you know how time-consuming and distracting all of the routine activities can be. Between managing your hardware, debugging, and configuring your environment, it can feel like the real work of coding gets lost in the shuffle.

That's where our platform comes in. Our streamlined interface and powerful tools make it easy to tackle all of those essential tasks so you can focus on what really matters: developing flawless, high-performance code for your autonomous flight project.

So why wait to unleash your coding power? Sign up for our platform today and discover the difference it can make in the speed, quality, and focus of your autonomous flight coding work.

Usability

Our platform is incredibly user-friendly and provides seamless access to the simulation in just a few clicks. Together with a simulator that displays simulation data accurately and without delay, there is a map editor allows users to edit the ArUco marker map and add or modify other objects on the scene directly within the simulation window. Additionally, users can create pre-configured workspaces complete with autonomous flight code and simulation scene configuration. Each user can also create their templates or apply a pre-made one to their workspace in just a few clicks. In addition to its other features, Clover Cloud Platform provides users with a convenient code editor for autonomous flight coding. Users can write code in the built-in editor and run it directly from the editor, viewing program output in real-time in the terminal. The platform also includes a file manager that simplifies file manipulation tasks, further enhancing the user's overall experience. With these tools at your fingertips, Clover Cloud Platform delivers an unparalleled level of accessibility and convenience for autonomous flight simulation.



The CodeSandbox for COEX Clover

You can describe the usability and relevance of our platform in another way. Have you heard of CodeSandbox? Our platform offers the same convenience, flexibility, and accessibility as CodeSandbox, but is specifically designed to work with the COEX Clover drone simulation.

How to work with our platform?

Let's dive into the sea of functionality that our platform offers. Detailed description of each feature is available in our documentation, here we will provide a general overview of the platform.

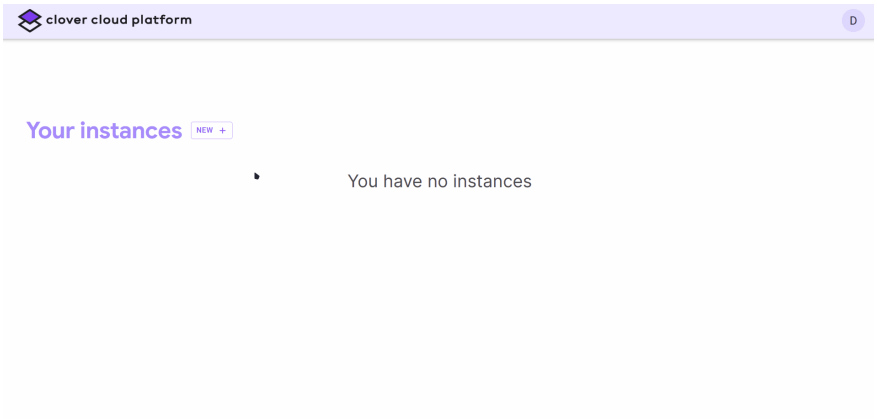
Creating an account

First, you should create an account on our site. You can do this by clicking on this link.

Instance management

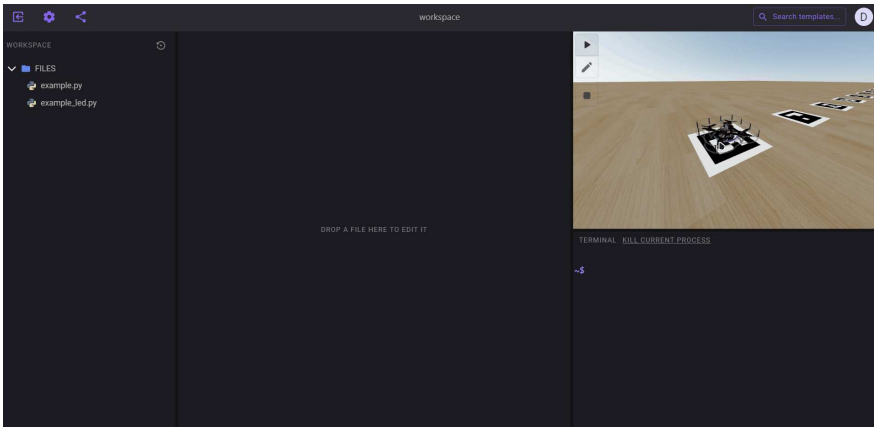
After creating an account, you will be taken to the dashboard. Here you can create, start, stop and delete workspaces.

> Workspaces are containers with Gazebo simulator and our software that provide data flow for simulation visualization, as well as handle requests from file manager, code editor and terminal.



Workspace overview

In the workspace, in addition to the simulator, you have a file manager, code editor and terminal. There is also an editing mode in the simulator - one of the key features of our project. It allows you to quickly and conveniently edit the simulation scene, namely: move ArUco markers, change their size, change id of the marker, load instead of marker picture, add new markers or delete them. You can also add 3d objects to the scene and change their position, size and color. Below is an example of working with our workspace.



Templates

Templates are another key feature of our platform.Is there something you can't do and you want to see how to properly perform a task? Look for the right template with ready-made code in the Template Browser and apply it to your workspace! Each user can create a template with an autonomous flight code and simulator configuration and share it.

About the development of the platform

Our team has worked tirelessly to develop a simple yet multifunctional platform. We utilized the most modern standards and tools and implemented numerous optimization methods to ensure seamless performance and error-free operation. The frontend programming language chosen was JavaScript with the React framework, as a design system we utilizing Material Design style for an elegant and intuitive user interface. With the help of GitHub Actions the website is being built and deployed to Firebase hosting. The platform's backend is written in Python and contains multiple simultaneously running scripts. User data is secured and stored in a MongoDB database. Communication between the server and site is enabled through web sockets and the socket.io library, guaranteeing lightning-fast data transfer with minimal lag.

You can view the source code of our platform by clicking on the links below:

Repository with the frontend-side code

Repository with the backend-side code

Conclusion

In conclusion, we have successfully created a truly convenient and useful platform, suitable for both novice and advanced COEX Clover drone users. Beginners can test their first autonomous flight code without the need for demanding simulator installation or virtual machines. They can also explore all of the drone's functions and capabilities without editing any configuration files. Advanced users benefit from access to their workspace from anywhere in the world and on any device, along with a convenient code-sharing system. In the

future, we plan to add more new features to our platform, scale our network to serve a greater number of users, and collaborate with COEX to integrate their Clover quadcopter documentation into our platform, offering users a very simple and user-friendly way to learn to program autonomous drone flight. We also want to express gratitude to the COEX customer support team for their assistance in resolving complex issues that arose during development.

Autonomous Racing Drone: CHETAK

CopterHack-2023, team **DJS PHOENIX**.

Team Information



We are the DJS Phoenix, the official drone team of Dwarkadas. J. Sanghvi College of Engineering

The list of team members:

- Shubham Mehta, @Just_me_05, Mentor.
- Harshal Warde, @kryptonisinert, Mechanical.
- Parth Sawjiyani, @Non_Active, Mechanical.
- Soham Dalvi, @devilsfootprint_1973, Mechanical.
- Vedant Patel, @VedantMP, Mechanical.
- Harsh Shah, @harsssshhhhh, Mechanical.
- Lisha Mehta, @lishamehta, Mechanical.
- Shubh Pokarne, @Shubhpokarne, Electronics.
- Tushar Nagda, @tushar_n11, Electronics.
- Deep Tank, @Kraven, Electronics.
- Khushi Sanghvi, @Cryptoknigghtt, Programmer.
- Harshil Shah, @divine_fossil, Programmer.

- Omkar Parab, @Omkar_parab21, Programmer.
- Madhura Korgaonkar, @Madhura221, Programmer.
- Shruti Shah, @Shrutishah22, Programmer.
- Aditi Dubey, @aditi_0503, Marketing.
- Krisha Lakhani, @krishalakhani, Marketing.

Project Description

This year, our team DJS Phoenix, presents to you a fully Autonomous Racing Drone. The drone scans for ArUco tags on the gates and passes through them.

Project Idea

This project proposes to develop an autonomous racing drone that can navigate through complex courses at high speeds while avoiding obstacles and detecting changes in the environment. In racing competitions, autonomous drones can compete in high-speed, precision races that challenge their agility, speed, and accuracy. These competitions could be held in indoor arenas or outdoor tracks, and they could attract enthusiasts and spectators from all over the world. With their advanced capabilities, autonomous racing drones could usher in a new era of racing events that are more exciting and challenging than ever before. From racing competitions to search and rescue operations, the autonomous racing drone can be used in a wide range of applications that benefit individuals, businesses, and society as a whole.

Potential Outcome

Problem

In many industries and applications, there is a need for fast, efficient, and safe movement of goods and information. Drones have become an increasingly popular tool for a wide range of applications, from aerial photography to surveying and monitoring. However, operating a drone requires a certain level of skill and experience, which can be a barrier for individuals or businesses who want to take advantage of this technology. Additionally, traditional drones can be expensive and time-consuming to operate, limiting their accessibility and effectiveness. Therefore, there is a need for a more user-friendly and affordable solution that can expand the use of drones to new audiences and applications.

Solution

The solution to the above problem statement is an autonomous racing drone. An autonomous racing drone is equipped with a camera that scans the ArUco tags for gate detection which is supported by software used in autonomous flights that enable it to navigate through a predetermined course while avoiding obstacles and achieving high speeds. Unlike traditional drones, an autonomous racing drone does not require manual control, making it an ideal solution for those who

do not have the skills or experience to operate a drone.Its autonomous capabilities make it a more accessible and user-friendly solution than traditional drones, enabling individuals or businesses to take advantage of this technology without requiring extensive training or expertise.



Additional Information

In 2017, a student committee for DJS Phoenix was formed. In India, our team has participated in a number of contests, including IDRL-IIT GandhiNagar (sixth rank), IDRL-SVPCET Nagpur(second rank) and TECHNOXIAN (second place out of 50 national teams). In CopterHack-2021, our team participated, and we placed eighth internationally. We are back with improved concepts after learning from the previous season.

For more information checkout gitbook: https://djs-phoenix.gitbook.io/chetak-faster-than-you-can-imagine/.

Project Video

CopterHack-2023, team **Clover with Motion Capture System**. Click logo for project video.



Table of Contents

- Team Information
- Educational Document
- Introduction
- Project Description
- Hardware
- Data Transfer
- Examples
- Trajectory Tracking
- Auto-Tuning
- Conclusion

Team Information

The list of team members:

- Sean Smith, @ssmith_81, roboticist and developer: GitHub, Linkedin.

Educational Document

My Gitbook, with detailed step by step analysis of the proposed project during the CopterHack 2023 competition can be found: MoCap Clover Gitbook.

This page gives a broad overview on the motivation and purpose behind this project, it also provides research and industry based knowledge around UAV application that the reader may find interesting. If the user is interested in the technical details and implementation then refer to the educational Gitbook document.

Introduction

Aerial robotics has become a common focus in research and industry over the past few decades. Many technical developments in research require a controlled test environment to isolate certain characteristics of the system for analysis. This typically takes place indoors to eliminate unwanted disturbances allowing results to be more predictable. Removing localization and pose feedback concerns can be accomplished with motion capture (MoCap) systems that track unmanned aerial vehicles (UAVs) pose with high precision as stated:

"OptiTrack's drone and ground robot tracking systems consistently produce positional error less than 0.3mm and rotational error less than 0.05°" [reference].

This enables researchers to study the dynamics and behavior of UAVs in different environments, evaluate their performance, and develop advanced control algorithms for improved flight stability, autonomy, and safety. Research facilities around the world tend to built research drones from the ground up using off-the-shelf components with open source platforms such as PX4. While the end goal is the same: transferring pose feedback to the flight controller along with high level commands, the platforms and methods can vary significantly depending on factors such as onboard and offboard computing frameworks and data transfer methods. Many developers have a detailed background and understanding of the theoretical components of their research, however, adapting hardware configurations to their own platform such as sensor feedback and sensor fusion is not obvious. The purpose of this project is to provide detailed documentation on integrating the Clover platform with the MoCap system along with examples to familiarize users with the hardware, sensor fusion, high and low level controller development, and trajectory tracking.

Project Description

In this article, we will provide an overview of MoCap systems for tracking UAV pose in research applications, highlighting their significance, advantages, and potential impacts in the field of UAV controller development.

Document structure

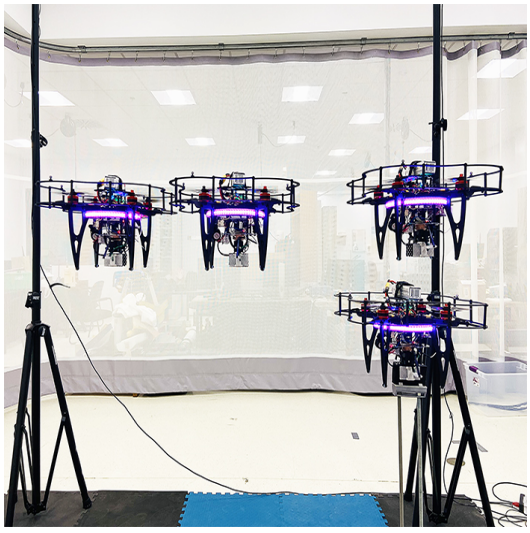
The Motion Capture System educational document is divided into three main sections outside of the Introduction and Conclusion. Each section and its purpose is listed:

Hardware

The main goal in this section is to educate the reader on the MoCap system hardware and software. This can be further divided into several steps including camera placement, marker placement, and system calibration. A summary of the process is provided:

Task	Description
Camera Placement	Position the motion capture cameras in strategic locations around the area where the UAV will be flying. The number of cameras and their placement will depend on the size of the area and the desired capture volume. Typically, cameras are placed on tripods or mounted on walls or ceilings at specific heights and angles to capture the UAV's movements from different perspectives. **A simple 4-camera setup example is provided in the educational document**.
Marker Placement	Attach OptiTrack markers to the UAV in specific locations. OptiTrack markers are small reflective spheres that are used as reference points for the motion capture system to track the UAV's position and movements. **An example placement on the Clover is shown in the educational document**.
System Calibration	Perform system calibration to establish the spatial relationship between the cameras and the markers. This involves capturing a calibration sequence, during which a known pattern or object is moved in the capture volume. The system uses this data to calculate the precise positions and orientations of the cameras and markers in 3D space, which is crucial for accurate motion capture.

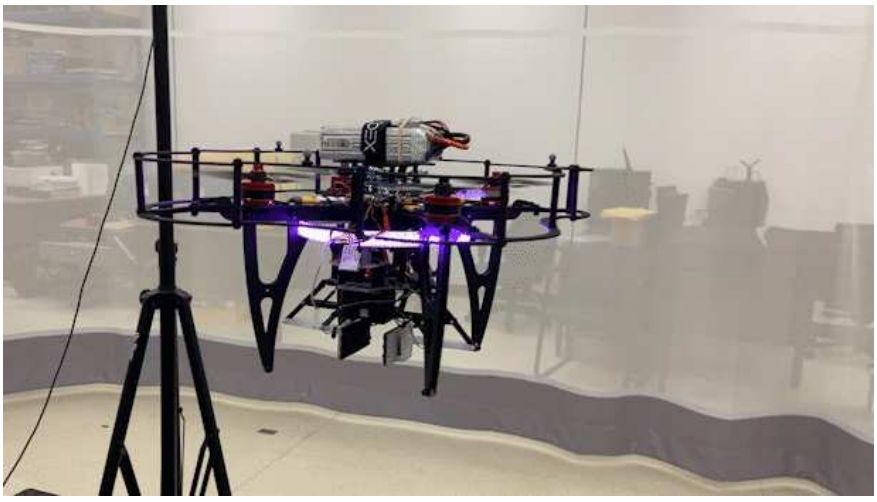
With these components completed correctly, you are well on your way to commanding indoor autonomous missions like this:



Overall, configuring a motion capture system for UAV research requires careful planning, precise marker placement, accurate system calibration, and thorough validation to ensure accurate and reliable data collection for your research purposes. For more information, refer to the informative documentation.

Data Transfer

With the data acquired from the MoCap system, the main goal in this section is to transfer it to the Raspberry Pi onboard the Clover and remap it to the flight controller/PX4 for control. A summary of the steps are listed:



- Data Acquisition: The motion capture system continuously tracks the position and orientation (pose) of the UAV using markers attached to the UAV and cameras positioned in the capture volume. The system calculates the 3D pose of the UAV in real-time and can be viewed through the motive software.
- Data Transmission: The pose data is transmitted from the motion capture system to a Raspberry Pi using VRPN and a ROS network. While this works, I have implemented a strictly UDP data transmission method where highlighting the setup process and ease of use will be a future development, both configurations can be seen in the below figures. The Raspberry Pi acts as an intermediary for processing and relaying the data to the flight controller onboard the UAV using MAVROS. The connection can be established using USB or UART, I chose UART in my setups.

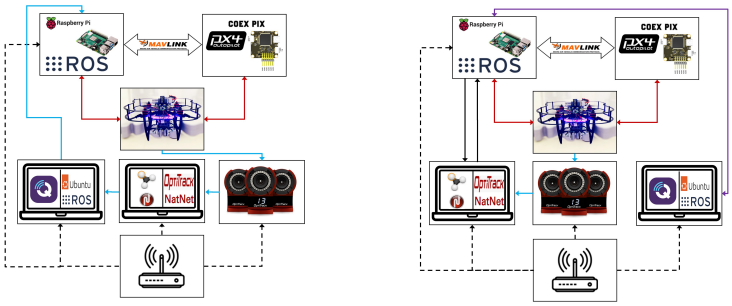


Fig.1(a) - Left figure: ROS network experimental setup topology. Legend: Black dotted line is the provided local network; Blue solid line is the Clover pose transmission where the final transmission from laptop to Pi is over a ROS network; Red line is hardware connections; MAVLink arrow is

communication via a MAVLink protocol. .

Fig.1(b) - Right figure: UDP transmission experimental setup topology. Legend: Black dotted line is the provided local network; Black solid line is the UDP client-server drone pose transmission; Light blue line is the pose data transmission; Red line is hardware connections; Purple line is communication via secure shell protocol and ROS network communication; MAVLink arrow is communication via a MAVLink protocol. .

- Data Processing: The Raspberry Pi receives the pose data from the motion capture system over a ROS network on a VRPN ROS topic, this was initially parsed from the sensor readings into position and attitude.
- Data Remapping: Once the pose data is processed, the Raspberry Pi maps it to the to a gateway/MAVROS topic sending it to the flight controller onboard the UAV. All coordinate transformations (ENU->NED) are taken care of with MAVROS.
- Flight Control Update: The flight controller onboard the UAV receives the remapped pose data and uses it to update the UAV's flight control algorithms. The updated pose information can be used to adjust the UAV's flight trajectory, orientation, or other control parameters to achieve the desired flight behavior or control objectives based on the motion capture system feedback.
- Closed-Loop Control: The flight controller continuously receives pose feedback from the motion capture system via the Raspberry Pi, and uses it to update the UAV's flight control commands in a closed-loop fashion (PX4 uses a cascaded PID control system with more details provided in the educational document). This allows the UAV to maintain precise position and orientation control based on the real-time pose data provided by the motion capture system.

Overall, sending pose feedback from a motion capture system to a Raspberry Pi and remapping the data to the flight controller onboard a UAV involves acquiring, processing, and transmitting the pose data in a compatible format to enable real-time closed-loop control of the UAV based on the motion capture system's feedback.

Examples

This section provides two practical examples to help the user better understand the Clover platform, sensor fusion, UAV applications such as trajectory tracking, high level commands, and low level control. The reader will become familiar with an abundance of state-of-the-art open source UAV platforms/technologies such as:

Platform	Description
PX4	PX4 is an open-source flight control software for drones and other unmanned vehicles used on the Clover. It supports a wide range of platforms and sensors and is used in commercial and research applications.
Robot Operating System (ROS)	ROS is an open-source software framework for building robotic systems. It provides a set of libraries and tools for developing and managing robot software and is widely used in drone and robotics research.
MAVLink	MAVLink is a lightweight messaging protocol for communicating with unmanned systems. It is widely used in drone and robotics applications and provides a flexible and extensible communication framework.
QGroundControl (QGC)	QGC is an open-source ground control station software for drones and other unmanned vehicles. It provides a user-friendly interface for managing and monitoring drone flights and is widely used in commercial and research applications.

1. **A figure-8 high-level trajectory generation**: this example is outlined for both Software in the Loop (SITL) simulations and hardware testing with the Clover platform. Check out this interesting example from my trajectory tracking section!

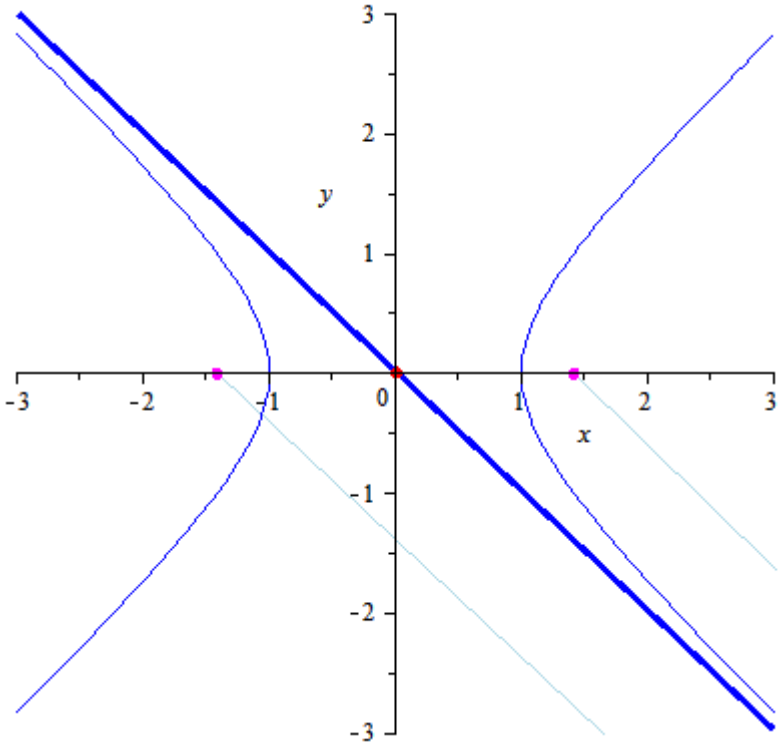


Fig.2 - Lemniscate of Bernoulli [reference].

Here's a summary of the importance of trajectory tracking for UAV applications:

- *Navigation and Path Planning*: Trajectory tracking allows UAVs to follow pre-defined paths or trajectories, which is essential for tasks such as aerial mapping, surveying, inspection, and monitoring.
- *Precision and Safety*: Trajectory tracking enables precise control of the UAV's position, velocity, and orientation, which is crucial for maintaining safe and stable flight operations. Precise trajectory tracking allows UAVs to avoid obstacles, maintain safe distances from other objects or aircraft, and operate in confined or complex environments with high precision, reducing the risk of collisions or accidents.
- *Autonomy and Scalability*: Trajectory tracking enables UAV autonomy, allowing them to operate independently without constant operator intervention. This enables UAVs to perform repetitive or complex tasks autonomously, freeing up human operators to focus on higher-level decision-making or supervisory roles. Trajectory tracking also facilitates scalable operations, where multiple UAVs can follow coordinated trajectories to perform collaborative tasks, such as swarm operations or coordinated data collection.
- *Flexibility and Adaptability*: Trajectory tracking allows UAVs to adapt their flight paths or trajectories in real-time based on changing conditions or objectives. UAVs can dynamically adjust their trajectories to accommodate changes in environmental conditions, mission requirements, or operational constraints, allowing for flexible and adaptive operations in dynamic or unpredictable environments.

In summary, trajectory tracking is crucial for UAV applications as it enables precise navigation, safety, efficiency, autonomy, and scalability, while optimizing payload performance and adaptability to changing conditions. It plays a fundamental role in ensuring that UAVs can accomplish their missions effectively and safely, making it a critical component of UAV operations in various industries and domains.

1. **Clover adaptive auto-tuning**: The second example shows the user how to implement the adaptive auto-tune module provided by PX4 to tune the low-level controllers or attitude control module. You can take a look into how this is accomplished with the Clover platform in the auto-tuning section.

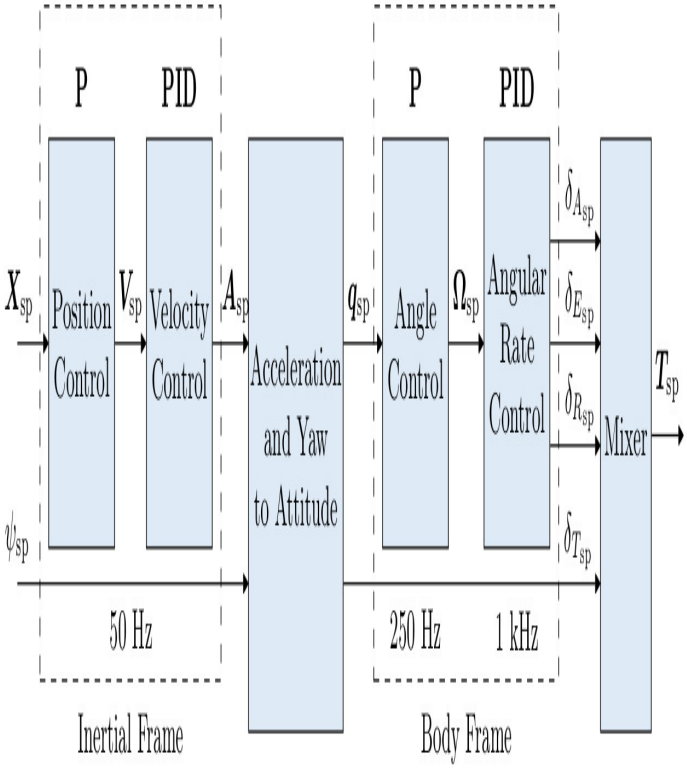


Fig.3 - Cascaded PX4 control system [reference].

This is a much faster and easier way to tune a real drone and provides good tuning for most air frames. Manual tuning is recommended when auto-tuning dos not work, or when fine-tuning is essential. However, the process is tedious and not easy especially for users with limited control background and experience. The Clover airframe provides sufficient base settings where auto-tuning can further improve performance depending on the Clover being used.

Here's a summary of the importance of low-level controller performance for UAV applications:

- *Flight Stability and Safety*: The low-level controller, typically implemented as a PID (Proportional-Integral-Derivative) or similar control algorithm, governs the UAV's attitude and position control. Properly tuning the low-level controller ensures that the UAV remains stable during flight, with accurate and responsive control inputs. This is essential for safe and reliable UAV operations, as it helps prevent undesired oscillations, overshooting, or instability that can lead to crashes or accidents.
- *Control Precision and Responsiveness*: Accurate control is crucial for achieving precise and responsive UAV maneuvers, such as smooth trajectory tracking, precise hovering, or dynamic maneuvers. Proper tuning of the low-level controller allows for precise control of the UAV's attitude, position, and velocity, enabling it to accurately follow desired flight trajectories, respond to changing conditions or commands, and perform complex flight maneuvers with high precision.

- *Adaptability and Robustness*: UAV operations can be subject to varying environmental conditions, payload configurations, or operational requirements. Proper low-level controller tuning allows for adaptability and robustness, enabling the UAV to perform reliably and accurately across a wide range of conditions or mission requirements. Tuning the controller parameters can help account for changes in payload mass, wind conditions, or other external factors, ensuring stable and responsive flight performance.



In summary, low-level controller tuning is crucial for UAV applications as it directly affects flight stability, control precision, payload performance, energy efficiency, adaptability, and compliance with safety and regulatory requirements. It is an essential step in optimizing the performance and safety of UAV operations, ensuring reliable and effective flight control for various applications across different industries and domains.

Conclusion

Over the course of this project I was able to extend my knowledge in robotic applications while enduring many ups and downs along the way. This greatly helped me with my research when testing controller development was required. The motivation behind this documentation is to improve this experience for other researchers, robotic developers, or hobbyists that have a desire to learn fundamental robotic application which is beginning to shape the world we know today. These details can be explored in a GitBook for those who are interested.

I provided many details on the interworking components required to achieve an indoor autonomous flight setup with the COEX Clover platform. With an extensive background in UAV control, I tried to provide a basic understanding of this for the readers benefit. There are many more sections I would like to include along with improving upon the existing ones. A few examples include firmware testing with hardware in the loop simulations, advanced trajectory generation, and an extensive list of flight examples for the Gazebo simulator with application to hardware.

Lastly, I would like to thank the entire COEX team that made this project possible by providing a wonderful platform with support. I would like to give a special thanks to Oleg Kalachev for helping me debug and succeed through applied learning. With that being said, I hope you all enjoy the resourceful content provided, and I plan on releasing more detailed documents on other interesting topics as I progress through my research and development.



Swarm in Blocks

CopterHack-2023, team **Atena**.

Project Status 📌

Final Video 📌



Table of contents 📌

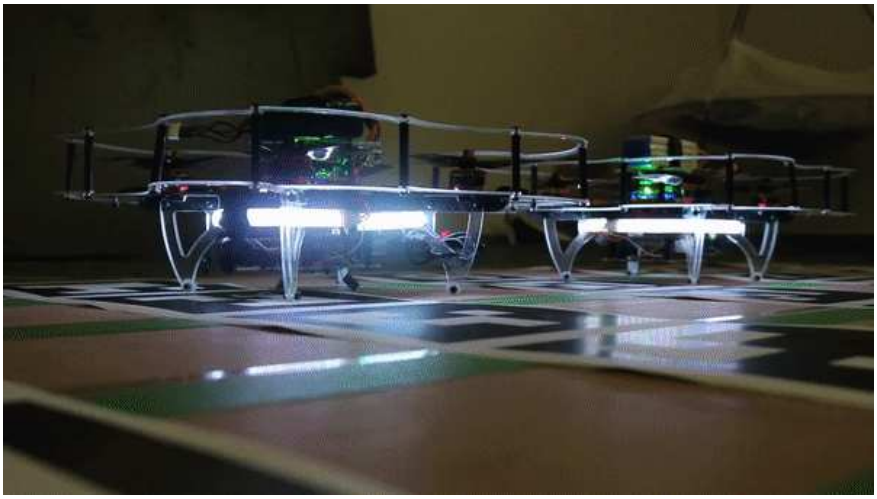
- Introduction
- Getting started
- Usage modes
- New Swarm Features
- Conclusion

Introduction 📌

Nowadays, **swarms of drones** are getting more and more applications and being used in several different areas, from agriculture to surveillance and rescues. But controlling a high amount of drones isn't a simple task, demanding a lot of studies and complex software.

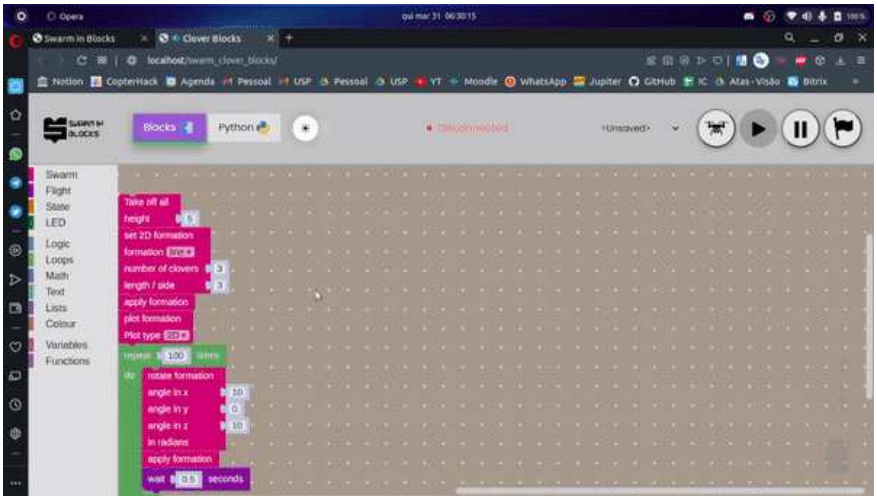
Swarm in Blocks (from it's origin in 2022) was born looking to make a *high-level interface based on the blocks language*, to make simple handling swarms, without requiring advanced knowledge in all the necessary platforms, creating tools to allow a lot of applications based on the user needs and also using the Clover platform.

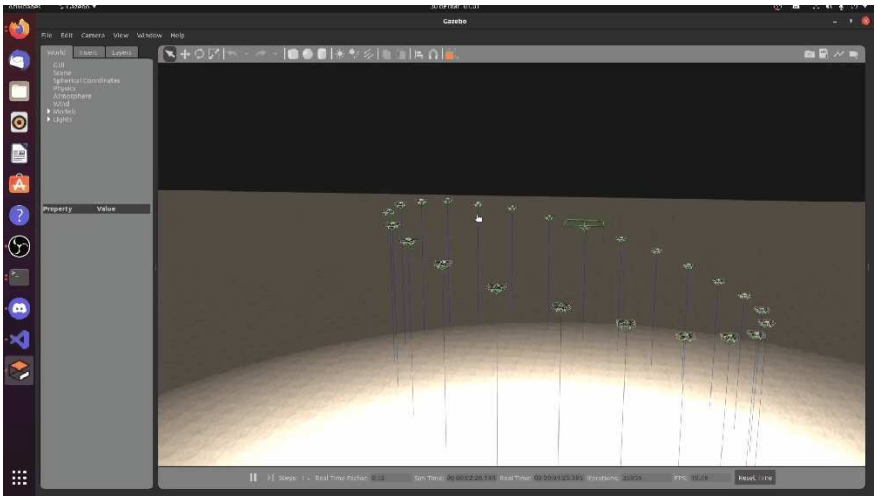
In 2023, Swarm in Blocks has taken an even bigger step, looking to fulfill our biggest vision **"It's never been easier to Swarm"**, we talk to transcend the local scope of the past project and explore the biggest problems for implementing a Swarm. For Copterhack 2023, we present Swarm in Blocks 2.0, an even more complete platform with the purpose of facing the biggest difficulties of a Swarm in a simple and polished way.



Swarm in Blocks 2022

Swarm in Blocks is a CopterHack 2022 project. It's a high-level interface based on the blocks language, which consists of fitting code parts, like a puzzle. Each script represents a functionality, for example, conditional structures, loops, or functions that receive parameters and return an instruction to the swarm.





For more information on our project from last year, see our final article in Swarm in Blocks 2022. In addition, we also recommend watching our final video from last year, Swarm in Blocks 2022 - Final Video.

Even with the huge facilities that the block platform offers, we realized that this was just the *tip of the iceberg* when it comes to deploying real swarms. Several other operational and conceptual problems in validating a real swarm still haunted the general public. With that, this year's project comes precisely with the purpose of **tackling the main problems in validating a Swarm in a simple and polished way**.

What's new

As already mentioned, of the various problems that can increase the complexity of a real swarm, we decided to deal with the ones that most afflicted us and reintegrated our solutions into our central platform, building a single extremely complete and cohesive platform.

Problem	Our Solution
Possible collision between drones (lack of safety especially for large Swarms)	Collision Avoidance System
Giant clutter to keep track of all Clovers in a swarm individually (several terminals, many simultaneous computers with several people to keep track of)	Swarm Station
Lack of basic features for handling a swarm pre-implemented in the Clover platform (such as access to battery data and raspberry computational power)	Full integration of low level data in our Swarm Station
Lack of security in indoor tests regarding the limitation of physical space (walls and objects) in the Swarm region	Safe Area Pop Up in Swarm Station
Decentralization of information and platforms for access	Web Homepage
Difficulty configuring physical drones for swarm	Our complete documentation with pre-designed settings for swarms in our repo image
Lack of a center for reports of successful tests with swarms of drones for the Clover platform describing the test conditions (odometry, etc.)	Show off section in our Gitbook

And many other solutions are also featured on our platform, for more information please check the solutions described clearly and in detail throughout our **Gitbook**. We recommend reading in order to understand the fundamental precepts of our platform.

📖 **Access our Gitbook!**

💻 **Access our GitHub!**

[↑ to top ↑]

Getting started 📌

Our platform was made to be extremely intuitive and easy to use. To start (after completing the installation we suggested in our gitbook), you can run the command:

```
roslaunch swarm_in_blocks simulation.launch num:=2
```

After that, you can open your browser and access our homepage by typing `localhost` in the search bar.

[↑ to top ↑]

Usage modes 📌

The Swarm in Blocks can be programmed either with the blocks interface or directly in Python and we developed three main launch modes, each one focused on a different application of the project, they are:

- *Planning Mode:* Its main goal is to allow the user to check the drones' layout, save and load formations, before starting the simulator or using real clovers. In order to need less computational power and avoid possible errors during the simulation.
- *Simulation Mode:* In this mode happens the simulation indeed, starting the Gazebo, the necessary ROS nodes and some other tools. It allows applying the developed features, which will be explained ahead and see how they would behave in real life.
- *Navigation Mode:* The last mode will support executing everything developed in real clovers so that it's possible to control a swarm with block programming. The biggest obstacle yet is the practical testing of this mode, due to the financial difficulty of acquiring a Clover swarm.

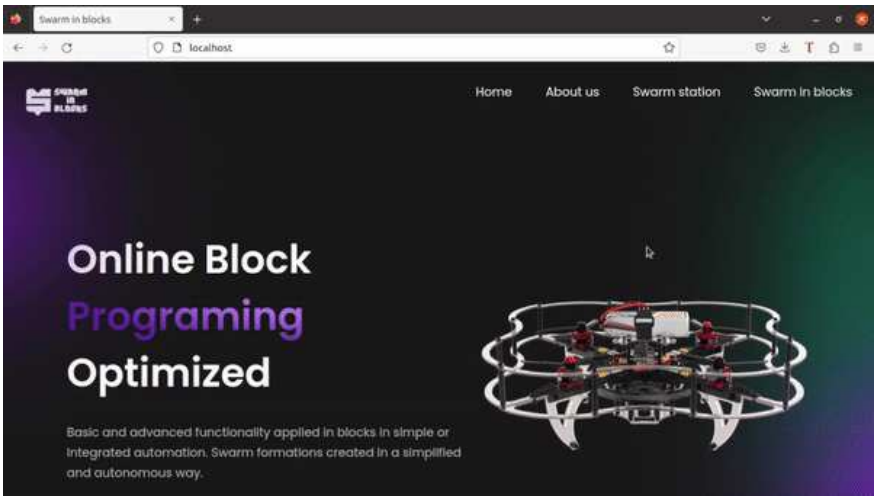
[↑ to top ↑]

New Swarm Features 📌

With our vision of solving the problems that most plague the deployment of a real swarm, we have developed several features (and even integrated platforms), below we will list our main developments:

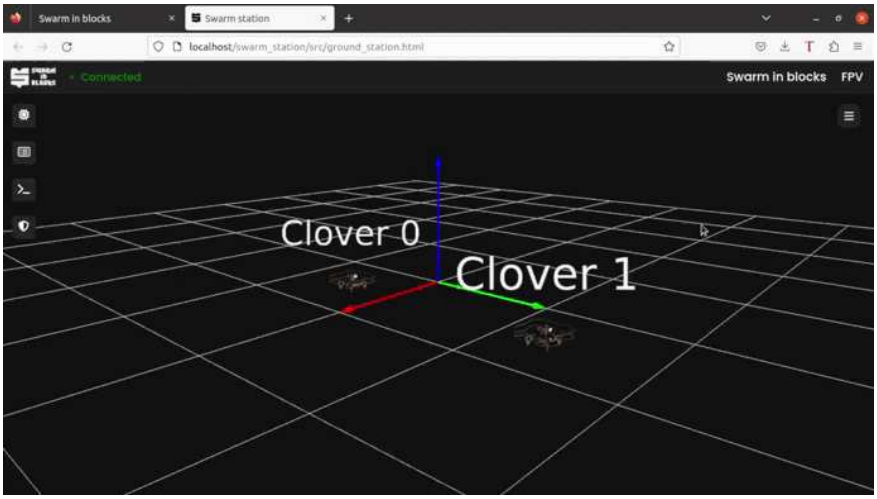
Homepage

Like last year, we really wanted to make it easier for the user to go through our platform. That's why this year we decided to restructure our Homepage, gathering our main features and functionalities.



Swarm Station

The main feature from our platform is the *Swarm Station*, which is a **3d Web Visualizer** that shows in real time all the necessary information regarding the drones state, such as real time positioning and visualization, which clover is connected, the topics available and a lot more. Also, you can define a safe area to ensure each drones safety, forcing them to land in case they cross the forbidden area. The front end runs completely on the web browser, saving processing and installation resources. It also comes with a web terminal, allowing the user to open several instances of a terminal emulation in just one click.



This package uses the ROS suite `rosbridge_server` to establish a communication between the ROS environment and the web server.

To run it, we recommend using **Firefox** browser to assure stability. But feel free to test it on other navigators.

If you launched our `simulation.launch` from the `swarm_in_blocks` package, then you just have to run

```
roslaunch swarm_station swarm_station.launch
```

Otherwise, you have to make sure that the `rosbridge_websocket` is running on port `9090`:

```
roslaunch rosbridge_server rosbridge_websocket.launch port:=9090
```

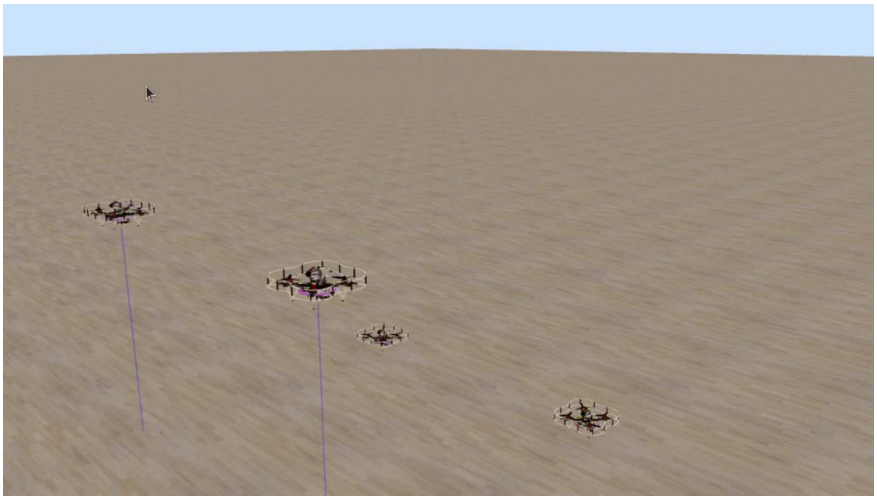
For more detailed instructions on how to use each single feature from the Swarm Station, check our Gitbook page about the station.

Swarm Collision Avoidance

When many drones move close to each other, collisions are very likely to occur. To avoid this problem, an algorithm was developed to avoid collisions between drones. When analyzing a collision, 3 types of scenario are possible, the case where one clover is stationary and the other in motion, the case where both are in motion and with parallel trajectories, and finally the case where both are in motion and with non-parallel trajectory.

To turn on the collision avoidance, it is necessary to run:

```
rosrun swarm_collision_avoidance swarm_collision_avoidance_node.py
```

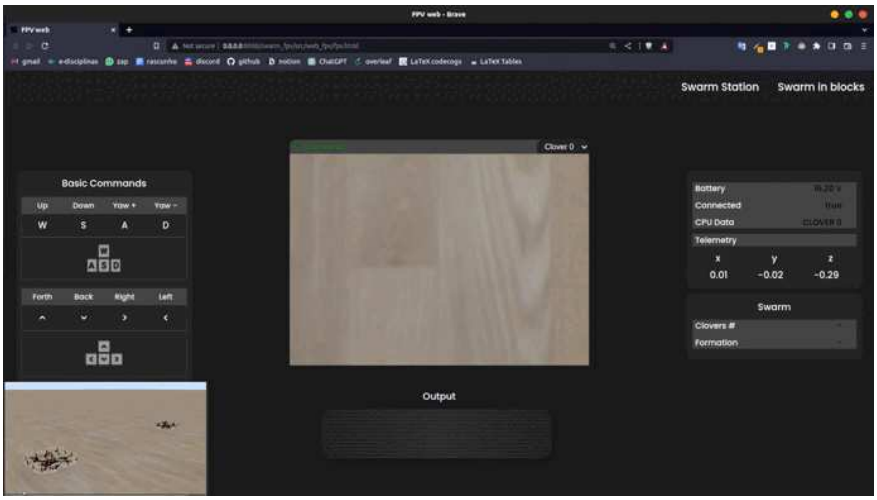


Rasp Package

The Raspberry package was developed to instantiate a node that will be responsible for collecting essential processing, memory and temperature information from the raspberry and send it to the Swarm Station. It's the package that should be put on the `catkin_ws/src/` directory of each Raspberry Pi, because it also contains the `realClover.launch` needed to launch the swarm on real life.

Swarm FPV

This package is a reformulation of one of the CopterHack 2022 implementations, the **Swarm First Person Viewer**. This year, we decided to restart its structure, making it run also completely on the web to integrate with the Swarm Station. It also depends on the `rosbridge_websocket` running on the port `9090` (default).



Real Swarm

In order to fly a real swarm using clover, we decided to take an approach of putting every clover on the same ROS network / environment so that the master could talk to each one of them.

We did this by separating each drone topics / nodes / services with namespaces. The goal is to achieve the same effect as the simulation that we've done in **CopterHack-2022**, so each drone would have its own `/cloverID` namespace, and the ID is the identifier for each drone.

In other words, instead of just `simple_offboard` node for a single drone, we'd now have `/clover0/simple_offboard`, `/clover1/simple_offboard` and so on.

To launch it, you need to first stop clover's default daemon, and then connect all Raspberries to the same network. After that, you should connect all their `roscore` to the same IP address (the master's), and then launch the `realClover.launch` file passing the `ID` argument as a parameter. Again, for more detailed information on how this works, please check out our gitbook:

```
sudo systemctl stop clover
roslaunch rasp_pkg realClover.launch ID:=0
```



We are aware that in the video the calibration of the drone control is not ideal, however, the objective of this test was really to validate the operation of the swarm in a real environment (which was actually done).

[↑ to top ↑]

Conclusion

Engineering and robotics challenges have always been the main driver of Team Athena, from which we seek to impact society through innovation. Last year, during CopterHack 2022, there was no lack of challenges of this type, and in them we grew and exceeded our limits, all to deliver the best possible project: **Swarm in Blocks**. All the motivation to facilitate a task as complex as the manipulation of swarms of drones, even through block programming, delighted us a lot and we hope that it delights all our users.

With that came the Swarm in Blocks 2.0, which brought with it innovations that optimized the clover's flight control and that could allow for greater emotions in the handling of the drone, in addition to focusing on greater flight safety. The Swarm in Blocks 2.0 presents new features for this year, such as the Web terminal, First Person View (FPV), Collision Avoidance, Clover UI and Swarm Station. However, the work will not stop there. Our goal is to further improve our system and next steps include validating Collision Avoidance outside the simulated world and performing performance tests with network communication solutions to optimize Real Swarm.

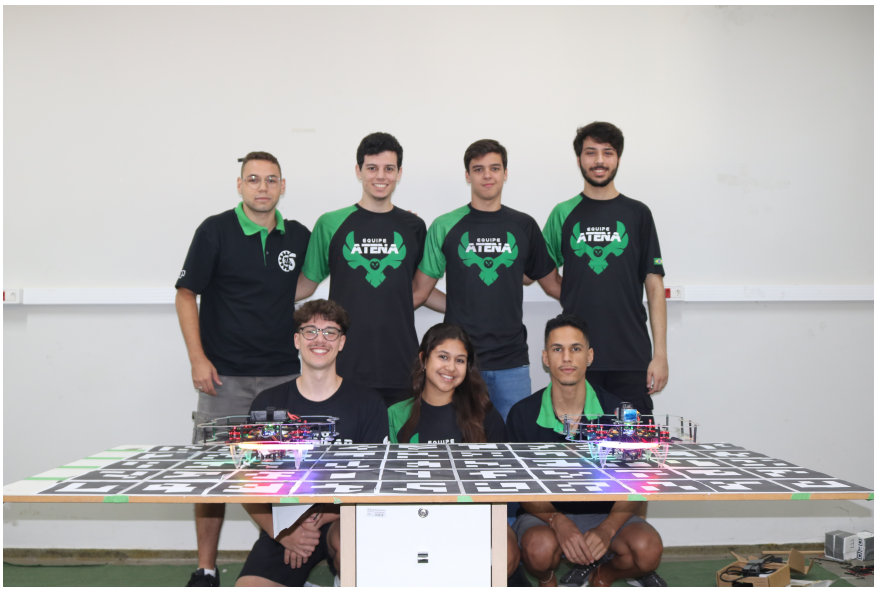
Finally, we thank the entire COEX team that made CopterHack 2023 possible and all the support given during the competition. We are Team Atena, creator of the Swarm in Blocks platform and we appreciate all your attention!

[↑ to top ↑]

The Atena Team

Atena Team 2023 (Swarm in Blocks 2.0):

- Agnes Bressan de Almeida : GitHub, LinkedIn
- Felipe Tommaselli: GitHub, LinkedIn
- Gabriel Ribeiro Rodrigues Dessotti : GitHub, LinkedIn
- José Carlos Andrade do Nascimento: GitHub, LinkedIn
- Lucas Sales Duarte : GitHub, LinkedIn
- Matheus Della Rocca Martins : GitHub, LinkedIn
- Nathan Fernandes Vilas Boas : GitHub, LinkedIn



In honor of Atena Team 2022:

- Guilherme Soares Silvestre : GitHub, LinkedIn
- Eduardo Morelli Fares: GitHub, LinkedIn
- Felipe Tommaselli: GitHub, LinkedIn
- João Aires C. F. Marsicano: GitHub, LinkedIn
- José Carlos Andrade do Nascimento: GitHub, LinkedIn
- Rafael Saud C. Ferro: GitHub, LinkedIn

[↑ to top ↑]

Advanced Clover 3: The Platform

CopterHack-2023, team **FTL**.

Team Information

```
#include "veryInterestingCommandDescription.h"
```

Team members:

- Maxim Ramanouski, @max8rr8.

Country: Belarus.

Project Description

Last year at CopterHack 2022, we created a project that simplified the simulation of Clover, and in 2021, we created a project that simplified the development of products for Clover (IDE and library for writing). The time has come to combine them and achieve unlimited power.

Project Idea

The idea of the project is to combine CloverIDE and CloverSim (a tool for running Clover simulations). Thus, a platform is planned that allows developing products based on Clover using a simulator and an advanced IDE. The platform will include the following features:

- Add a web interface that allows using CloverSim without touching the command line.
- Work both in the browser (without installing anything) and from CLI.
- Have a course that covers different aspects of clover.
- Simplify installation, especially in WSL.
- Running a simulation on a remote device (more powerful computer or cloud).

Project videos

Video presentation of the project: link.

Live presentation at CopterHack: TBD.

CLI demonstration: link.

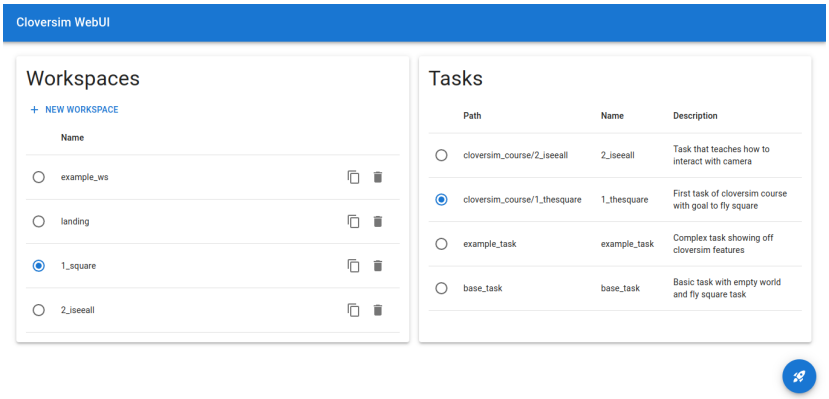
Installation

Installation process is described in the project documentation.

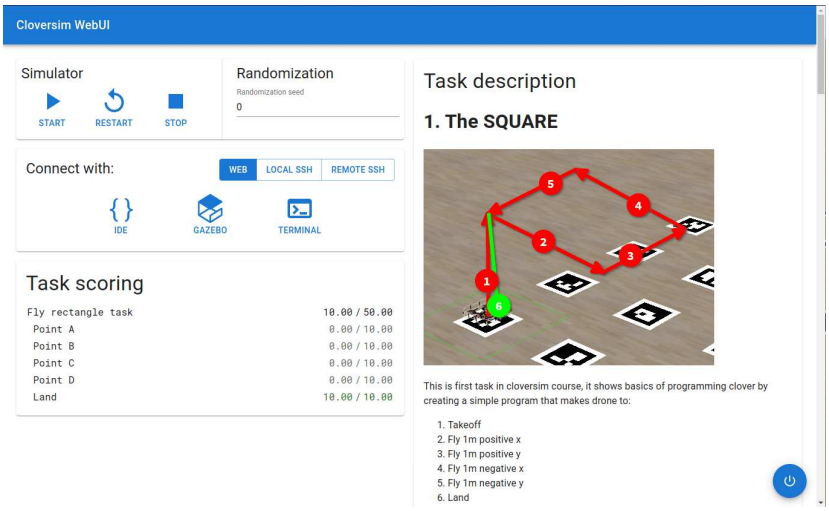
Usage

The CloverSim platform offers a seamless workflow for users:

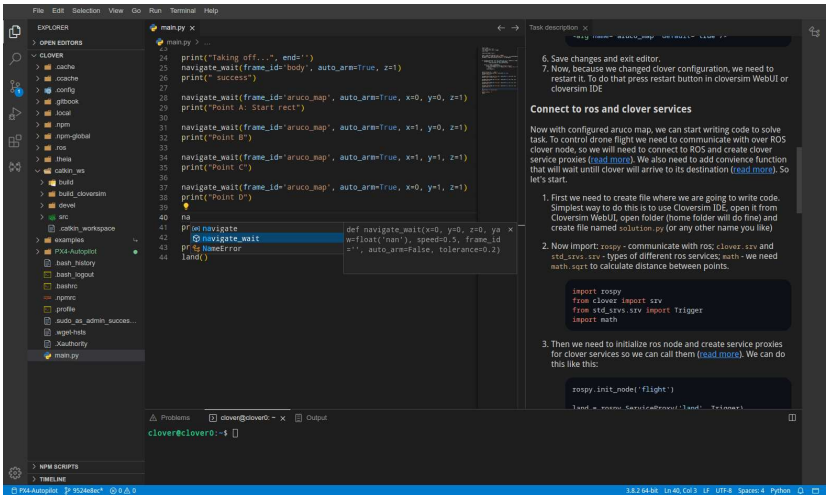
1. Users can effortlessly select or create a workspace and task and launch them with ease.



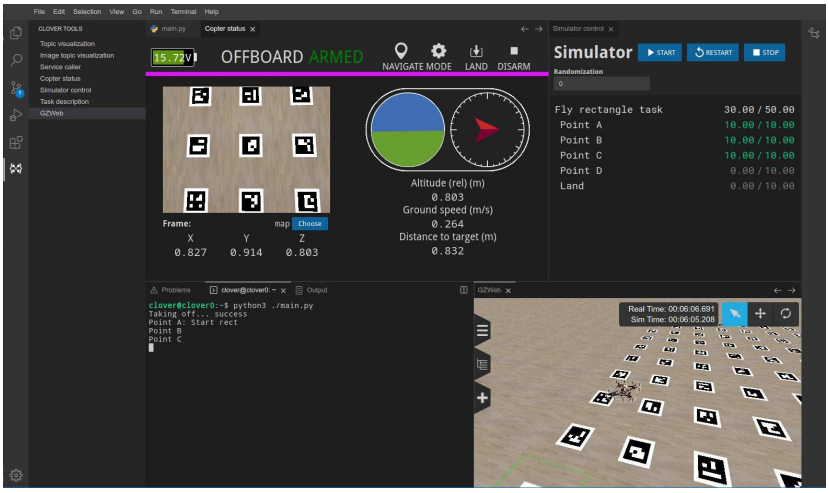
2. After launching the simulation, users are presented with CloverSim WebUI that provides them with an intuitive way to view their scores and progress, control the simulator, and access task descriptions and scoring information. From it users can open terminal, gzweb and more importantly they can easily access the CloverSim IDE to solve task.



3. The IDE provides a full suite of tools and features for writing and debugging code. One example is autocompletion to help streamline the development process, making it more efficient and effective.



4. Users can launch their programs with ease and monitor its progress via the GZWeb, CopterStatus, and SimulatorStatus views of the IDE.



5. Users can track their progress and scores in real-time and effortlessly restart the simulator if necessary. Additionally, different randomization seed can be set to check various inputs and outcomes.

We also have video demonstration/tutorial: link.

More features

- Easy installation process.
- Efficient simulation launch, surpassing traditional virtual machines.
- Generation of dynamic Gazebo worlds with randomization based on seed.
- Real-time task completion verification and score presentation.
- Execution with security in isolated containers.
- Multiple project capability without the need for multiple virtual machine images.
- WebUI for ease of use, removing the need to use the command line.
- IDE similar to VSCode with support for C++ and Python, including autocompletion and autoformatting.

- Custom-patched GZWeb with bug fixes and additional features, including the display of the Clover LED strip.
- GZWeb provides a follow-objects feature superior to that of Gazebo.
- IDE includes tools to interact with ROS, such as topic visualization, service calling, and image topic visualization.
- IDE also includes Copter Status, displaying most of the drone's information, including position, camera, and LED strip, in one view.
- IDE integrates with the simulator by providing control from it, viewing task descriptions, and opening GZWeb.

We also have developed a learning course based on CloverSim: link. It currently has the following tasks:

- 1_thesquare - First task of CloverSim course with goal to fly square.
- 2_iseeall - Task that teaches how to interact with camera.
- 3_landmid - Find and land onto randomly positioned object.
- 4_flybyline - Flying along the line.
- 5_posknown - Find position of objects relative to ArUco map.

More details

At this point, our platform consists of four major parts:

- CloverSim - tool that manages simulation.
- CloverSim Basefs - container image that is used in simulator.
- Clover IDE - clover ide tools and theia.
- CloverSim course - course with tasks based on our platform.

CloverSim

The simulation architecture is a continuation of work from CopterHack 2022, but while 2022 version was closer to Proof-of-Concept, the updated version is more robust.

There are three major difference in simulator architecture

- Replacement of `systemd-nspawn` with `runc` provides us higher degree of container control and seemingless support of non-systemd systems, for example WSL.
- Migration to squash fs images, which greatly reduced size of installed CloverSim from 13 gigabytes to just 3.5 gigabytes.
- Tasks are now mounted instead of being copied and also build before starting.

Because of the way catkin_make works, it is incredibly slow when new packages are added (whole cmake configuration is rerun for all packages). catkin_make provides a way to build only some packages, but it caches this packages and to reset this cache you need to recompile whole catkin_make. But we have found a solution: `catkin_make -DCATKIN_WHITELIST_PACKAGES="task;CloverSim" --build`

`build_CloverSim` This command, builds only CloverSim and task package in separate build directory, this drastically reduces time that catkin_make takes, and keeps expected behavior of catkin_make without arguments.

There are also differences in tool that launches simulation:

- Client-server architecture allows us to create web UI and run CloverSim on server.
- More robust error handling improves user experience.
- Rewritten in rust, better reliability and development experience.

CloverSim basefs

Version 2 integrates CloverIDE into system. We also updated clover in simulator to v0.23 and added web terminal. Basefs is now squashed and doesn't require additional installation. It also uses patched(by us) version of gzweb that is more suitable for our use-case:

- Unlike original GZWeb assets can be dynamically loaded, which is required to support dynamically generated tasks.
- It also implements multiple bugfixes for rendering, UI.
- Fixed performance, original gzweb had two constantly running loops that used 200% of cpu. We fixed this by instead using synchronization primitives.
- Clover LED strip is rendered, our gzweb connects to ROS and pulls LED data from there to render LED strip like Gazebo does.
- Users can now follow-objects like in Gazebo better actually.
- Reconnect on disconnect, when simulator is restarted gzweb looses connection and it now can automatically reconnect.

Patched gzweb available there: FTL-team/gzweb.

CloverIDE

CloverIDE got some updates too:

- We have updated theia and extensions used.
- Better C++ support via clangd.
- Clover IDE tools can now reconnect after simulator restart.
- Copter Status now displays LED strip status.
- Tools ui has better support for different themes.

But the most important change is CloverSim integration, there are new tools (task description, simulator control and gzweb). While gzweb tool is just an iframe (though it's very cool to have it in IDE).

Task description and simulator control are more interesting as they have to interact with both IDE and CloverSim, to maintain different versions support we use quite interesting trick, extension webview after being initialized dynamically loads JavaScript from CloverSim. That provides better integration between two.

CloverSim course

CloverSim course is a new part of our platform. It uses robust task API of CloverSim to create practical learning course. It currently teaches different aspects of clover development that i encountered during my participation in different contests involving clover. But we are happy to accpet suggestions about other aspects we should teach in out course.

Conclusion

This project is a final (or maybe there is more?) project of our advanced clover saga. AdvancedClover is a project that is easy to use and greatly improves experience during learning about clover, participating in clover based competitions and development clover based projects. We thank COEX team for their support and look forward to further cooperation.

"QCS" - the network of Clover charging stations

CopterHack-2023, team **Lyceum 128**.

Network realisation

Our charging stations use Python web server created with Django framework. On that server we storage information about charging stations:

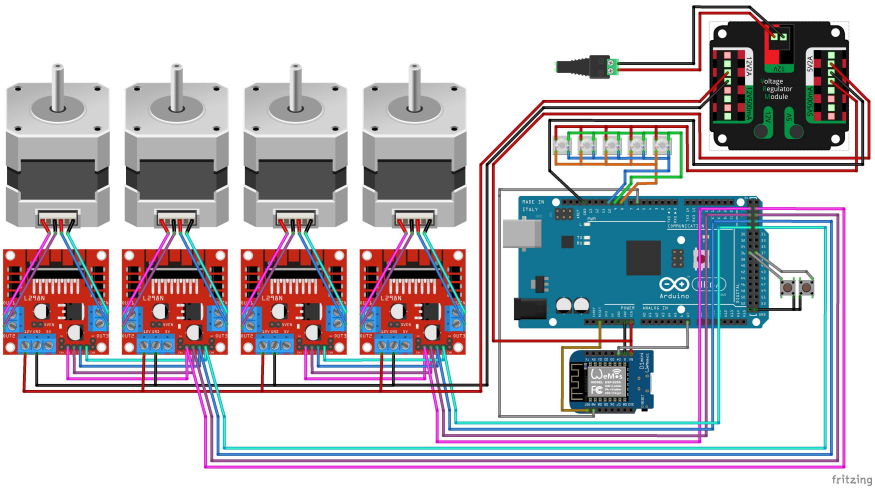
- Position (GPS + ArUco marker).
- Possibility to drone landing.
- Drone info (If it's on it).

To connect to server we use API with special personal key for every drone and station. It can be regenerated if secured key became public.

If you want to test station without drone you can use API Debug page. You must be in your account to open it.

Electronics in the station

There are Arduino Mega and Wemos D1 on the station.



Wemos D1 connect with server to collect information, do tasks. Arduino Mega receive signals from Wemos and make physical updates such as moving landing platform, LED indication and other more.

After completing mission Wemos send request to a server to confirm updates on the server.

Clover flight

We're using recursive landing algorithm to achieve success landing. Small ArUco marker is on the landing platform. Camera can use this marker on the ~25cm height. Next drone use standard landing.

Visit our landing and API page

https://qcs.pythonanywhere.com/

Source code

Of that project is in our GitHub page.

Team

CH2023, Lyceum 128.

- Mikhail Konstantinov, @mikemka, programmer.
- Julia Shvecova, @Juli_Phil, science adviser.
- Oleg Sherstobitov, @kulumuluu, constructor.

Swarm-in-blocks

Final Video



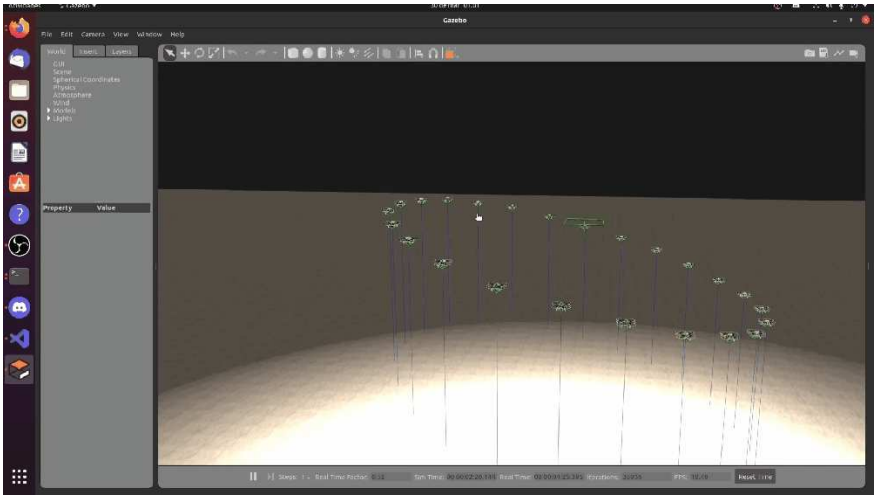
Detailed Gitbook

Check our Gitbook, with the detailed information about all that was developed by Atena Team during CopterHack 2022: https://swarm-in-blocks.gitbook.io/swarm-in-blocks/introduction/swarm-in-blocks.

> The Clover Platform was forked and adapted to work with swarms. The Swarm in Blocks project is a separated repository with all our swarm manipulation tools for clover. Link of the fork of the Clover Platform: https://github.com/Grupo-SEMEAR-USP/clover . Link of the Swarm in Blocks repository: https://github.com/Grupo-SEMEAR-USP/swarm_in_blocks .

Introduction

Nowadays, swarms of drones are getting more and more applications and being used in several different areas, from agriculture to surveillance, and rescues, but controlling a high amount of drones usually isn't a simple task, demanding a lot of studies and complex software. Swarm in Blocks was born looking to make a high-level interface based on the blocks language, to make simple handling swarms, without requiring advanced knowledge in all the necessary platforms, creating tools to allow a lot of applications based on the user needs and also using the Clover platform, which has a lot of advantages as being complete and intuitive, supporting all the project goals.



Usability

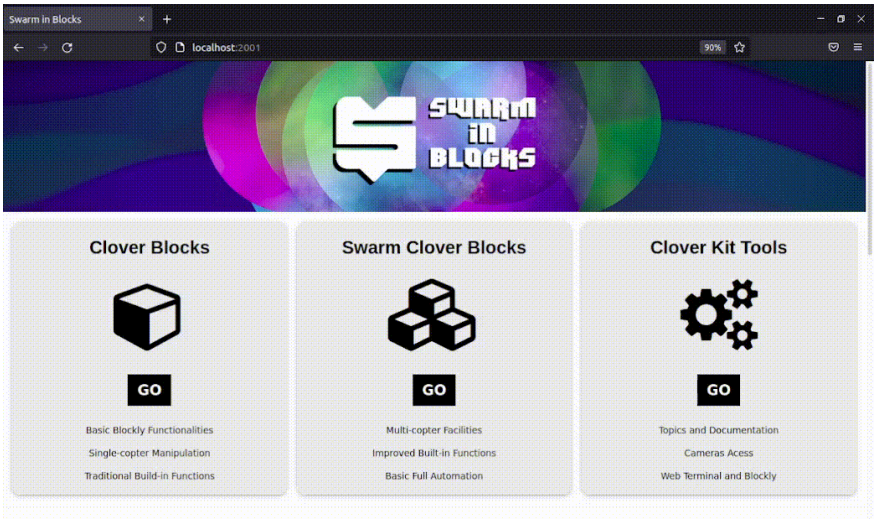
How it works

The Swarm in Blocks can be programmed either with the blocks interface or directly in Python and we developed three main launch modes, each one focused on a different application of the project, they are:

- **Planning Mode:** Its main goal is to allow the user to check the drones' layout, save and load formations, before starting the simulator or using real clovers. In order to need less computational power and avoid possible errors during the simulation.
- **Simulation Mode:** In this mode happens the simulation indeed, starting the Gazebo, the necessary ROS nodes and some other tools. It allows applying the developed features, which will be explained ahead and see how they would behave in real life.
- **Navigation Mode:** The last mode will support executing everything developed in real clovers so that it's possible to control a swarm with block programming. The biggest obstacle yet is the practical testing of this mode, due to the financial difficulty to afford a Clover swarm.

Blocks Interface

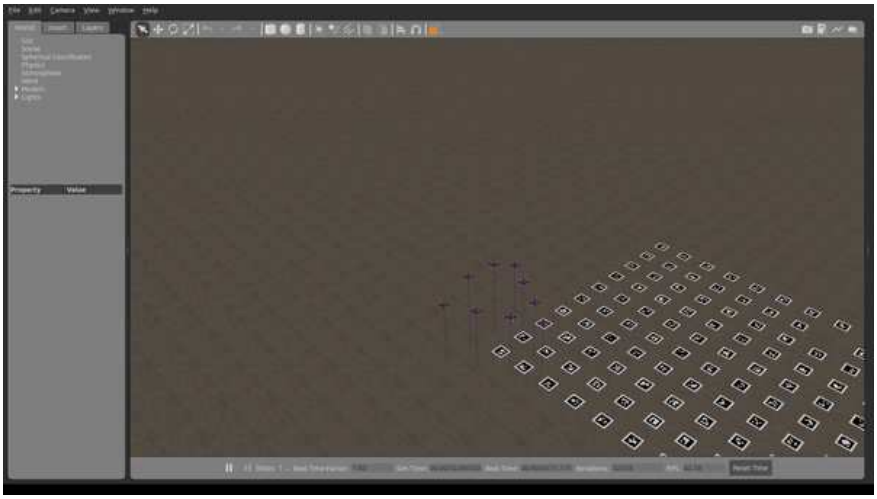
The entire Swarm in Blocks project was designed so that the user was in an intuitive and comfortable environment within the manipulation of swarms, for this, the existing platform with clover packages was completely rethought and adapted. In our gitbook, we have more details about the front-end design, how the user can interact with it, and achieve our main goal: programming in blocks.



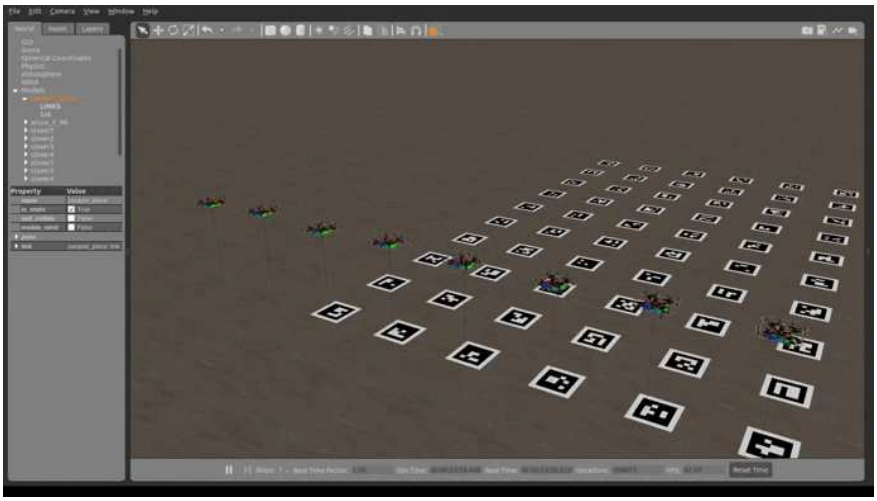
Features

Along with the project, we developed some features, that can be used together or independently and also serve as base for the implementation of more specific and advanced innovations. Here are the list and a brief explanation of each tool, to see more details and instructions about their use, check our gitbook!

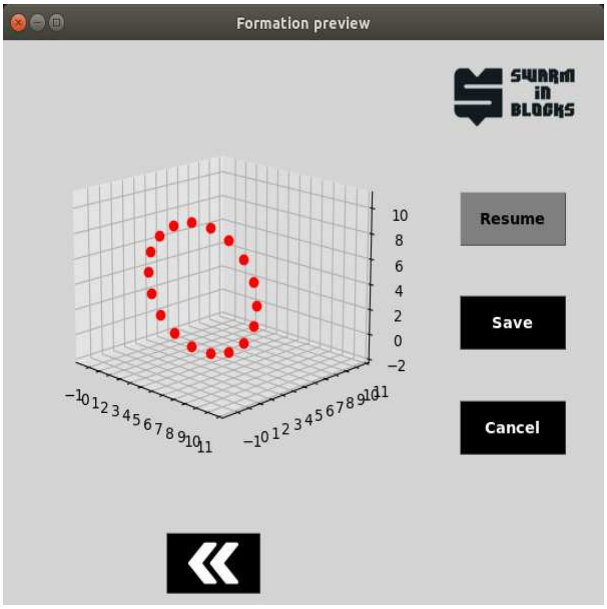
- **Formations:** There are some types of formation that were developed in order to create figures and other images, for uses in engineering and spectacles. They are:
    - 2D Formations: We made functions to generate some simple geometries, allowing the user to set the number of clovers used and the size of the figure, the geometries options made until now are *circle*, *empty square*, *full square* and *triangle*.
    - 3D Formations: Besides the 2D figures, there are also some simple 3D geometries, which are *cube*, *sphere*, and *pyramid*.
    - Alphabet: There is also the option to generate letters and words, using our Clover swarm.
    - 3D Figures: Lastly, we have the alternative to make other more complex 3D formations, for this we use a library called Open3D that deals with 3D data, allowing the drones to create any 3D image the user inputs since it's in the supported formats.
- **Transformations:** In addition to creating the formations, it's important to give the option of editing their disposition, so some operations were developed. It also makes possible to execute more complex actions and activities with the Clovers. The transformations operations are:
    - Scale: Changes the distance between the drones, increasing or decreasing the image.
    - Translate: All the drones move the same distance in the chosen directions.
    - Rotate: The formation rotates around a determined axis.



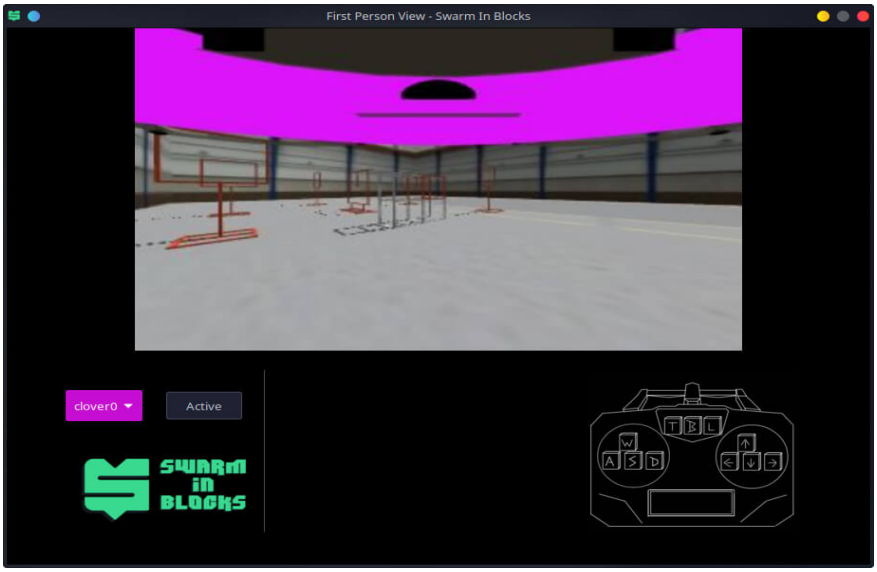
- **LED effects:** Enjoying the LEDs included in the Clover, we made some functions to apply effects in all the swarm, creating figures and operations with the LEDs too.



- **Swarm Preview:** The main goal of this feature is to help the user to visualize how the swarm will behave in the simulation or real life, without using a lot of computational power and avoiding some problems in the simulation. This way a 2D or 3D image illustrating the drones' disposition can pop up on the screen when using this function.



- **First Person View (FPV):** The FPV node makes it a lot easier to visualize each drone's camera individually and also control each one of them at a time using keyboard bindings.



All these features can be very useful for some applications and also be attractive to arouse the curiosity of the general public.

Conclusion

Over the last months we studied a lot, grew, and surpassed our limits, trying to explore some swarm applications, all to deliver the best possible project: **Swarm in Blocks**. Our motivation was to facilitate such a complex task as the manipulation of swarms of drones, through block programming, and it delighted us a lot and we hope it will fascinate all our users. We tried to resume all the project and its features in this article, but as there are many details and needed explanations, it was made a gitbook, to explore them for those who are interested.

For us, the results achieved were very expressive and positive, however, we believe that there is still room for improvement in the project, both considering the robustness of the swarm and the simplification of the usability of our platform. Improvement in the collision avoidance system, implementation of more formations and tests in real Clovers are some of the points that we hope to develop in future opportunities.

Finally, we thank the entire COEX team that made it possible for CopterHack 2022 to take place and all the support given during the competition. We are **Atena Team**, creator of the **Swarm in Blocks** platform and we thank you for your attention!

The Atena Team members

- Guilherme Soares Silvestre : GitHub, LinkedIn
- Eduardo Morelli Fares: GitHub, LinkedIn
- Felipe Tommaselli: GitHub, LinkedIn
- João Aires C. F. Marsicano: GitHub, LinkedIn
- José Carlos Andrade do Nascimento: GitHub, LinkedIn
- Rafael Saud C. Ferro: GitHub, LinkedIn

Obstacle avoidance using artificial potential fields method

CopterHack-2022, team **Stereo**.

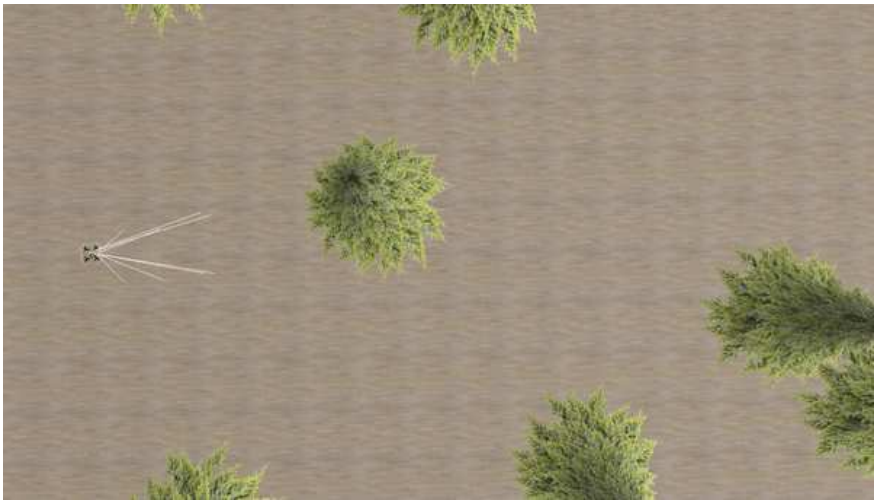
Team information

The list of team members:

- Denis Konstantinov, @den_konstantinov, engineer and developer.

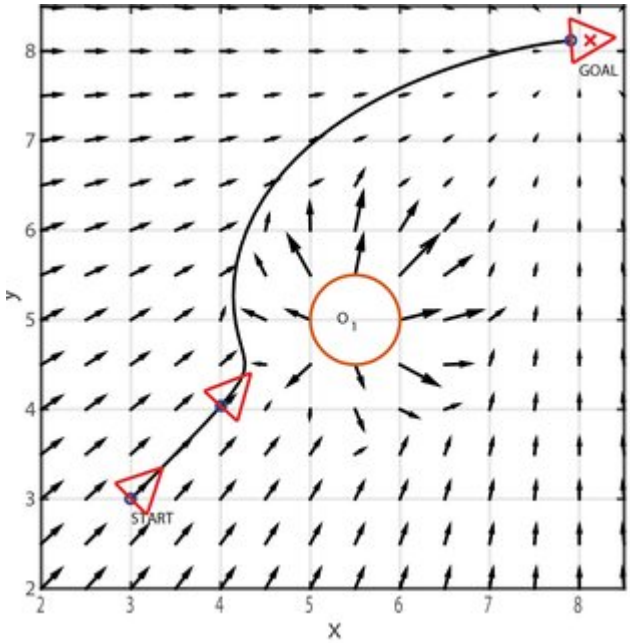
The project repository is here

Project description



This repository contains obstacle avoidance system for quadcopters with Raspberry Pi 4 onboard computer. The code in this repository is designed to work with Clover Raspberry Pi image and special PX4-based firmware modified for easier communication with Raspberry Pi.

Artificial potential fields method is based on considering quadcopter, obstacles and target point as electric-charged points. Quadcopter and obstacles have positive charge, and target point is assigned with negative charge. This results in quadcopter "attracting" itself to the target point, while being repelled by the same-signed charges of obstacles. Using this analogy, you can compute a safe, collision-free trajectory, which can be executed by the vehicle.



It's obvious that you need some sort of geometrical information about the surrounding world if you want to avoid obstacles. This algorithm uses Intel RealSense D435 depth camera - it provides a 3D point cloud which can be easily used for potential fields computation.

Installation

This guide is intended to be used on Ubuntu 20.04. Python version is 3.8.10, but it's very likely that it will work on other versions like 3.7, 3.9, 3.10 without any changes.

1. Install the Clover simulator.
2. Install realsense_gazebo_plugin and realsense2_description:

```
cd ~/catkin_ws/src
git clone https://github.com/issaiass/realsense_gazebo_plugin
git clone https://github.com/issaiass/realsense2_description
cd ~/catkin_ws
catkin_make
```

3. Replace your `~/catkin_ws/src/clover/clover_description` folder with one in this repository. This will add RealSense D435 to quadcopter model and make other minor changes so you will be able to reproduce our results.
4. Install all necessary Python packages

```
pip3 install -r requirements.txt
sudo apt-get install ros-noetic-ros-numpy
```

5. Make sure your `PYTHONPATH` env variable is set as

```
PYTHONPATH=/home/<username>/catkin_ws/devel/lib/python3/dist-packages:/op
```

If it's not, add the following paths to `PYTHONPATH`.

Launch

Launch the clover simulator and spawn some obstacles:

```
roslaunch clover_simulation simulator.launch
```

Open another terminal window, and launch the takeoff script. The drone will arm its motors and take off to the altitude specified in takeoff.py script:

```
python3 ./scripts/takeoff.py
```

Then, you should open main.py script and specify the target point (as x, y, z array). It is defined on the line

```
nav = AvoidanceNavigation(np.array([0, 0, 2]))
```

Finally, launch main.py to start the obstacle avoidance flight:

```
python3 main.py
```

Obstacle avoidance algorithm description

This algorithm uses a point cloud produced by stereo camera to infer the flight path by simulating a motion of charged particle. Each point from the point cloud is considered as obstacle, and repels the vehicle with the force directed in 'point->vehicle' direction. The repelling force has a magnitude `q_repel / dist` (dist is distance from vehicle to point). Similarly, an attraction point attracts the vehicle with the constant magnitude of `q_attract`. **The sum of these force vectors is the desired flight direction.**. The speed of a particle is set as `speed` argument of `AvoidanceNavigation` and directed along the resulted force vector. This way, we get a motion equation which we can solve numerically. Now, we have the trajectory which can be executed by the vehicle.

The algorithm is still in development, so we can't guarantee that default parameters will work in all cases. However, you may tweak the parameters yourself. Here is a brief description of each parameter in descending order of importance:

- `speed` - if your vehicle systematically crashes, the first thing you should do is to decrease the flight speed. Values of 1.5-2 m/s show good performance on most obstacle types

- `dist_threshold` - this sets a 'sphere of influence' for individual obstacle point. Larger values of this parameter will result in trajectories with bigger margin from obstacle
- `lead` - this is a smoothing factor of trajectory execution. If this value is too big, vehicle will ignore sharp obstacle avoidance maneuvers and risks crashing into a small/thin obstacle. If this value is too small, vehicle will fly too wobbly.
- `q_repel` - the strength of repelling force is determined by this parameter.

The Clover Rescue Project

CopterHack-2022, Clover Rescue Team - When something went wrong.

Team information

The list of team members

- Кирилл Лещинский, @k_leshchinskiy - TeamLead.
- Кузнецов Михаил, @fletchling_dev - Software Developer.
- Даниил Валишин, @Astel_1 - Tech Specialist/Python programmer.
- Роман Сибирцев, @r_sibirtsev - Hardware engineer/tester.

Project description

Table of contents

- Idea
- How it works
- Required hardware
- Operating Instructions
- Installation instructions
- Work example/functions
- Settings
- Mobile version
- Bots
- Our plans for the future

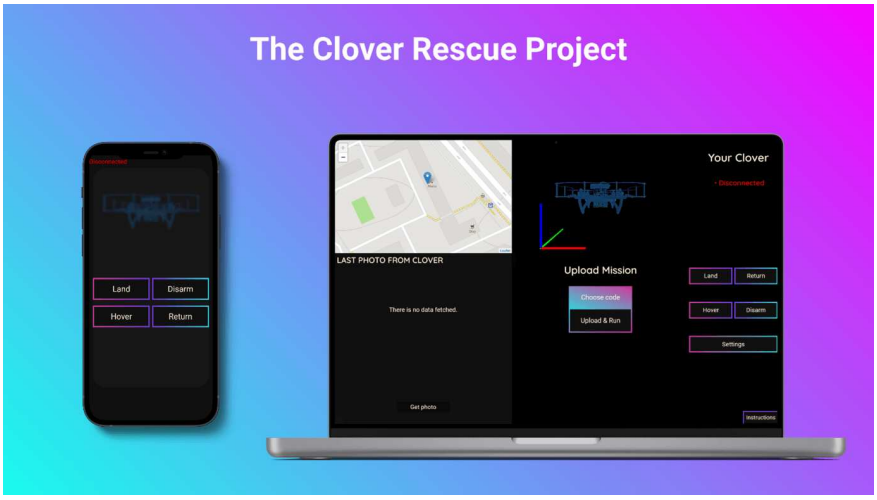
Project Idea

The idea of this project came immediately, it lies on the surface. A system that makes situations where pilots cannot find their flown away drone or stop it at full speed flying into a wall, a thing of the past, is something that pilots have been missing for a long time. The key feature of our software is that users can manage their Clover from anywhere in the world, this software replaces FTP and SSH (users can upload the code to their drone and run it directly from our site). Also, if radio communication with the Clover is lost, it can be returned to the user's or takeoff location with just one click. Moreover, the user can monitor the status in realtime, as well as location, camera data, and airborne position data of the drone. There are also functions that can be useful in emergency situations, such as landing, hovering and disarming the drone remotely from our website. A mobile version of the site with full functionality is also available!

Presentation video

Functional review video

Link to the website



How It Works

Links to repositories:

- https://github.com/DevMBS/CRTClient
- https://github.com/DevMBS/CRTClover

The first repository is the main server that users and their drones connect to. This server provides communication and control of the drone through a user-friendly interface. The second repository represents the server that runs on the drone and connects to the main server. This server reads and transmits telemetry to the main server (which is displayed as a 3-D visualization). It also takes commands from the user and executes them. The Socket.IO library is used to transfer data between the client, server and drone. After connecting the client and the drone to the server, they are added to a unique room with their UID, and already in it they exchange data. You can learn more about how it works by visiting the repositories.

Required Hardware

All you need is COEX Clover 3/4, Raspberry Pi 3/4, USB WIFI Modem and RPi Camera!

Operating Instructions

Firstly, users need to register on our website.



After registration the main control panel and installation instructions open.

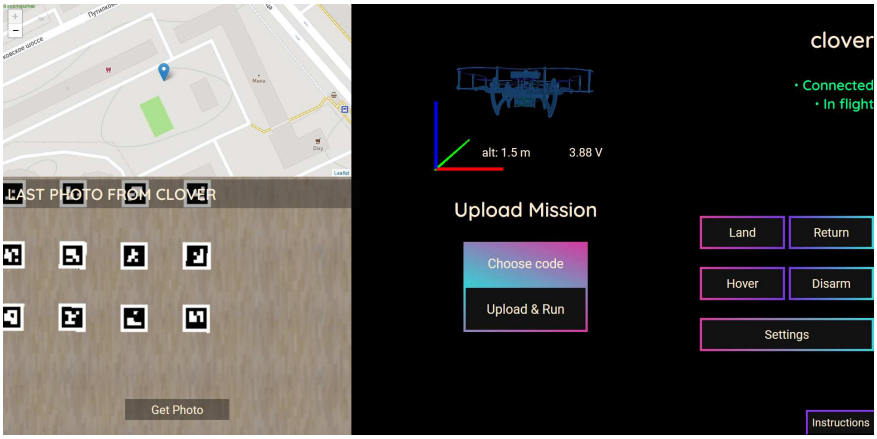
Installation Instructions

First, you need to connect to your Clover via SSH and paste the command indicated in the instructions that opened (if it is not open, you can open it by clicking on the "Instructions" button). The command looks like that:

```
wget https://48c5-94-29-124-254.eu.ngrok.io/assets/installers/install.sh && suc
```

When the software is installed, the server will automatically start. After installation, you can forget about manually launching the software, it will automatically start and connect to the main server after turning on the Clover!

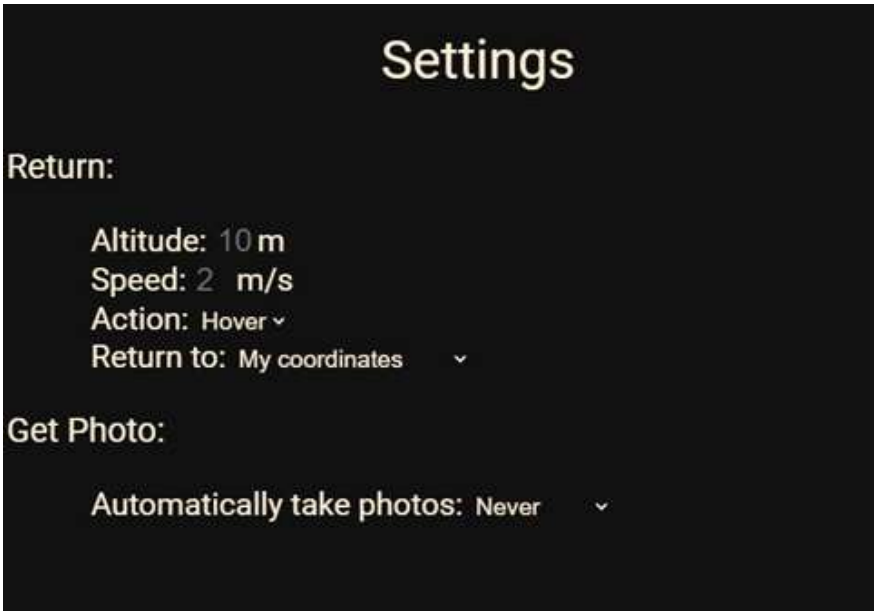
Work Example, Functions



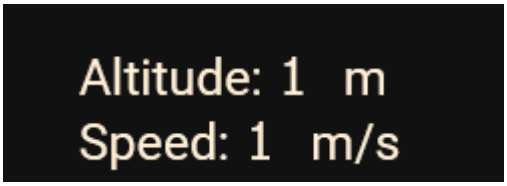
On the website there are several commands for controlling the drone. “Get photo” allows you to get an image from the drone camera. The “Land” button lands the drone. The "Return" command returns drone to the operator, according to GPS coordinates (this requires a stable connection of the drone with 10 or more satellites). "Hover" makes the drone hover in the air. "Disarm" instantly disables the drone's motors, so you need to be careful with this command. The “Choose Code” and “Upload & Run” buttons allow users to select a code written in Python, upload it to the drone and run it. Also, users will see output of their code and all its errors. There is also interactive map with markers, blue marker the is location of

the user, purple marker is the location of his drone. Also, as you can see, there is a real-time visualization of the Clover's airborne position, as well as its altitude and the average voltage between the battery cells.

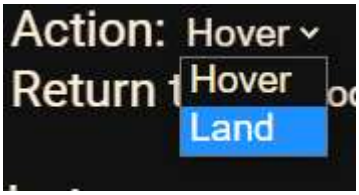
Settings



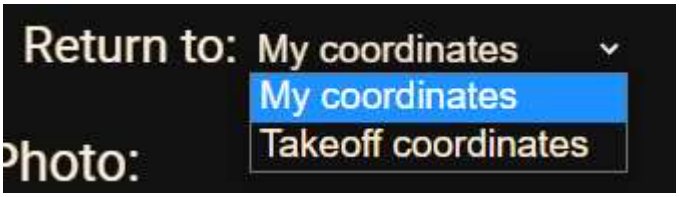
In the settings users can set speed and altitude of the return.



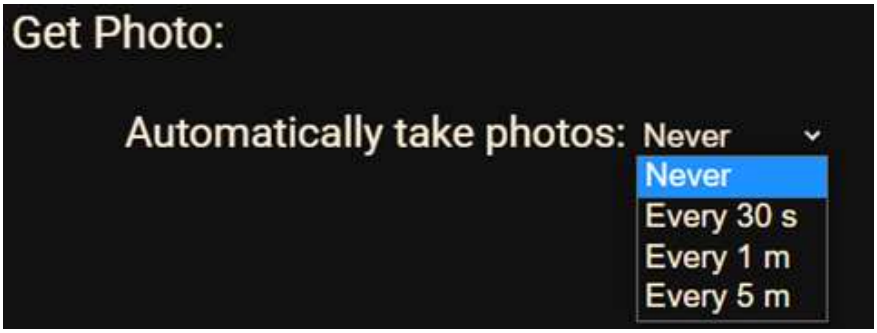
Users can choose an action after return (hover or land).



...And the place where the drone will return (User coordinates or takeoff coordinates).



Users can also set the period for automatically receiving photos from the drone.

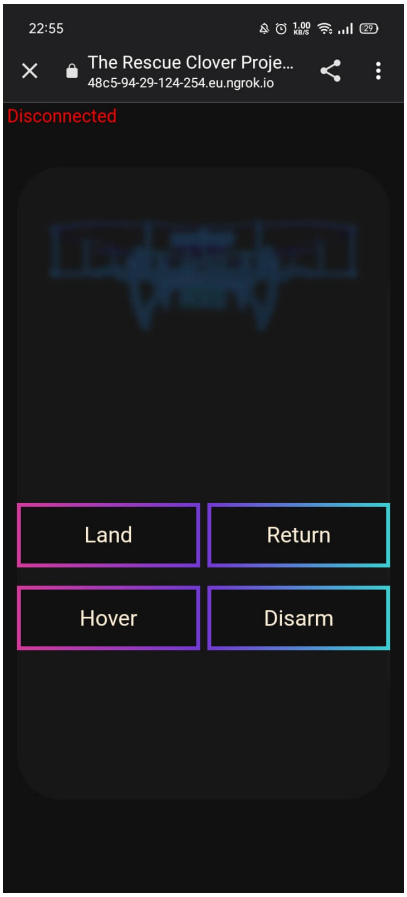


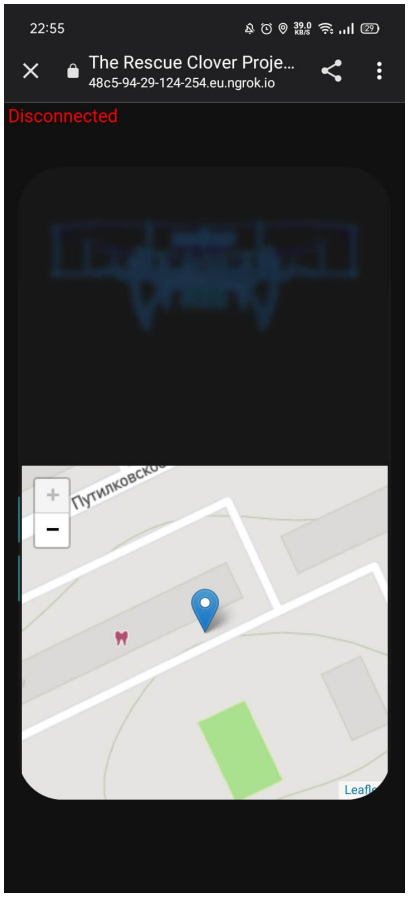
At the top of the website is the status of your drone (Disconnected/Connected, disarmed, Connected, in flight).

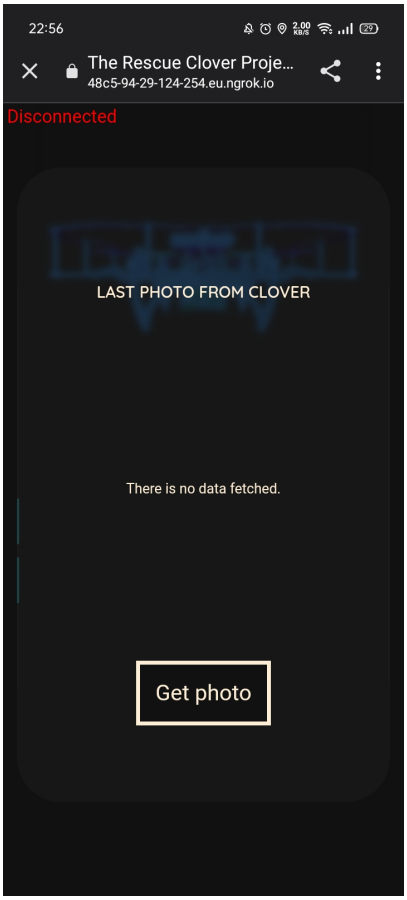


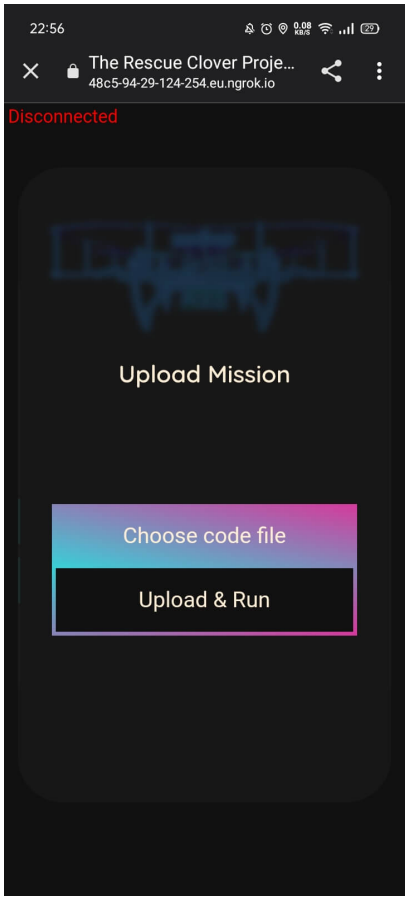
Mobile Version

The mobile version of the site has absolutely the same functionality (swipe to the right/left to move between control panels).

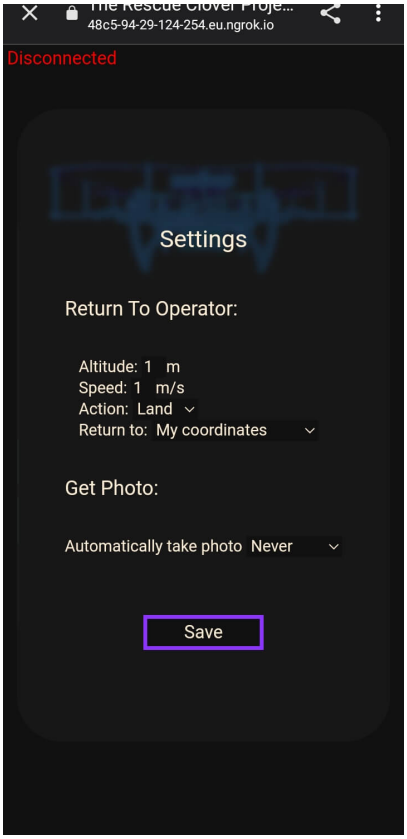

22:55
The Rescue Clover Proje...
48c5-94-29-124-254.eu.ngrok.io
Disconnected
Land
Return
Hover
Disarm


22:55
The Rescue Clover Proje...
48c5-94-29-124-254.eu.ngrok.io
Disconnected


22:56
The Rescue Clover Proje...
48c5-94-29-124-254.eu.ngrok.io
Disconnected
LAST PHOTO FROM CLOVER
There is no data fetched.
Get photo

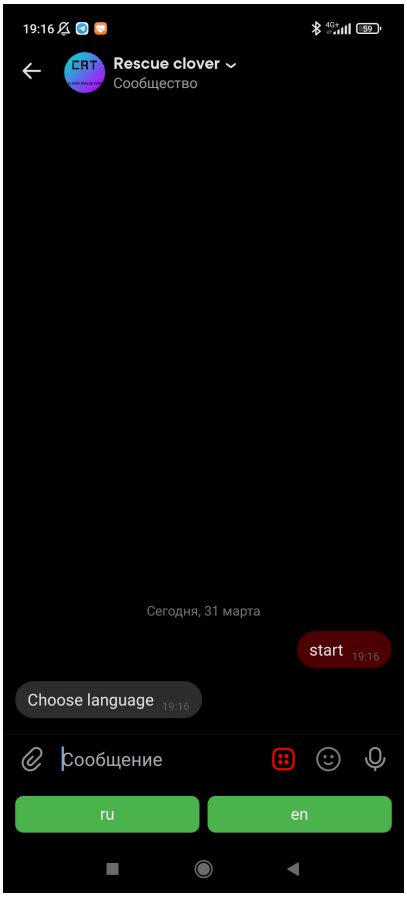

22:56
The Rescue Clover Proje...
48c5-94-29-124-254.eu.ngrok.io
Disconnected
Upload Mission
Choose code file
Upload & Run

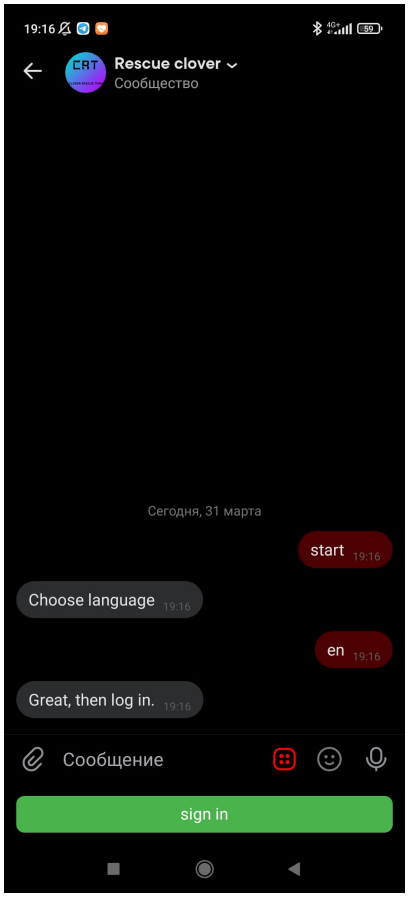

22:56

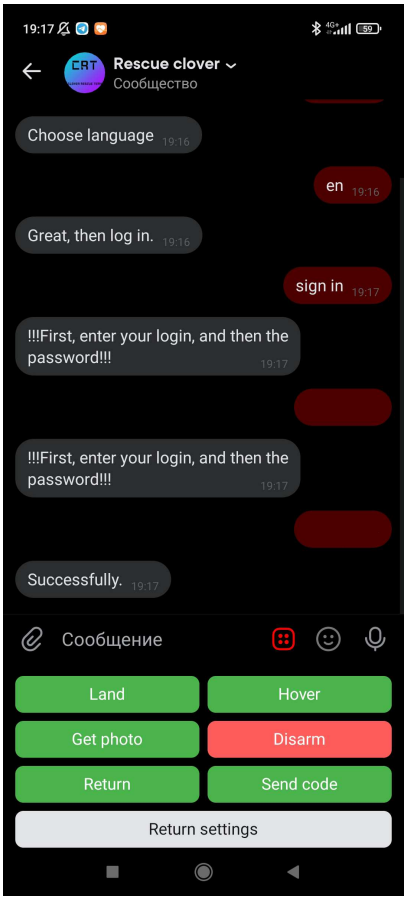


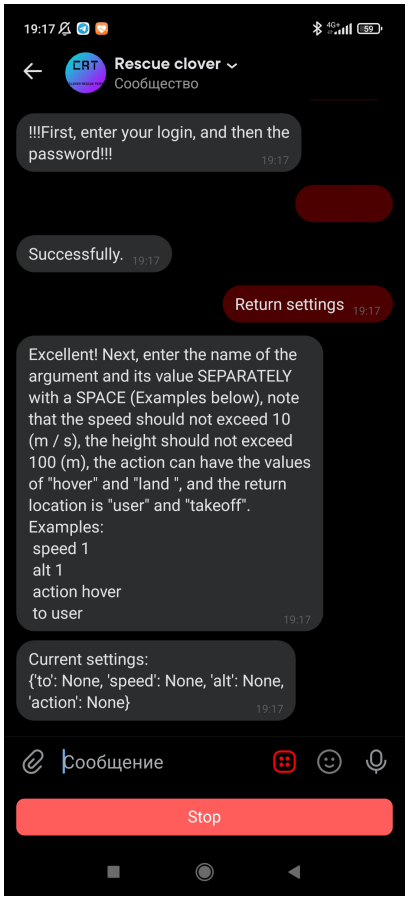
Bots on the social networks and messengers

Bots on the social networks is an example of what you can do based on our project. They have the main functionality of our website and The Clover Rescue Team is still working on their features.

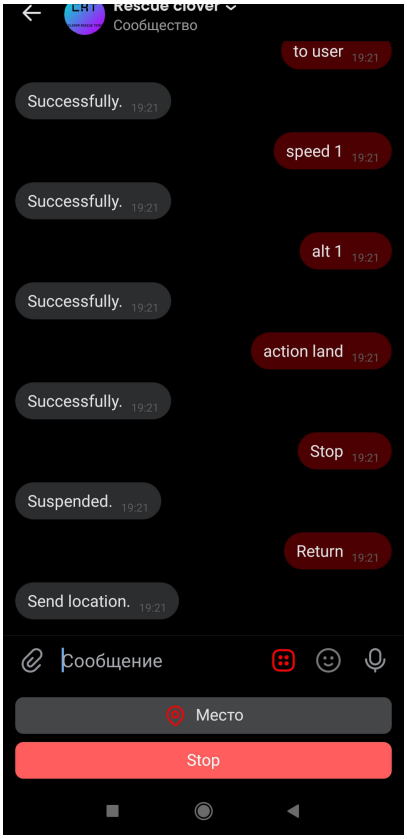

19:16
Rescue clover
Сообщество
Сегодня, 31 марта
start 19:16
Choose language 19:16
Сообщение
ru
en

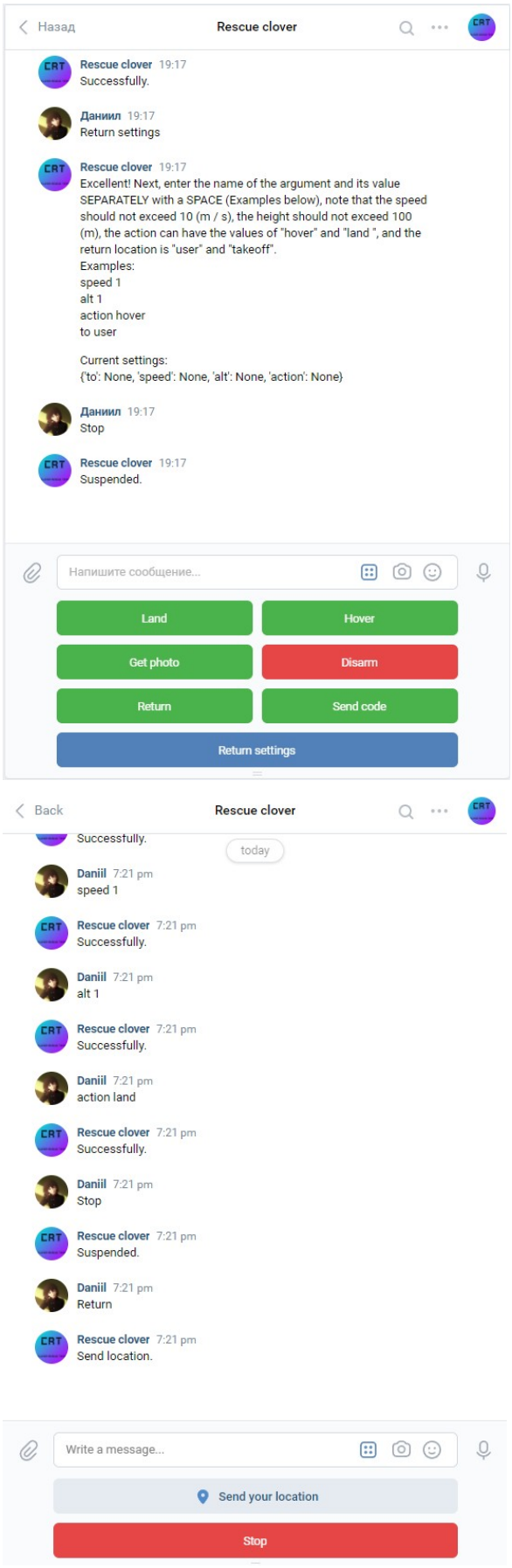

19:16
Rescue clover
Сообщество
Сегодня, 31 марта
start 19:16
Choose language 19:16
en 19:16
Great, then log in. 19:16
Сообщение
sign in


19:17
Rescue clover
Сообщество
Choose language 19:16
en 19:16
Great, then log in. 19:16
sign in 19:17
!!!First, enter your login, and then the password!!! 19:17
!!!First, enter your login, and then the password!!! 19:17
Successfully. 19:17
Сообщение
Land
Hover
Get photo
Disarm
Return
Send code
Return settings


19:17
Rescue clover
Сообщество
!!!First, enter your login, and then the password!!! 19:17
Successfully. 19:17
Return settings 19:17
Excellent! Next, enter the name of the argument and its value SEPARATELY with a SPACE (Examples below), note that the speed should not exceed 10 (m / s), the height should not exceed 100 (m), the action can have the values of "hover" and "land ", and the return location is "user" and "takeoff".
Examples:
speed 1
alt 1
action hover
to user 19:17
Current settings:
{'to': None, 'speed': None, 'alt': None, 'action': None} 19:17
Сообщение
Stop


19:21


Сообщество
to user 19:21
Successfully. 19:21
speed 1 19:21
Successfully. 19:21
alt 1 19:21
Successfully. 19:21
action land 19:21
Successfully. 19:21
Stop 19:21
Suspended. 19:21
Return 19:21
Send location. 19:21
Сообщение
Место
Stop



VK Bot Repository

Link to the bot

Demonstration video of the bot

Our plans for the future

We do not plan to stop and want to continue the development. Here is a list of what will be added.

1. Built-in code editor.
2. More drone control bots in social networks and messengers.
3. Socket API will be written so users can create their applications in different programming languages based on our app.
4. Drone swarm controls.

CopterCat_cm4

CopterHack-2022, команда **CopterCat**.



Team Information

Line-up:

- Lapin Matvey (https://t.me/l_motya), engineer/programmer.
- Konovalov Evgeny (https://t.me/egnknvlv), engineer/friend.
- Skandakov Egor (https://t.me/hjbaa), friend.
- Jalilov Emil, friend.

Project Description

Development of a modern board for PX4 FMUv6U firmware, 55x40 mm sizes and an additional WiFi module to implement cool things, for example, a distributed network.

Project idea

Flight controller on stm32h7 with space for RPi CM4 and built-in ESP32 to create a distributed network.

Planned results

FMUv6U flight controller board and API for interacting with a distributed network via RPi.

Using the "Clover" platform

At the stage of the project: debugging and demonstration of capabilities. After: using CopterCat as the main one.

Specification

FMU

- STM32H753IIK6 480Mhz Cortex-M7
- 2Mb + 256Kb FLASH
- 1Mb RAM
- ICM20602, ICM42605, BMI088, BMP388, BMM150
- Fully compliant with FMU-v6u standard

Raspberry Pi

- Support for RPi CM4 board.
- SD card slot.
- Ability to flash the built-in eMMC.
- CAT24C256 EEPROM.
- Support 2 cameras (CAM0 is two lines, CAM1 is four lines).
- USB-OTG support.

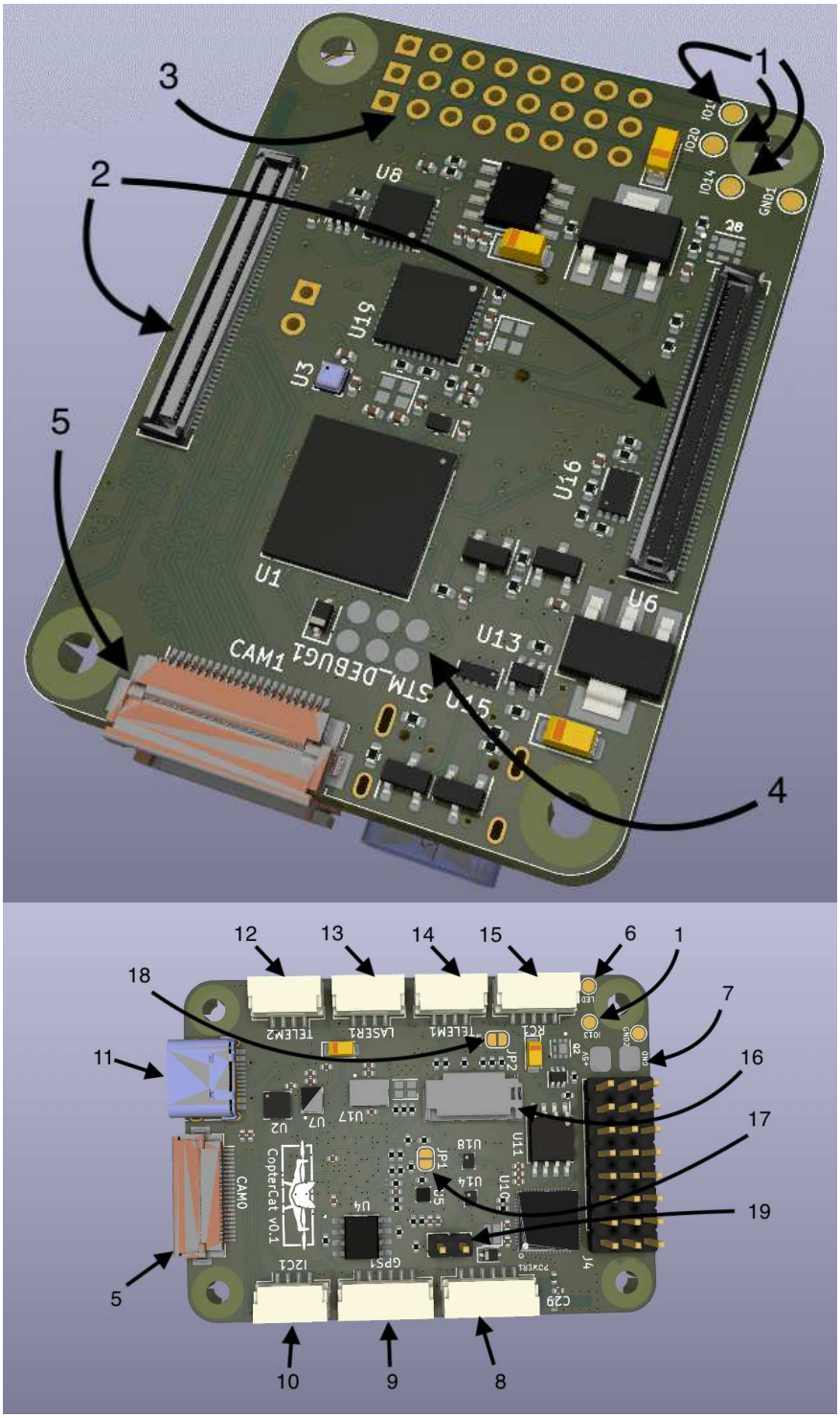
ESP32

- 16MB external FLASH (W25Q128JVS).
- 8MB external PSRAM (LY68L6400SLIT).
- Built-in antenna.
- USB-TTL converter.

Remaining

- USB-HUB USB2514B.
- USB-PD with physical switching.
- Communication between ESP32 and STM32 via UART.
- 3 power options.
- 4 universal GPIOs from ESP32.
- USB Type-C.
- Dimensions 40x55mm, board 4 layers.

Connectors and jumpers



1. GPIO ESP32 4 I/O ports for connecting external equipment.
2. RPi CM4 connectors.
3. ESC pins 8 pcs.
4. Programming and debugging pins for JTAG STM32.
5. Camera connectors (22 pin cable with 0.5 mm distance between conductors).
6. Contact for connecting the address tape.
7. Main power contacts 5V.
8. JST-6 standard PX4 power cable.
9. JST-6 GPS+compass+5V.

10. JST-4 I2C+5B.
11. USB Type-C.
12. JST-4 UART7+5B.
13. JST-4 I2C RPi+3.3B for connecting a rangefinder.
14. JST-4 UART5+5V.
15. JST-5 Standard connector for connecting the control receiver.
16. SD card slot (for RPi).
17. Jumper BOOT for STM32.
18. RPIBOOT jumper for flashing the RPi CM4 eMMC module.
19. Jumper for switching the USB connector operation mode (when the jumper is closed, USB works as a HUB input and when connected to a computer, STM32, ESP32 and RPi CM4 in OTG mode will be displayed; when the jumper is open, USB will work to connect external devices to the RPi, for example stereo cameras).

Firmware download

FMU

When you first start, the microcontroller will have to load the PX4-bootloader through the JTAG port. detailed instructions here.

To connect to a computer:

1. Close jumper 19.
2. Connect USB Type-C to your computer.
3. The device should appear in QGC.

You can also flash firmware via RPi:

1. Install the RPi CM4 into the connector on the board.
2. Open jumper 19.
3. The device will appear in the `/dev` folder on the RPi.

ESP32

You can write a program either in Arduino IDE or in VS Code with the plugin esp-idf. Then compile and upload to the microcontroller. You can download in two ways.

From computer:

1. Close jumper 19
2. CP2104 will appear in the connected devices
3. Download the firmware according to the instructions for the selected IDE

With RPi CM4:

1. Install the RPi CM4 into the connector on the board.
2. Open jumper 19.
3. Compile your code to .bin format.
4. Upload the resulting file to the RPi.

5. Download the firmware to the microcontroller using esptool.py description+installation.

RPi CM4

The SD card slot works like a standard RPi. For boards with eMMC, the boot order of the operating system does not differ from the CM4 IO Board instruction.

General information

- All files required for ordering are located in the `/gerbers` folder here.
- The project was made in the program KiCAD v6.
- Component Libraries are from snapeda.

Ikshana — Autonomous valet parking drone assistance

CopterHack-2022, team **DJS Phoenix**.

Team information



We are the DJS Phoenix, the official drone team of Dwarkadas. J. Sanghvi College of Engineering.

The list of team members:

- Shubham Mehta, shubhamdmehta3257@gmail.com, Team Lead.
- Akhilesh Sadalgekar, akhilesh.sadalgekar01@gmail.com, Mechanical.
- Aman Bhatt, aman.bhatt0001@gmail.com, Mechanical.
- Manan Dedhia,manandedhia2001@gmail.com, Mechanical.
- Harshal Warde, harshal.warde@gmail.com, Mechanical.
- Karan Pandit, karandpandit26@gmail.com, Mechanical.
- Soham P Dalvi, dalvisoham710@gmail.com, Mechanical.
- Khushi Sanghvi, khushisanghvi940@gmail.com, Programming.
- Ankit Sawant, ankitsawant26@gmail.com, Electronics.
- Shubh Jatin Pokarne, shubhpokarne91@gmail.com, Electronics.
- Parth Sawjiyani, sawjiyaniparth@gmail.com, Marketing.

Project description

Ikshana is a fully autonomous drone that operates in a parking lot. It scans for available parking places and then guide drivers by directing them to an optimal unoccupied parking location using LED Blinkers.

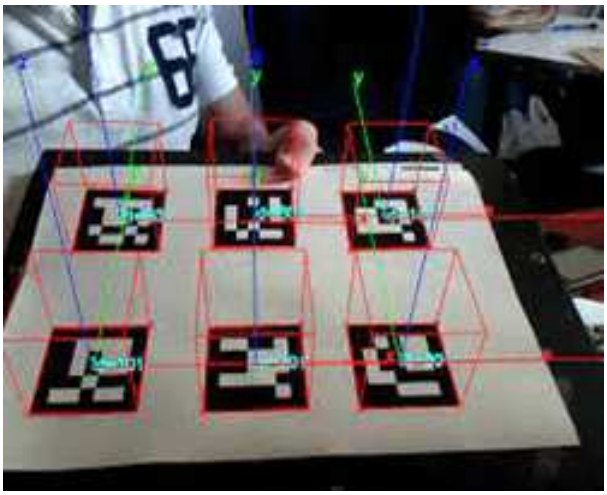
Project idea

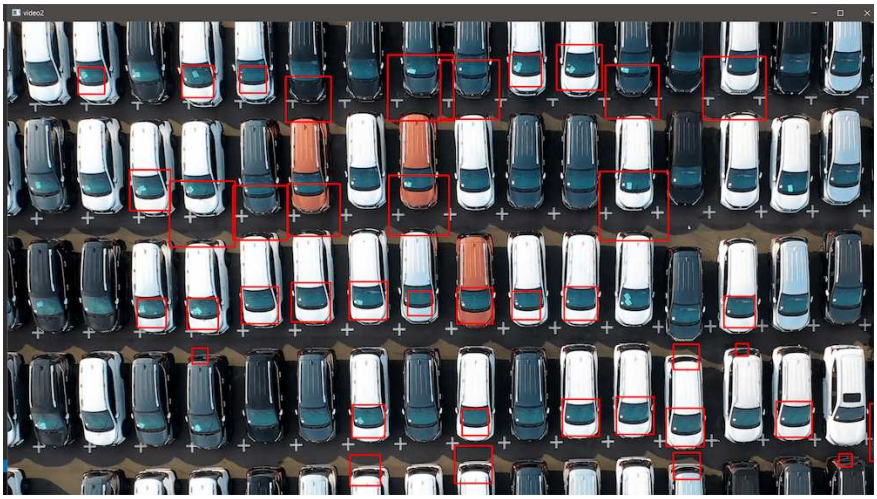
We came up with the concept of using drones to search for parking spots, this alleviates the problem of having to do it manually which is tedious and time consuming. It will help us save time and effort.

The driver will be able to see our drone, which will lead the vehicle to the parking location.

The drone's arms include programmable LEDs. By blinking in a relay pattern, these LEDs will direct you down the path, whether to turn right, left, continue or stop.

The drone is equipped with sensors to maintain a safe distance in all direction and avoid obstacles.





The potential outcomes

Problem

Getting around parking lots can be annoying at times and finding a vacant spot in the parking lot is a time-consuming and difficult task.

Solution

Ikshana will help the driver reach an ideal parking spot. With the help of Machine Learning and ROS it will be able to find spaces for vehicle parking.

Such a system is really helpful in parking lots as it avoids hassles and commotion while also reducing labor, being more efficient and time saving.

By the end of the project, we expect a DIY drone that has autonomous functionality and it is eligible to perform the given task efficiently.

Using Clover platform

The COEX Clover platform is used for simulating and implementing OpenCV, ROS for the drone. It helps us integrate ROS with our Raspberry Pi.

Additional information at the request of participants

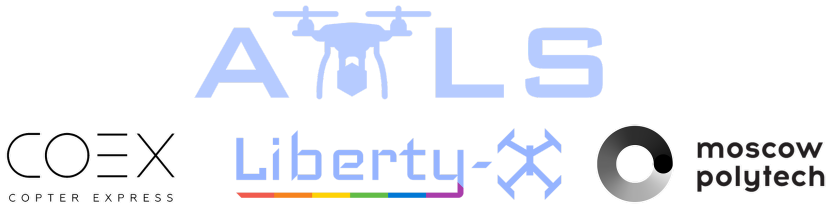
After two years after the COVID-19 epidemic, there is a good opportunity for the team to reunite. Previously, we spent the most of our time in our workshop.

However, we learn to work online through various online platforms. Although procuring parts was time-consuming because very few suppliers were prepared to ship their items, and even after placing orders, delivery of those parts might take months. So, finally, we are able to collaborate and solve difficulties in a timely manner. This journey of Copterhack'22 will be full of new insights and experiences.

To find out more about our project visit the link below

https://djs-phoenix.gitbook.io/ikshana/.

Autonomous Multirotor Landing System (AMLS)



The goal is to automatically land a drone on a moving platform

AMLS Article

In this Article we will describe AMLS project. Namely, AMLS Optical stabilization, GPS holding, GPS following, Altitude holding, Grabbing, Weather protection, Speed measurement and Illumination systems. In addition, we will make clear of how it works and how it was done!

Our main GitHub repository

https://github.com/XxOinvizioNxX/Liberty-Way

Developers

- Pavel Neshumov
- Andrey Kabalin
- Vladislav Yasnetsky



Table of contents

- 0. How does it work?

- 0.1. A video about our project
- 1. GPS hold and Flight to waypoints functions
  - 1.1. Serial reading
  - 1.2. UBlox GPS parsing
  - 1.3. Set current waypoint
  - 1.4. Waypoint edition (To fly to waypoints)
  - 1.5. Waypoint stabilization
- 2. GPS following
- 3. Compass
- 4. Altitude stabilization (barometer)
- 5. Optical stabilization
  - 5.1. So difficult and so important
  - 5.2. First steps
  - 5.3. Inverse approach
  - 5.4. Java edition
  - 5.5. Liberty-Way
  - 5.6. Communication with the drone
  - 5.7. Camera gimbal
- 6. Eitude AMLS Platform
  - 6.1. Grabbing system
  - 6.2. Weather protection system
  - 6.3. Speed measurement system
  - 6.4. Illumination system
- 7. Conclusion

---

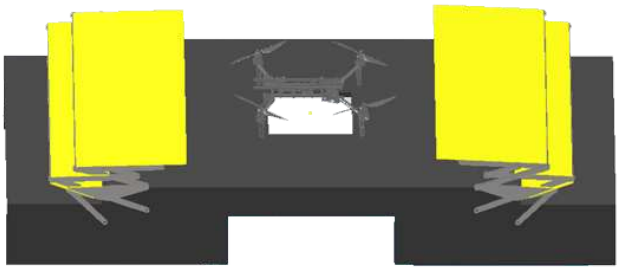
0. How does it work?

The AMLS system consists of two parts:

- The drone



- And the platform either mobile (implemented on a vehicle), either stable (pick-up-point)

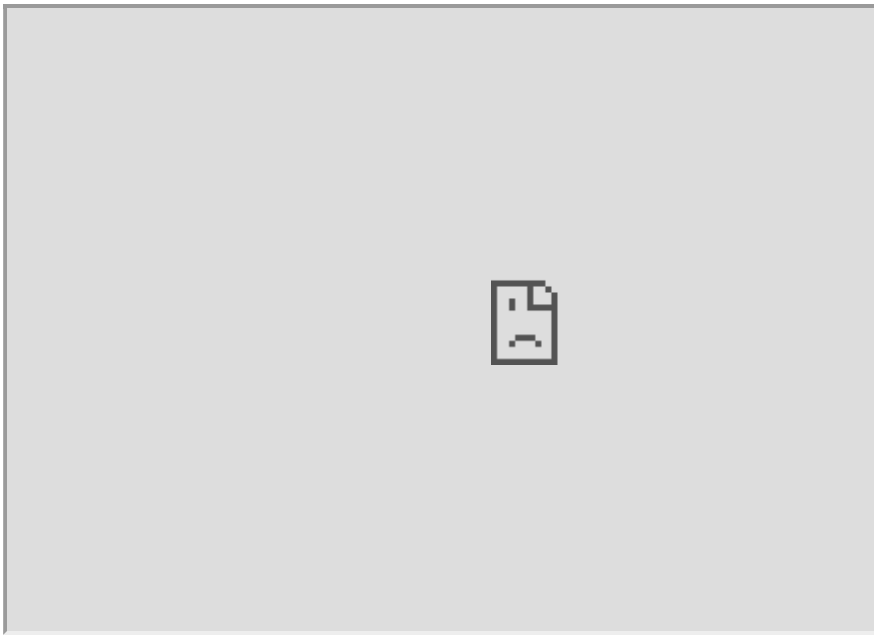


How the system operates:

- Firstly, a drone with a delivery package is far from the platform and it has no visual contact with it. The drone recieves GPS coordinates of a platform by using cellular communication or any other radio channel (The drone has Liberty-Link implemented on it. This module is able to adjust its position, whatever the firmware of the flight controller. The module is installed inside the line between a receiver and a flight controller.
- The drone is moving to received coordinates. The coordinates might be renewed in the process (but not frequently, thus preventing the channel from overloading)
- When the drone is close to the platform but there is still no visual contact, the program runs GPS stabilization. Here the data is being transmitted over the closest radio communication channel of high freqency, so the drone can catch up with the platform.
- Meanwhile, the drone descends (barometers are installed on both, the drone and the platform). Descending goes on untill altitude reaches 1.5-2 meters above the platform.
- While descending and when visual contact with the platform camera is established, the program enables visual (precision) stabilization. And as soon as the drone's tag is within camera's field of view, the algorithm will capture the drone.
- When optical stabilization is enabled, GPS is working as a back up plan (in case something goes wrong, GPS stabilization launches again).
- In order to use optical stabilization the drone is equipped with ArUco tag which can be captured by a camera and by using the closest radio communication channel, the system transmits adjustment data to the drone.
- Along with optical stabilization, the program launches landing algorithm. The algorithm artificially and smoothly reduces the setpoint of height (Z) until it reaches a certain threshold.
- When the drone is approaching on the desirable height, the program enables grabbing system implemented on the platform. Those grips are used to catch and hold the drone in the process of landing and after the drone was caught.
- When the landing is completed, the platform starts maintenance work and in order to protect the drone frome external influences, the program enables

weather protection and closes the roof above landing area.
- Landing accomplished!

Short video about our project (clickable)



1. GPS hold and Flight to waypoints functions

At the beginning, the drone with the package is far from the platform. Then via cellular communication or another suitable radio channel, platform GPS coordinates are sent to the drone (the Liberty-Link module is installed on the drone, this module is capable of correcting its position, regardless of the firmware of the flight controller. (The module is placed between the receiver (RC) and the flight controller)

GPS module will be built in Liberty-Link, so it would have the ability to maintain the drone's GPS position and follow GPS points.

GPS-module will be used from the UBlox group (for instance, UBlox Neo-M8). There will be 1 or 3 (to minimize the error) modules.



Modules operate via UART, configured to send data 5 times per second. The Liberty-Link firmware will read data from the modules and calculate the coordinates of the current position.

1.1. Serial reading

Reading data from a module into a buffer looks like this:

```
// Read data from the GPS module
while (GPS_serial.available() && new_line_found == 0) {
    // Stay in this loop as long as there is serial information from the GPS av
    char read_serial_byte = GPS_serial.read();
    if (read_serial_byte == '$') {
        // Clear the old data from the incoming buffer array if the new byte ec
        for (message_counter = 0; message_counter <= 99; message_counter++) {
            incoming_message[message_counter] = '-';
        }
        // Reset the message_counter variable because we want to start writing
        message_counter = 0;
    }
    // If the received byte does not equal a $ character, increase the message_
    else if (message_counter <= 99)
        message_counter++;

    // Write the new received byte to the new position in the incoming_message
    incoming_message[message_counter] = read_serial_byte;

    // Every NMEA line ends with a '*'. If this character is detected the new_
    if (read_serial_byte == '*') new_line_found = 1;
}
```

1.2. UBlox GPS parsing

After that, latitude, longitude, a type of correction (2D, 3D) and the number of satellites are calculated from the filled buffer. Parsing GPS data of the UBlox protocol looks like this:

```
// If the software has detected a new NMEA line it will check if it's a valid
if (new_line_found == 1) {
    // Reset the new_line_found variable for the next line
    new_line_found = 0;
    if (incoming_message[4] == 'L' && incoming_message[5] == 'L' && incoming_me
        // When there is no GPS fix or latitude/longitude information available
        // Set some variables to 0 if no valid information is found by the GPS
        l_lat_gps = 0;
        l_lon_gps = 0;
        lat_gps_previous = 0;
        lon_gps_previous = 0;
        number_used_sats = 0;
    }
    // If the line starts with GA and if there is a GPS fix we can scan the lin
    if (incoming_message[4] == 'G' && incoming_message[5] == 'A' && (incoming_m
        // Filter the minutes for the GGA line multiplied by 10
        lat_gps_actual = ((int)incoming_message[19] - 48) * (long)10000000;
        lat_gps_actual += ((int)incoming_message[20] - 48) * (long)1000000;
        lat_gps_actual += ((int)incoming_message[22] - 48) * (long)100000;
        lat_gps_actual += ((int)incoming_message[23] - 48) * (long)10000;
        lat_gps_actual += ((int)incoming_message[24] - 48) * (long)1000;
        lat_gps_actual += ((int)incoming_message[25] - 48) * (long)100;
        lat_gps_actual += ((int)incoming_message[26] - 48) * (long)10;
        // To convert minutes to degrees we need to divide minutes by 6
        lat_gps_actual /= (long)6;
        // Add multiply degrees by 10
        lat_gps_actual += ((int)incoming_message[17] - 48) * (long)100000000;
        lat_gps_actual += ((int)incoming_message[18] - 48) * (long)10000000;
        // Divide everything by 10
        lat_gps_actual /= 10;

        // Filter minutes for the GGA line multiplied by 10
        lon_gps_actual = ((int)incoming_message[33] - 48) * (long)10000000;
        lon_gps_actual += ((int)incoming_message[34] - 48) * (long)1000000;
        lon_gps_actual += ((int)incoming_message[36] - 48) * (long)100000;
        lon_gps_actual += ((int)incoming_message[37] - 48) * (long)10000;
        lon_gps_actual += ((int)incoming_message[38] - 48) * (long)1000;
        lon_gps_actual += ((int)incoming_message[39] - 48) * (long)100;
        lon_gps_actual += ((int)incoming_message[40] - 48) * (long)10;
        // To convert minutes to degrees we need to divide minutes by 6
        lon_gps_actual /= (long)6;
        // Add multiply degrees by 10
        lon_gps_actual += ((int)incoming_message[30] - 48) * (long)1000000000;
        lon_gps_actual += ((int)incoming_message[31] - 48) * (long)100000000;
        lon_gps_actual += ((int)incoming_message[32] - 48) * (long)10000000;
        // Divide everything by 10
        lon_gps_actual /= 10;

        if (incoming_message[28] == 'N')
            // When flying north of the equator the latitude_north variable wi
            latitude_north = 1;
        else
            // When flying south of the equator the latitude_north variable wi
            latitude_north = 0;

        if (incoming_message[42] == 'E')
            // When flying east of the prime meridian the longiude_east variab
            longiude_east = 1;
        else
            // When flying west of the prime meridian the longiude_east variab
            longiude_east = 0;

        // Filter the number of satillites from the GGA line
        number_used_sats = ((int)incoming_message[46] - 48) * (long)10;
        number_used_sats += (int)incoming_message[47] - 48;
```

```
        if (lat_gps_previous == 0 && lon_gps_previous == 0) {
            // If this is the first time the GPS code is used
            // Set the lat_gps_previous variable to the lat_gps_actual variable
            lat_gps_previous = lat_gps_actual;
            // Set the lon_gps_previous variable to the lon_gps_actual variable
            lon_gps_previous = lon_gps_actual;
        }

        // Divide the difference between the new and the previous latitudes by
        lat_gps_loop_add = (float)(lat_gps_actual - lat_gps_previous) / 10.0;
        // Divide the difference between the new and the previous longitudes by
        lon_gps_loop_add = (float)(lon_gps_actual - lon_gps_previous) / 10.0;

        // Set the l_lat_gps variable to the previous latitude value
        l_lat_gps = lat_gps_previous;
        // Set the l_lon_gps variable to the previous longitude value
        l_lon_gps = lon_gps_previous;

        // Remember the new latitude value in the lat_gps_previous variable for
        lat_gps_previous = lat_gps_actual;
        // Remember the new longitude value in the lat_gps_previous variable fo
        lon_gps_previous = lon_gps_actual;
    }

    // If the line starts with SA and if there is a GPS fix we can scan the lin
    if (incoming_message[4] == 'S' && incoming_message[5] == 'A')
        fix_type = (int)incoming_message[9] - 48;

}
```

1.3. Set current waypoint

When required data is received the main magic happens. To enable maintaining of the current position it will be enough to set the flag `waypoint_set = 1;` and set current coordinates as a waypoint:

```
l_lat_waypoint = l_lat_gps;
l_lon_waypoint = l_lon_gps;
```

After that, the calculation of the error in the coordinates will begin, correction works with the help of a PD - regulator. For D - component we use rotation memory.

1.4. Waypoint edit (To fly to waypoints)

If you just set the new `l_lat_waypoint` and `l_lon_wayoint` , which are located at a great distance from the drone, the drone will not be able to fly normally and stabilize at these coordinates. For smooth adjustments `l_lat_gps_float_adjust` and `l_lon_gps_float_adjust` can be used. These are `float` variables, changing which will smoothly shift `l_lat_waypoint` and `l_lon_waypoint` .

For example, if in the main loop you will constantly add a certain value to these variables:

```
l_lat_gps_float_adjust += 0.0015;
```

With set waypoint, the drone will move smoothly in the given direction. In the future, this will be used for the smooth drone's acceleration and deceleration while moving to its destination.

1.5. Waypoint stabilization

```
if (waypoint_set == 1) {
    //If the waypoints are stored

    // Adjust l_lat_waypoint
    if (l_lat_gps_float_adjust > 1) {
        l_lat_waypoint++;
        l_lat_gps_float_adjust--;
    }
    if (l_lat_gps_float_adjust < -1) {
        l_lat_waypoint--;
        l_lat_gps_float_adjust++;
    }

    // Adjust l_lon_waypoint
    if (l_lon_gps_float_adjust > 1) {
        l_lon_waypoint++;
        l_lon_gps_float_adjust--;
    }
    if (l_lon_gps_float_adjust < -1) {
        l_lon_waypoint--;
        l_lon_gps_float_adjust++;
    }

    // Calculate the latitude error between waypoint and actual position
    gps_lon_error = l_lon_waypoint - l_lon_gps;
    // Calculate the longitude error between waypoint and actual position
    gps_lat_error = l_lat_gps - l_lat_waypoint;

    // Subtract the current memory position to make room for the new value
    gps_lat_total_avarage -= gps_lat_rotating_mem[gps_rotating_mem_location];
    // Calculate the new change between the actual pressure and the previous me
    gps_lat_rotating_mem[gps_rotating_mem_location] = gps_lat_error - gps_lat_e
    // Add the new value to the long term average value
    gps_lat_total_avarage += gps_lat_rotating_mem[gps_rotating_mem_location];

    // Subtract the current memory position to make room for the new value
    gps_lon_total_avarage -= gps_lon_rotating_mem[gps_rotating_mem_location];
    // Calculate the new change between the actual pressure and the previous me
    gps_lon_rotating_mem[gps_rotating_mem_location] = gps_lon_error - gps_lon_e
    // Add the new value to the long term avarage value
    gps_lon_total_avarage += gps_lon_rotating_mem[gps_rotating_mem_location];

    // Increase the rotating memory location
    gps_rotating_mem_location++;
    if (gps_rotating_mem_location == 35)
        // Start at 0 when the memory location 35 is reached
        gps_rotating_mem_location = 0;

    // Remember the error for the next loop
    gps_lat_error_previous = gps_lat_error;
    gps_lon_error_previous = gps_lon_error;

    //Calculate the GPS pitch and roll corrections as if the nose of the multic
    //The Proportional part = (float)gps_lat_error * gps_p_gain.
    //The Derivative part = (float)gps_lat_total_avarage * gps_d_gain.
    gps_pitch_adjust_north = (float)gps_lat_error * gps_p_gain + (float)gps_lat
    gps_roll_adjust_north = (float)gps_lon_error * gps_p_gain + (float)gps_lon_

    if (!latitude_north)
        // Invert the pitch adjustmet because the quadcopter is flying south of
        gps_pitch_adjust_north *= -1;
    if (!longiude_east)
        // Invert the roll adjustmet because the quadcopter is flying west of t
        gps_roll_adjust_north *= -1;
```

```
    //Because the correction is calculated as if the nose was facing north, we
    gps_roll_adjust = ((float)gps_roll_adjust_north * cos(angle_yaw * 0.017453
    gps_pitch_adjust = ((float)gps_pitch_adjust_north * cos(angle_yaw * 0.0174

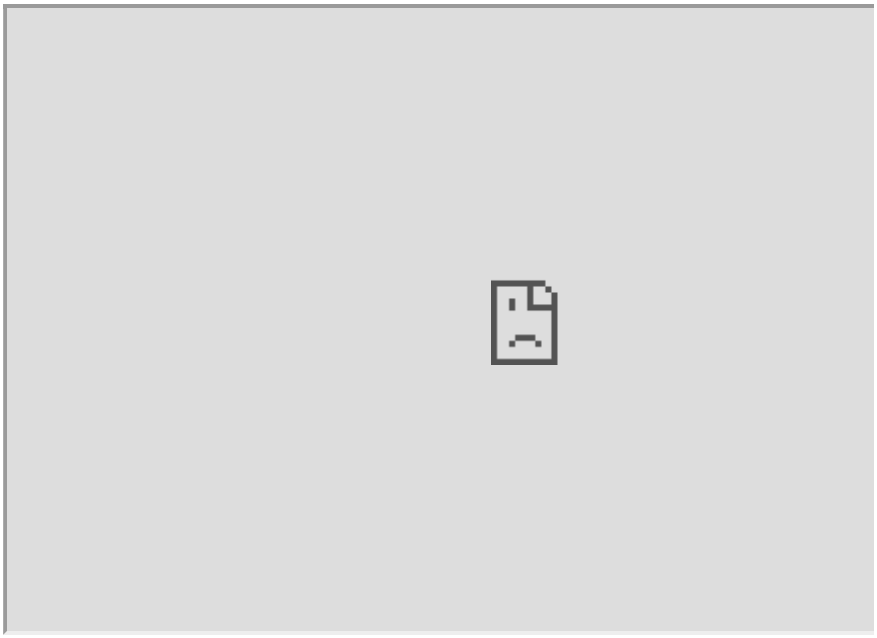
    //Limit the maximum correction to 300. This way we still have the full cont
    if (gps_roll_adjust > 300) gps_roll_adjust = 300;
    if (gps_roll_adjust < -300) gps_roll_adjust = -300;
    if (gps_pitch_adjust > 300) gps_pitch_adjust = 300;
    if (gps_pitch_adjust < -300) gps_pitch_adjust = -300;
}
```

2. GPS following

The main part of stabilization using GPS coordinates was the development of an algorithm for predicting the position of the drone. The simplest idea was to use a mathematical calculation of the next drone's position. This is calculated for the most accurate positioning in relation to the landing platform.

At the beginning we developed a simple algorithm for calculating the coefficient of coordinates' change. The development was done using Python. At the stage of testing this algorithm, the problem of simulating the generation of GPS coordinates arose. To solve this problem, many different resources were used: from open source homemade navigators to trying to use the Google Maps, Yandex Maps or 2GIS APIs. And only after 3 months, we thought of a simple solution: to change the values with some delta and to visualize using MatPlotLib or PyQtGraph. Prior to this, all testing of the algorithm was carried out using the PX4 firmware toolkit and the Gazebo drone motion simulator. As a result, many formalities were overcome in terms of communicating with the simulator and increasing performance (click on the picture to see the video).

The result of the GPS prediction (clickable):



The end result reached a point when the error of the predicted position varies from 0 to 70 centimeters.

3. Compass

Before optical stabilization launches (during GPS stabilization process), to calculate the GPS correction vector, you need to know the exact angle from the compass. For this, a compass built into the GPS module is used.

Because during the flight, the roll and pitch angles change and a user needs to correct the values from the compass. In general, calculating the angle from the compass looks like this:

```
// The compass values change when the roll and pitch angles of the quadcopter c
// The 0.0174533 value is phi/180 as the functions are in radians in stead of d
compass_x_horizontal = (float)compass_x * cos(angle_pitch * -0.0174533) + (floa
compass_y_horizontal = (float)compass_y * cos(angle_roll * 0.0174533) + (float)

// Now that the horizontal values are known the heading can be calculated. With
// Please note that the atan2 uses radians in stead of degrees. That is why the
if (compass_y_horizontal < 0)actual_compass_heading = 180 + (180 + ((atan2(comp
else actual_compass_heading = (atan2(compass_y_horizontal, compass_x_horizontal

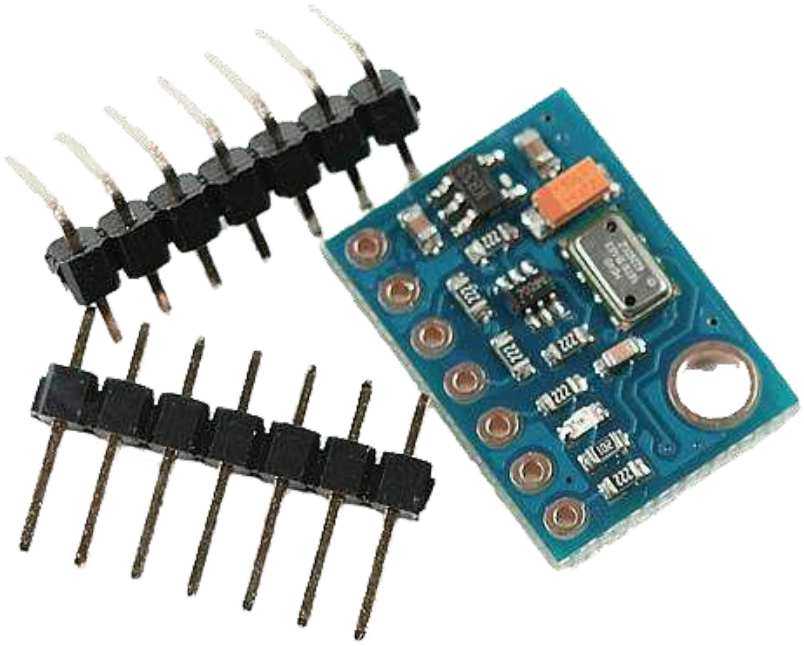
// Add the declination to the magnetic compass heading to get the geographic no
actual_compass_heading += declination;
// If the compass heading becomes smaller than 0, 360 is added to keep it in th
if (actual_compass_heading < 0) actual_compass_heading += 360;
// If the compass heading becomes larger then 360, 360 is subtracted to keep it
else if (actual_compass_heading >= 360) actual_compass_heading -= 360;
```

It is clear that the angle from the compass can also be used to maintain the yaw angle of the drone. With point-to-point flights, this may be realized. But at the moment, there is no urgent need for this, because after the start of optical stabilization, the algorithm is able to correct the drone regardless of its yaw angle. Also, during optical stabilization, the yaw angle is automatically corrected.

4. Altitude stabilization (barometer)

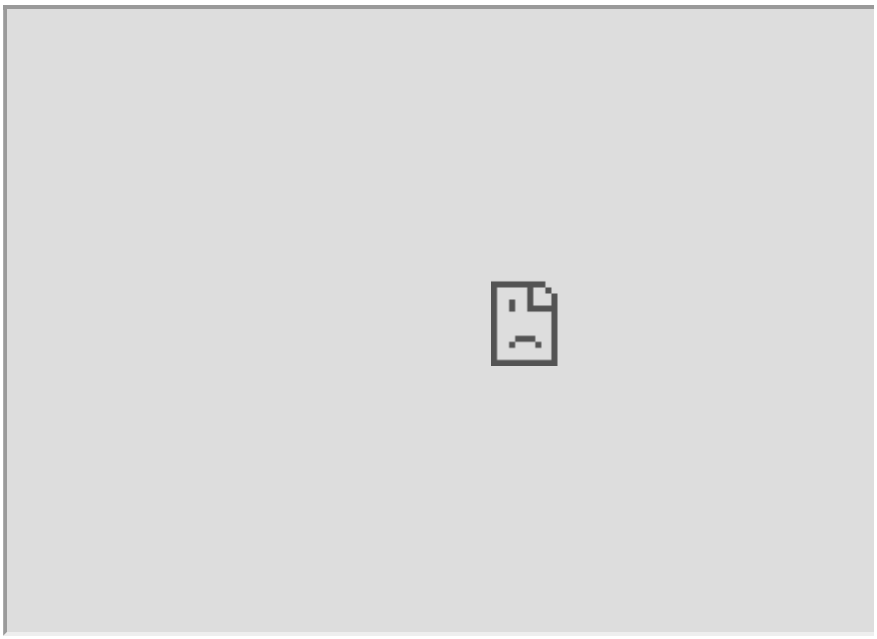
Before optical stabilization launches (during GPS stabilization process), our Liberty-Link module will be able to maintain altitude using a barometer.

The platform, as well as the Liberty-Link, will have MS5611 barometers.



According to the documentation, the height resolution is 10 cm. The algorithm will take the pressure values and by passing it through the PID-controller will stabilize the drone's altitude by changing the Throttle (3rd channel).

Altitude hold test (clickable):



During the flight along the waypoint, the setpoint of the pressure will decrease in order to increase the altitude (it is safer to fly in a straight line at a high altitude, so the drone would not crash into anything). And during GPS stabilization (when the

drone is already close to the platform), the drone will be set with a setpoint of pressure that correlates with ~ 1.5-2m height above the platform.

5. Optical stabilization

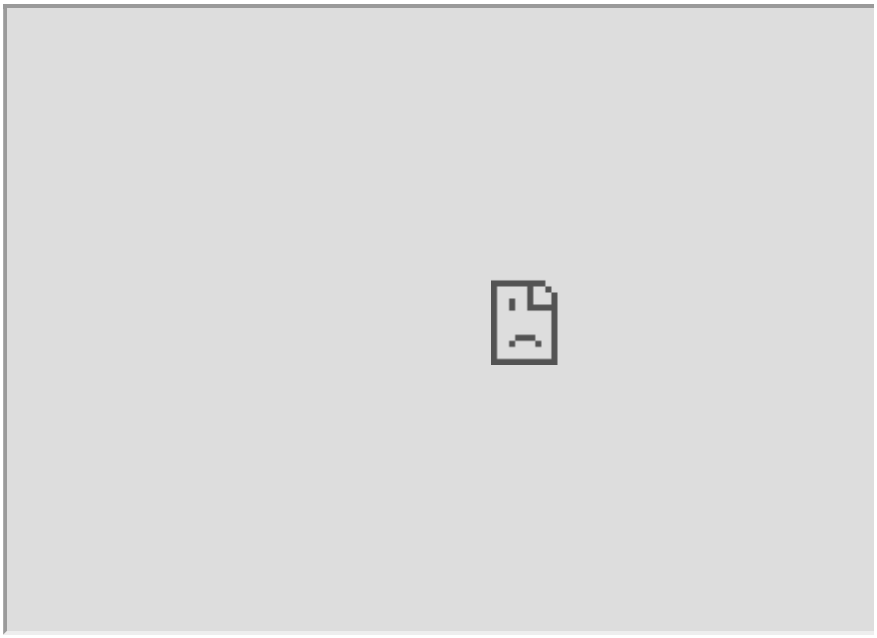
5.1. So difficult and so important

Optical stabilization is the most important and challenging part of our project. In the current condition it is possible to keep the drone above the platform still only using these algorithms. The current version of the OIS algorithm, along with a description of the usage and making, is available in our main GitHub repository. In the future, GPS stabilization will be added to it.

5.2. First steps

And as we couldn't predict the possibility of accomplishing our task, first of all, we started to think about the solution for the stabilization system. Afterwards, we settled with the stabilization using augmented reality tags. Firstly, it won't take much finances as we do not need GPS or RTK systems and it will be accurate enough to accomplish its purpose. Our first idea was to attach Raspberry Pi with Liberty_X as it was embodied in COEX Clover and to let Raspberry Pi handle all of the maths.

First optical stabilization prototype test (clickable):



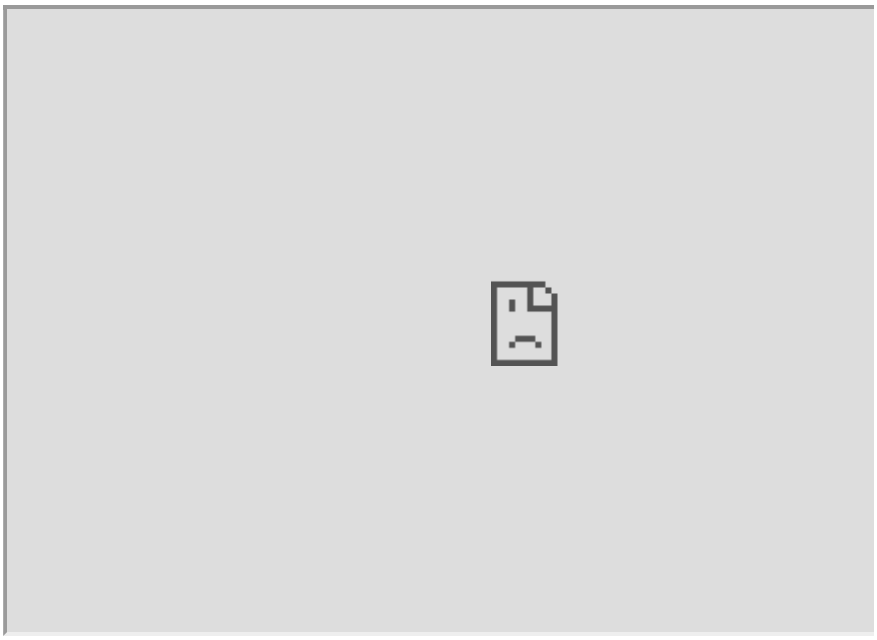
But few tests showed that Raspberry Pi computing power is not enough for amount of data needed to stabilize the drone. Furthermore, the idea of installing a Raspberry Pi on each drone is irrational for its own.

Also, we had middling prototypes, for example, attempts to use color markers (circles of different colors), but these ideas did not work well enough.

5.3. Inverse approach

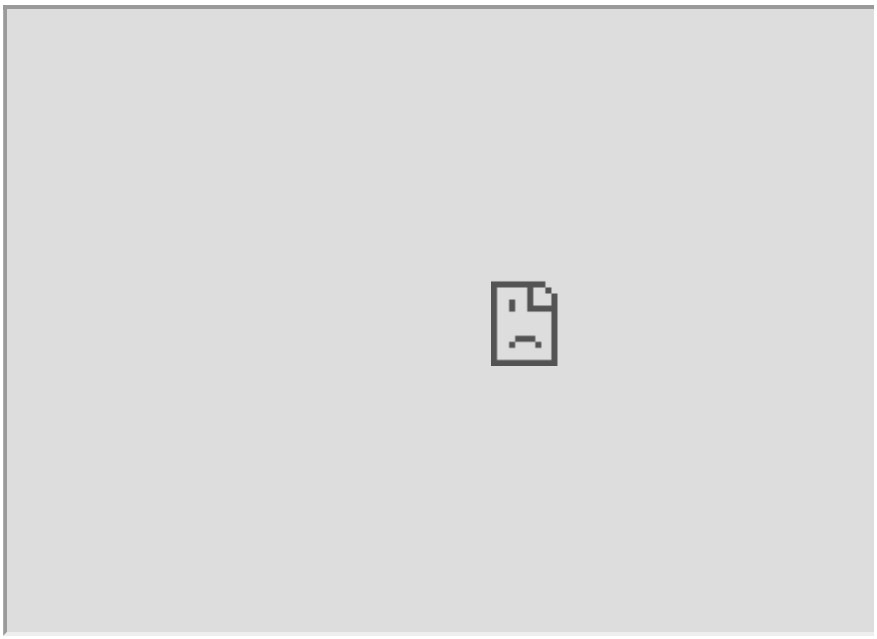
Then we came up with the idea of separating drone and stabilization system so the main math will be accomplished on the landing platform with powerful machine.

This was how we ended up with our current optical stabilization algorithm - the camera which is connected to a powerful machine and the machine is attached to the platform. The drone only has 4x4 ArUco tag and its controller.



Then, we came up with using Pose Estimation algorithms from OpenCV library. The first tests showed us that we are on the right track!

Pose Estimation Python (clickable):



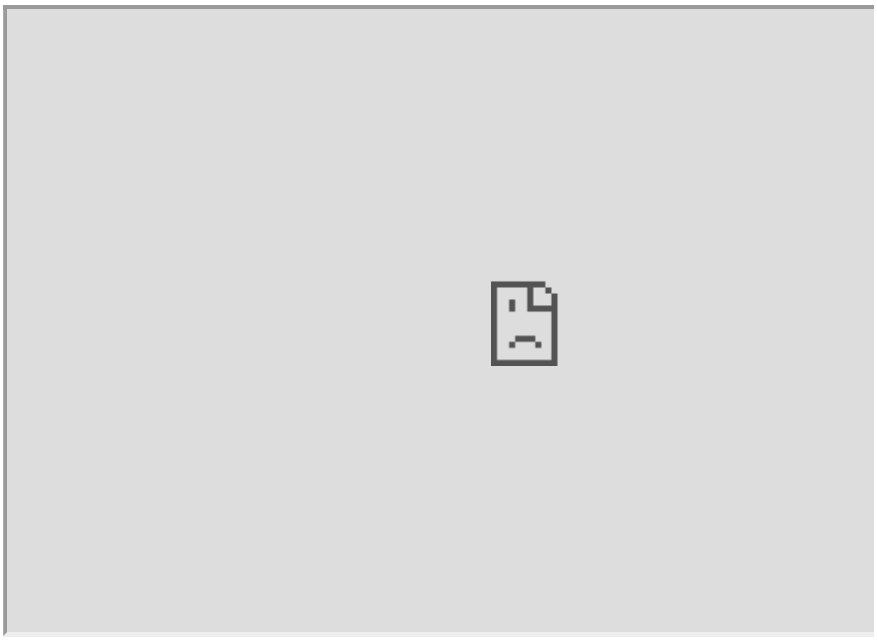
But, the algorithms were far from perfect. For example, since the code was written in Python (https://github.com/XxOinvizioNxX/Liberty-X_Point), the performance was not satisfyingly, and there was no suitable threading control either. Therefore,

we had to change something.

5.4. Java edition

Having weighed all the pros and cons, it was decided to rewrite all optical stabilization using Java. This is how the first version of Liberty-Way appeared. This time, it was decided to approach the OOP thoroughly, and, after a little tweaking, an excellent stabilization and landing algorithm was obtained.

Landing test on Liberty-Way v.beta_0.0.1 (clickable):

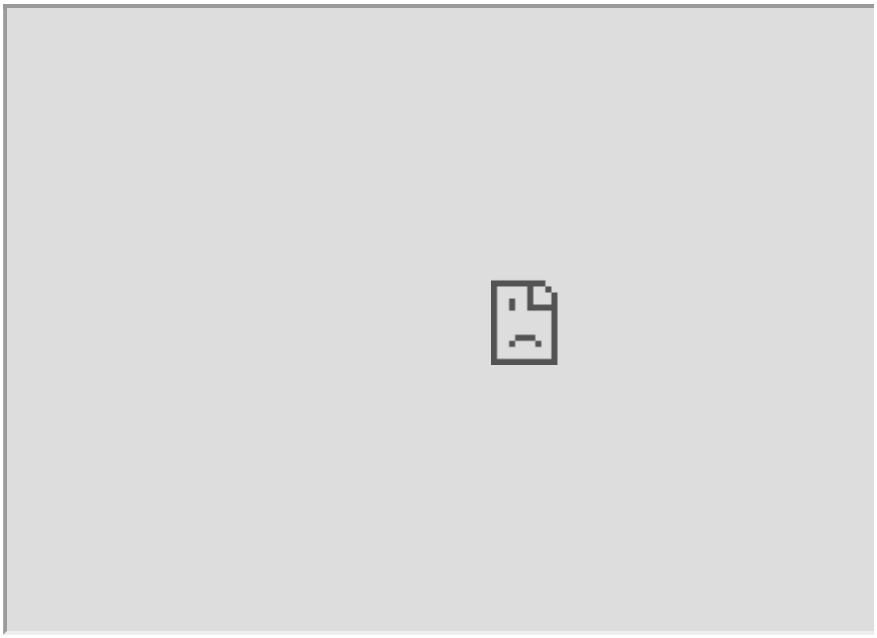


5.5. Liberty-Way

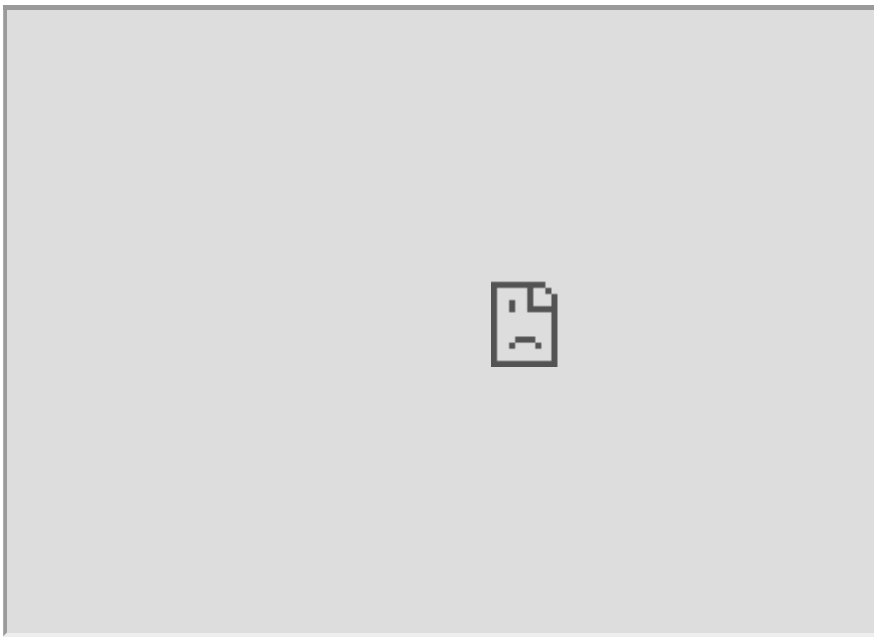
Then many improvements and bug fixes followed. As a result, Liberty-Way is a cross-platform web sarvar application that is very convenient for configuration and debugging. Also, in the latest versions (beta_1.0.3 - beta_1.1.2) a blackbox feature was introduced (for recording logs), as well as communication with the platform and many other necessary algorithms.

Full description, including all settings, startup, etc. you can find in our GitHub repository: https://github.com/XxOinvizioNxX/Liberty-Way

Video of static stabilization (clickable):

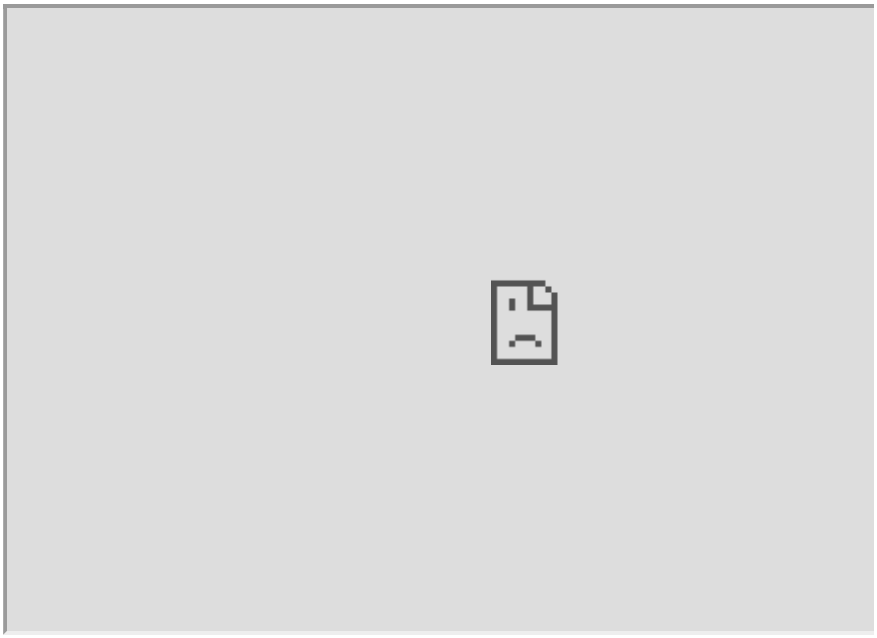


Liberty-Way can even stabilize a "thrown" drone (clickable):



There is a small bug in the video with the rotation angle, in the new release it has been fixed!

And, of course, example of how it works in motion (tested with beta_0.0.3 release) (clickable):



All basic settings are conveniently placed in separate JSON files (settings, PID), which allows a user to quickly change the required parameters without rebuilding the application. In fact, to run the application, you just need to download the latest release, unpack the archive and run it through the launcher corresponding to the preferable OS.

5.6. Communication with the drone

The Liberty-Way connects to the Liberty-Link module installed on the drone and adjusts its position by directly controlling four main channels of the remote control. In one cycle (each frame from the camera), 12 bytes of correction data are sent to the module:

Byte N	0	1	2	3	4	5	6	7	8	9	10	11
Description	Roll byte 1	Roll byte 2	Pitch byte 1	Pitch byte 2	Yaw byte 1	Yaw byte 2	Altitude byte 1	Altitude byte 2	Service Info	Check byte	Data suffix 1 (L)	Data suffix 2 (X)
Structure	1000-2000		1000-2000		1000-2000		1000-2000		0-3	XOR check sum	L	X
Example in HEX	0x05	0xDC	0x05	0xDC	0x05	0xDC	0x05	0xDC	0x00	0x00	0x4C	0x58
Example in DEC	1500		1500		1500		1500		0	0	76	88

Bytes description:

- **Roll bytes** - Roll correction values
- **Pitch bytes** - Pitch correction values
- **Yaw bytes** - Yaw correction values
- **Altitude bytes** - Altitude correction values
- **Service info** - sets the drone state (0 - Nothing to do, 1 - Stabilization, 2 - Landing (command not implemented and will be removed in the future. This is not a real landing, just to tell the drone to start decreasing altitude), 3 - Disable motors)
- **Check byte** - XOR sum of all previous bytes that is compared via transmittion in order to verify the data
- **Data suffix** - unique pair of ASCII symbols that is not represented in the packet in any form and that shows the end of the packet

On the drone side (Liberty-Link module), data reading is performing as follows:

```
while (Telemetry_serial.available()) {
    tdc_receive_buffer[tdc_receive_buffer_counter] = Telemetry_serial.read();
    if (tdc_receive_byte_previous == 'L' && tdc_receive_buffer[tdc_receive_buf
        tdc_receive_buffer_counter = 0;
        if (tdc_receive_start_detect >= 2) {
            tdc_check_byte = 0;
            for (tdc_temp_byte = 0; tdc_temp_byte <= 8; tdc_temp_byte++)
                tdc_check_byte ^= tdc_receive_buffer[tdc_temp_byte];
            if (tdc_check_byte == tdc_receive_buffer[9]) {
                direct_roll_control = (uint32_t)tdc_receive_buffer[1] | (uint32
                direct_pitch_control = (uint32_t)tdc_receive_buffer[3] | (uint3
                direct_yaw_control = (uint32_t)tdc_receive_buffer[5] | (uint32_
                direct_throttle_control = (uint32_t)tdc_receive_buffer[7] | (ui
                direct_service_info = (uint32_t)tdc_receive_buffer[8];

                if (direct_roll_control > 1100 && direct_roll_control < 1900 &&
                    direct_pitch_control > 1100 && direct_pitch_control < 1900
                    direct_yaw_control > 1100 && direct_yaw_control < 1900 &&
                    direct_throttle_control > 1100 && direct_throttle_control <
                    /*flight_mode == 2 &&*/ channel_7 > 1500) {
                    tdc_timer = millis();
                    tdc_working = 1;
                }
                else
                    tdc_working = 0;
            }
            else {
                direct_roll_control = 1500;
                direct_pitch_control = 1500;
                tdc_working = 0;
            }
        } else
            tdc_receive_start_detect++;
    }
    else {
        tdc_receive_byte_previous = tdc_receive_buffer[tdc_receive_buffer_count
        tdc_receive_buffer_counter++;
        if (tdc_receive_buffer_counter > 11)tdc_receive_buffer_counter = 0;
    }
}
if (millis() - tdc_timer >= 500) {
    tdc_working = 0;
}
if (tdc_working && direct_service_info == 2 && !return_to_home_step)
    return_to_home_step = 3;
if (!tdc_working)
    return_to_home_step = 0;
if (!tdc_working || direct_service_info < 1) {
    direct_roll_control = 1500;
    direct_pitch_control = 1500;
    direct_yaw_control = 1500;
    direct_throttle_control = 1500;
}
```

As a result, there are 4 variables:

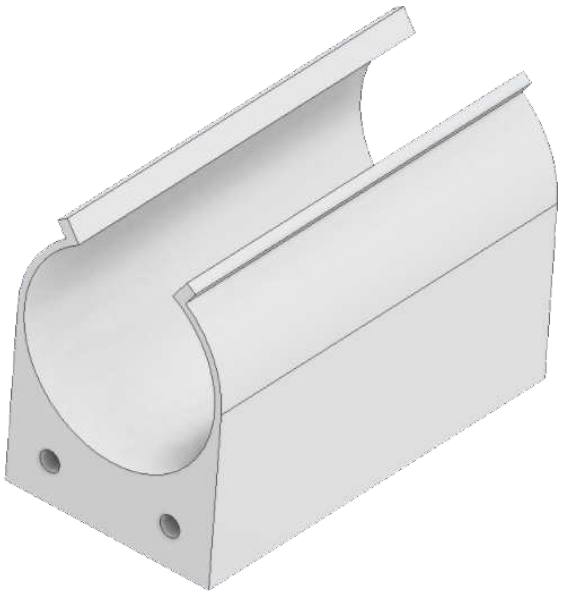
```
direct_roll_control
direct_pitch_control
direct_yaw_control
direct_throttle_control
```

Which are directly added to the data received from the control panel. Probably, in the future, other data will be added, at least for working with GPS. Stay tuned for updates in our repository.

5.7. Camera gimbal

To operate our system in real conditions, it is required to minimize camera shaking if we don't want to lose the tag on the drone. For that reason, a 3D model of a gimbal for attaching the drone to the platform was developed to stabilize a conventional webcam.

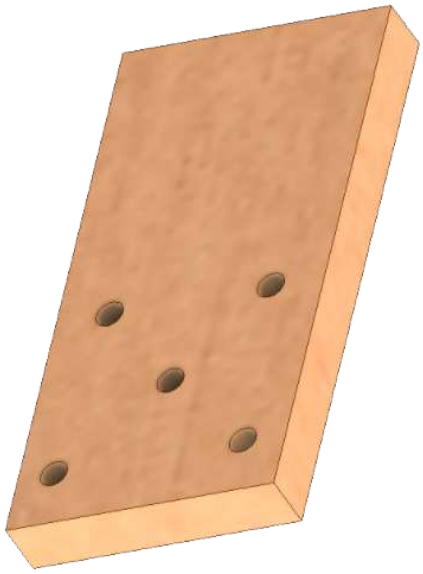
Camera mount:



Camera wire fixing (ferrite filter on the wire):



Fixing of the "crabs" latches on the suspension substrate:



An approximate view of the assembly of the entire suspension mechanism:



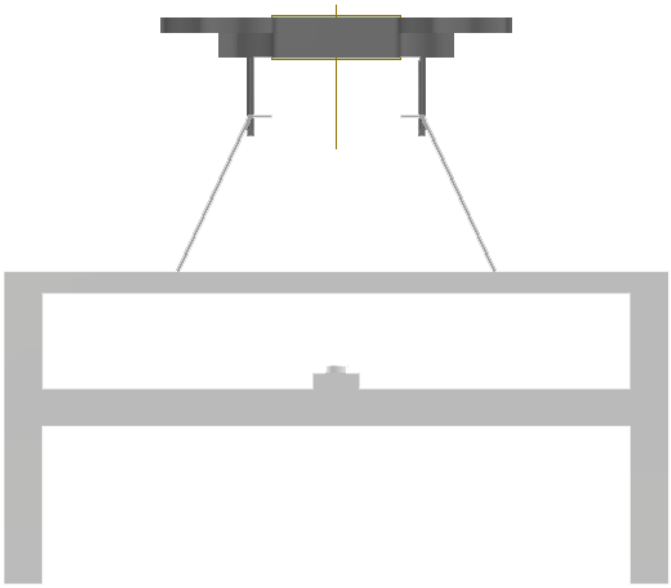
6. Eitude AMLS Platform

The platform is an interconnected system for landing the drone. The platform was planned to be controlled via the Serial interface, using the G-Code commands: The current platform code can be found in the Eitude GitHub repository: https://github.com/XxOinvizioNxX/Eitude

6.1. Grabbing system

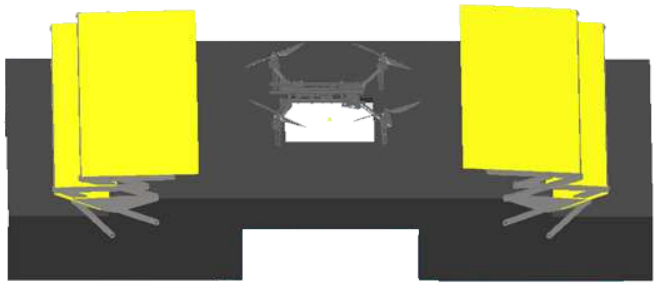
As you may know it doesn't matter how good is our stabilization, without grabbing system the drone will crash eventually. Hence we developed a 3D model of a grabbing system with 4 grips that have a hook at the end of each one. This will allow the system to slowly grab the drone while it lands and hold it steady after landing.

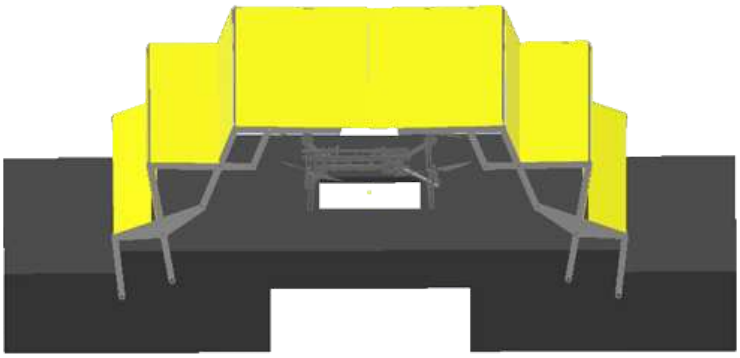




6.2. Weather protection system

As for the weather protection, we developed a 3D model to create a roof that will protect the drone from weather conditions while it is on the platform. The AMLS weather protection system consists of scissor-like mechanisms covered with a canvas, which are located around the edges of the platform. After a successful landing, the mechanisms on both sides of the platform close and protect the drone from external influences. The roof structure itself makes it light and strong, and the scissor-like mechanisms allow it to simply fold and unfold itself. Moreover, the assembly of such mechanisms will be simple and reliable.



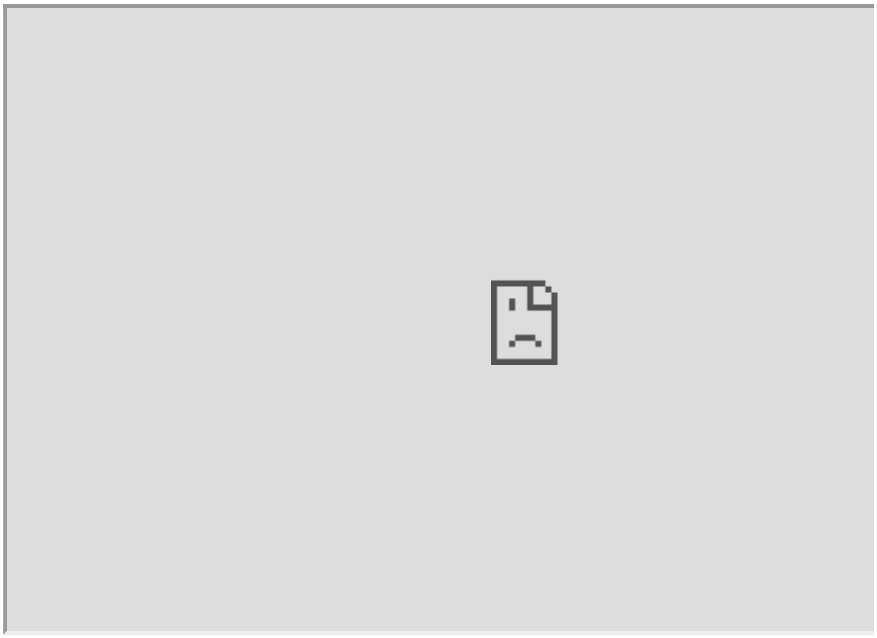


6.3. Platform speedometer

In order to land on a quickly moving platform, it is very useful to know its speed. For now, the platform does not have a GPS module, or any other way to measure absolute speed. Therefore, for a temporary solution to this problem, it was decided to calculate the speed using an accelerometer. For example, MPU6050. The IMU is mounted into the prototype platform through a soft backing and it's covered with a cap to protect it from the wind. The stabilization algorithm (Liberty-Way) sends a request to the platform `L1` to test the speed. A message `S0 L <speed in km / h>` is returned as a response.



Speedometer test (inside the gray circle, lower right parameter (SPD) - speed in km / h) (Clickable):



To calculate the speed, the acceleration is taken for short time periods, then multiplied with time, which results with the instantaneous speed. Then this speed is constantly added to the previous value:

```
void speed_handler(void) {
    gyro_signalen();

    // Filter accelerations
    acc_x_filtered = (float)acc_x_filtered * ACC_FILTER_KOEFF + (float)acc_x *

    speed = acc_x_filtered;
    // Convert acceleration to G
    speed /= 4096.0;
    // Convert to m/s^2
    speed *= 9.81;
    // Multiply by dt to get instant speed in m/ms
    speed *= (millis() - speed_loop_timer);

    // Reset timer
    speed_loop_timer = millis();

    // Convert to m/s
    speed /= 1000.0;
    // Convert to km/h
    speed *= 3.6;

    // Accumulate instantaneous speed
    speed_accumulator += speed;

    if (!in_move_flag) {
        // If the platform is not moving, reset the speed
        speed_accumulator = speed_accumulator * SPEED_ZEROING_FACTOR;
    }
}
```

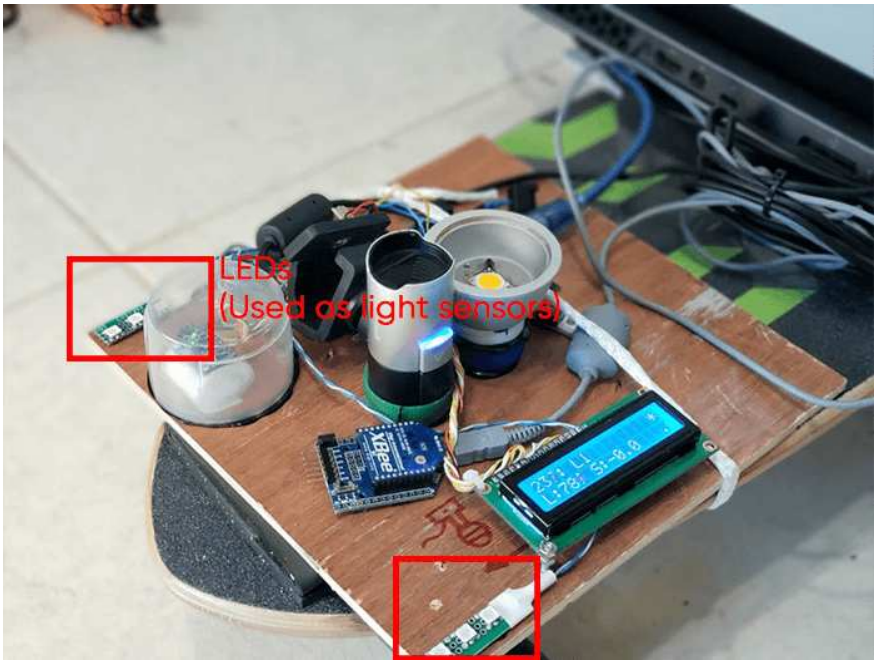
Despite having various filters, due to the error, the speed may not "return" to 0, therefore, vibrations are also measured, and if they are less than the certain threshold, it is considered that the platform is at a standstill and the speed gradually resets to zero.

The complete code of the speedometer can be found in the Eitude repository on GitHub: https://github.com/XxOinvizioNxX/Eitude

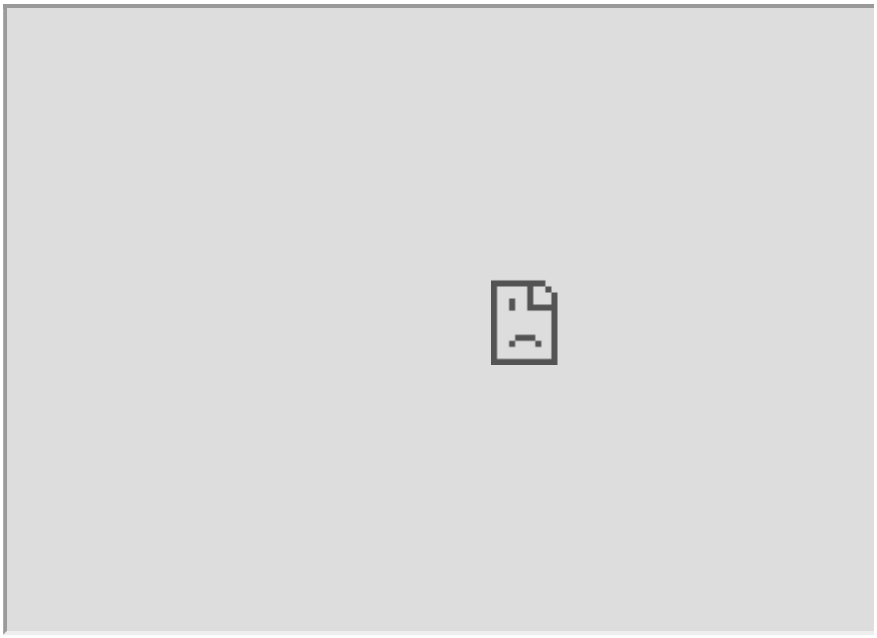
6.4. Platform light sensor

As our platform must work in various environmental conditions, and optical stabilization is very demanding on the visibility of the ArUco marker, it is important to have an automatic system for measuring the camera exposure by the level of illumination around it, and turning on additional illumination if there is a lack of lighting. In the long term, it is planned to use specialized sensors, for example, the BH1750, as light sensors.

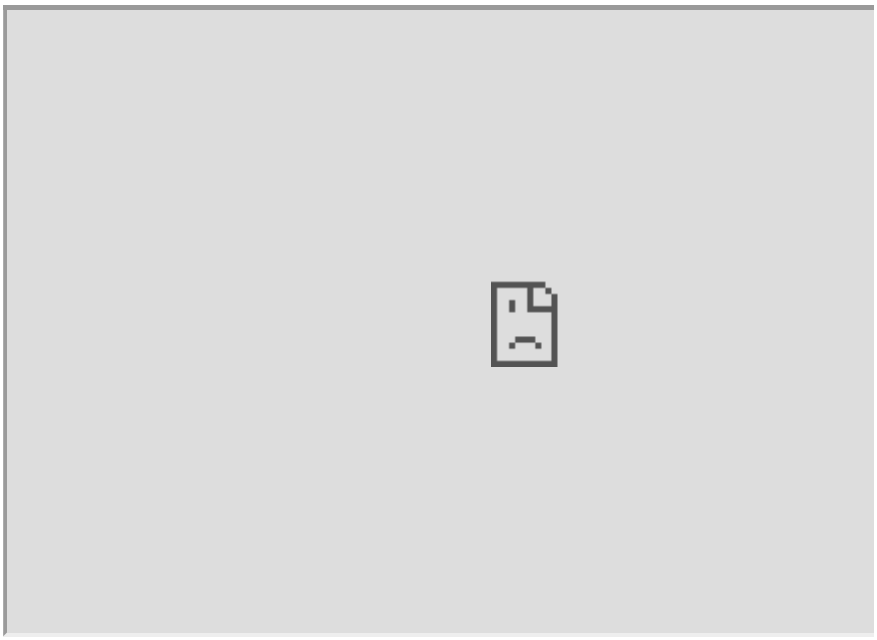
In the current version of the prototype, 6 LEDs are used as a light sensor and an ADC built into the microcontroller. The stabilization algorithm (Liberty-Way) sends a request `L0` to the platform to check the illumination level. A message `S0 L <luminance>` is returned as a response.



Test for determining the level of illumination using LEDs (clickable):



Exposure adjustment and adding additional illumination tests (clickable):



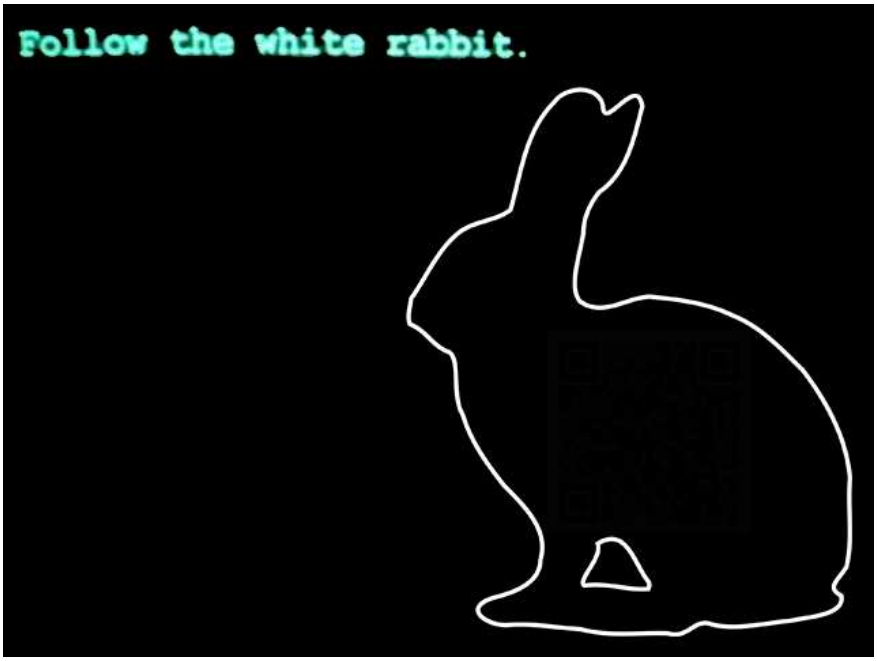
7. Conclusion

At the moment, there is a debugged prototype of optical stabilization, GPS holding, altitude stabilization via barometer, different platform prototypes and a great amount of 3D models eager to be constructed. The project of the automatical landing of a drone onto a moving platform is not yet complete.

Follow the updates:

- In our repository GitHub: https://github.com/XxOinvizioNxX/Liberty-Way
- On our YouTube channel: https://www.youtube.com/channel/UCqN12Jzy-1eJLkcA32R0jdg

In the future, we plan to do much more new and interesting stuff!



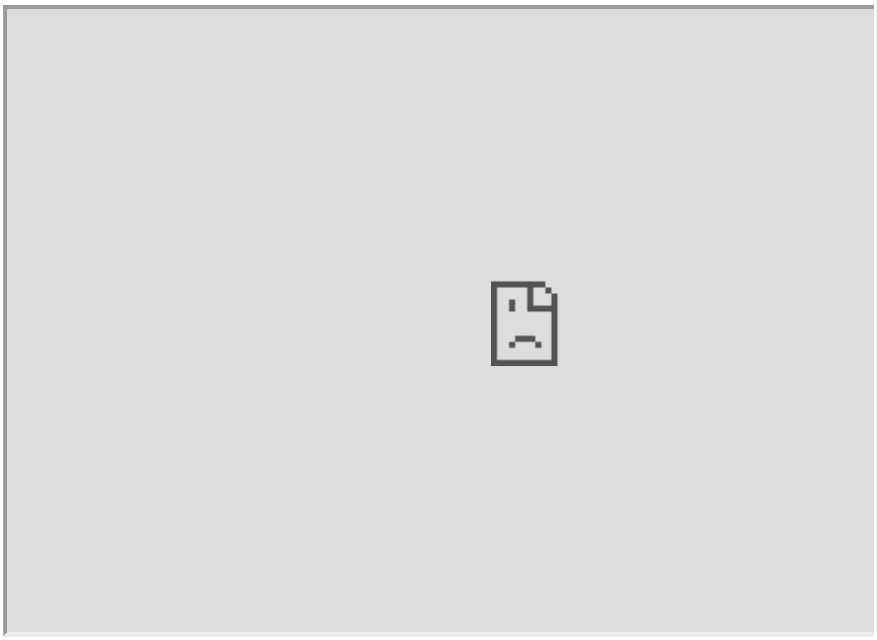
clever-show

Software for making the drone show controlled by Raspberry Pi, PX4 and COEX Clover package.

Create animation in Blender, convert it to drone paths, set up the drones and run your own show!

Project repository: https://github.com/CopterExpress/clever-show.

Demo video



12 drones perform in a show in Electrotheatre Stanislavsky, Moscow.

Team

- Arthur Golubtsov, software engineer in COEX, https://github.com/goldarte
- Artem Vasyunik, software engineer intern in COEX, https://github.com/artem30801

Innopolis Open 2020 - L22_ÆRO team

Team

- Yuryev Vasily.
- Okoneshnikov Dmitriy.

Final task description

New technologies are being implemented into various sectors of the economy including agricultural industry. Drones or UAV are no exception. Thanks to their usage, assessment of agricultural territories' states and analysis of landscape components became more effective and accessible.



The final task of Innopolis Open 2020 was dedicated to monitoring agricultural territories and consisted of the following elements:

- Takeoff (from QR-code) and landing (onto a small colored marker).
- QR-code recognition.
- Colored objects recognition (Colored markers were used to show "agricultural territories").
- Determination of their coordinates (their locations change).
- Making a report using the gathered data.

Code

The code is available on GitHub:
https://github.com/vas0x59/ior2020_uav_L22_AERO.

Main program

When implementing the code, in the original concept we used our own message types, multiple nodes, and other ROS stuff. For this you need to create a package and compile it, but due to the specifics of the competition (a single SD card was used for all teams), all the code was merged into a single file. This approach made debugging much more difficult, but running the code on the drone became much easier.

Parts of the program:

1. Takeoff.
2. QR-code recognition.
3. Color markers recognition.
4. Landing.
5. Generating a report and a video.

The final coordinates of markers are automatically grouped and averaged data from the recognition system received for the entire flight. The "Zig-zag" trajectory was chosen to cover the entire territory. The Gazebo simulator is used for debugging.

Color markers

`l22_aero_vision/src/color_r_c.py`

For image processing and object detection, we used functions from the OpenCV library.

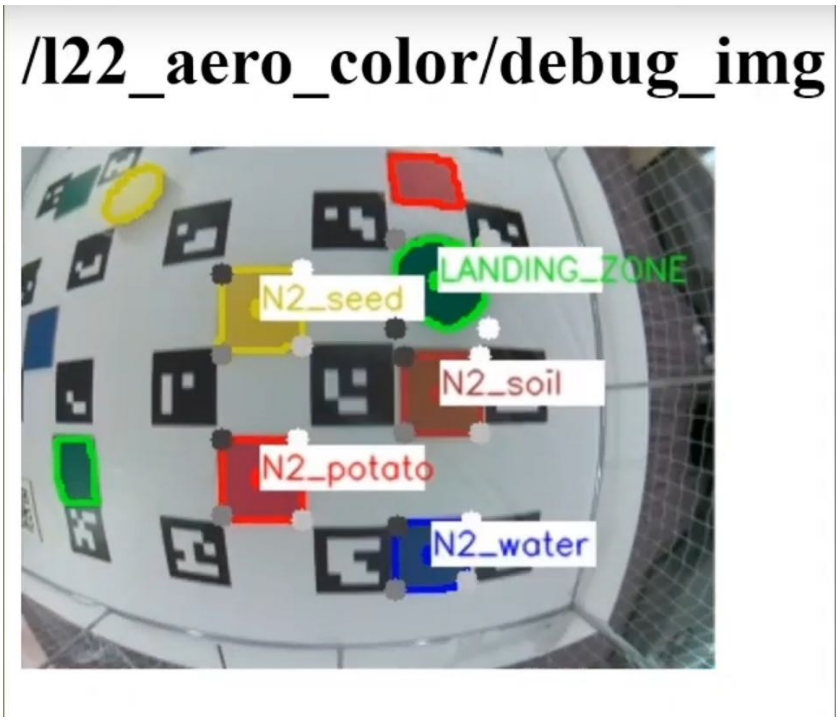
Algorithm:

1. Receiving the image and camera parameters.
2. Building a mask based on a specific color range (in HSV format).
3. Detection of contours of colored objects.
4. Determining the object type, getting the key points of the object in the image.
5. Determining the position of squares and circles using solvePnP based on the actual size of objects and key points in the image (OpenCV Docs).
6. Sending results to the topics `/l22_aero_color/markers` and `/l22_aero_color/circles` (coordinates relative to `main_camera_optical` frame).

During the development, our own message types and a service for changing the detector parameters during the landing were created. (`ColorMarker` , `ColorMarkerArray` , `SetParameters`).

To convert the position of colored objects into `aruco_map` frame TF library was used. (http://wiki.ros.org/tf)

Due to distortions at the edges of the image from the fisheye lens, all the recognized contours located near the edge of the image are ignored. This filter is disabled during landing. The object type is determined using the contour analysis functions (`approxPolyDP` - number of vertexes; `minAreaRect` , `contourArea` - area of bounding rect / area of the contour; `minAreaRect` - aspect ratio).



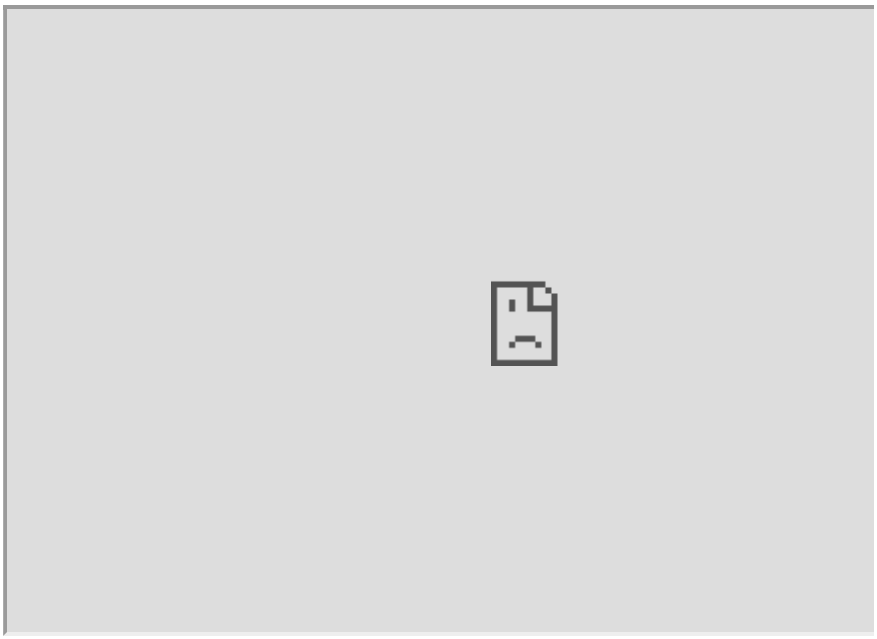
Examples of marker recognition:



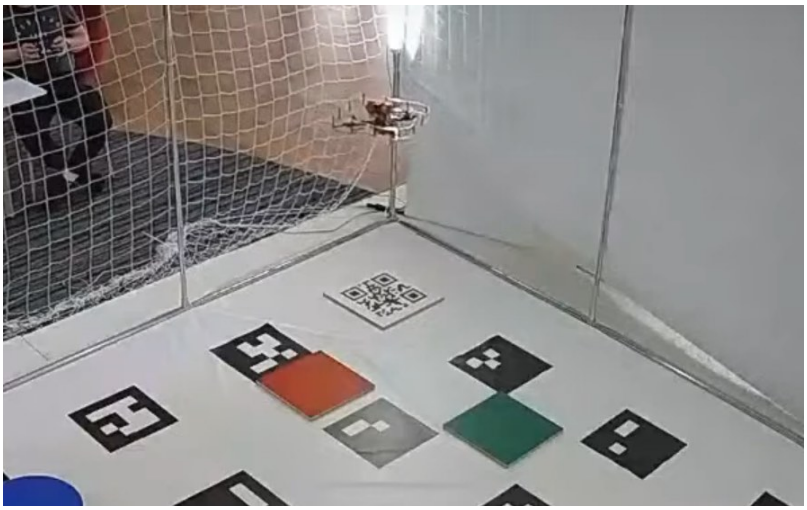
Visualization in RViz

`l22_aero_vision/src/viz.py`

To debug object recognition, a script has been created that visualizes the coordinates of markers in the RViz utility.



QR-code

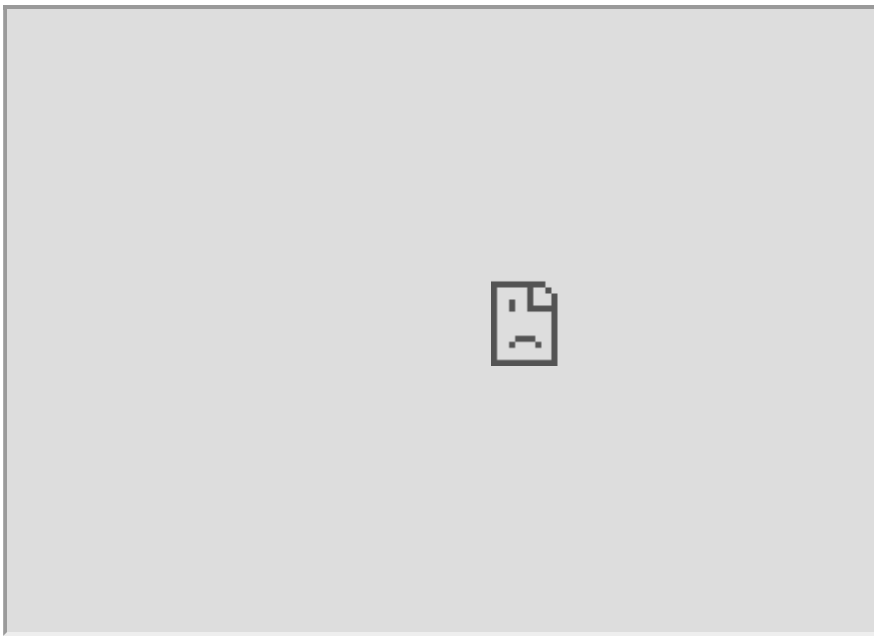


The PyZbar library was used to perform the QR-code recognition. In order to improve the accuracy of QR-code recognition, flights around the QR-code were performed at low altitude.

Landing

Landing is performed in 3 stages:

1. Flight to the intended landing zone and hovering at an altitude of 1.5 m.
2. Descending to a height of 0.85 m with 3 adjustments to the marker coordinates relative to `aruco_map` frame.
3. Descending for several seconds with adjustments based on the coordinates of the landing marker in `body` coordinate system (since ArUco-markers may no longer be visible), instead of `navigate`, `set_position` is used.



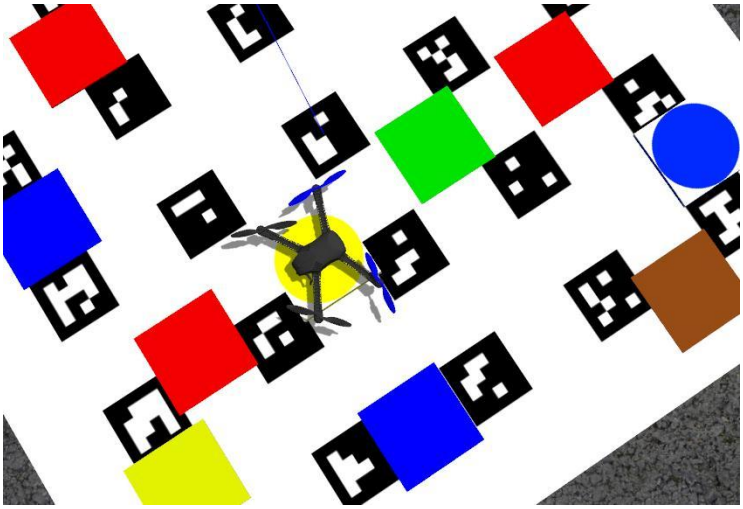
Gazebo

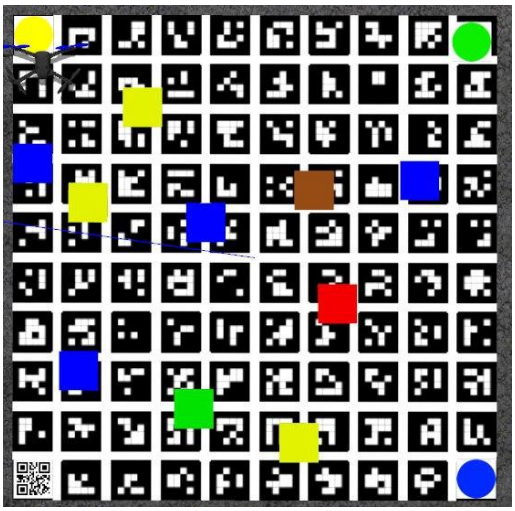
Due to limitations of opportunities to test the code on a real drone, we decided to use the Gazebo simulator.

To run the Clover software package in the simulator, you can use this set of scripts or original instruction from PX4.

For Innopolis Open we have prepared several test scenes. ior2020_uav_L22_AERO_sim.

Also, the use of the simulator accelerated the debugging of the full code execution, since the launch was performed with real time factor = 2.5.





During testing, some problems were found (e.g. incorrect position of `aruco_map`) while using distortion in the camera plugin, so the simulator used a Pinhole camera (without any distortions from the lens).

ROS

Created nodes, topics, messages and services.

Nodes

- `l22_aero_vision/color_r_c.py` - recognition of colored objects.
- `l22_aero_vision/viz.py` - visualization in RViz
- `l22_aero_code/full_task.py` - main code.

Topics

- `/l22_aero_color/markers` (`l22_aero_vision/ColorMarkerArray`) - list of rectangular markers.
- `/l22_aero_color/circles` (`l22_aero_vision/ColorMarkerArray`) - list of round markers.
- `/l22_aero_color/debug_img` (`sensor_msgs/Image`) - image for debugging.

- `/qr_debug` (`sensor_msgs/Image`) - image for debugging.

Messages

`ColorMarker`

```
string color
int16 cx_img
int16 cy_img
float32 cx_cam
float32 cy_cam
float32 cz_cam
float32 size1
float32 size2
int16 type
```

`ColorMarkerArray`

```
std_msgs/Header header
l22_aero_vision/ColorMarker[] markers
```

Services

`SetParameters`

```
float32 rect_s1
float32 rect_s2
float32 circle_r
int32 obj_s_th
int32 offset_w
int32 offset_h
---
```

Copter spheric guard

Introduction

Probably, everyone who has held a copter has had to fly it indoors. Such flights are associated with considerable risk of damaging the copter upon hitting walls and various items. Flying even in a relatively large space is associated with the risk of hitting an obstacle: there might be a tree or a building in the path of the copter, to say nothing about flying in confined spaces. Such "crash tests" are not very pleasant, and in the best case may cause losing a considerable sum on repairs, and in the worst case — in losing the copter. Such situations are even more unpleasant for a beginner who cannot avoid an obstacle and is just learning to fly.

All this made us search for a solution. Unfortunately, searching the Internet did not provide a sufficiently simple and, more importantly, cheap solution for ordinary users. For example, propeller guard protects the propellers themselves quite well, but upon the slightest touch upon an obstacle the copter would flip over and fall down. In general, guard either did not protect the copter fully or looked awkward and was too scarcely available.



We made a difficult decision: we had to make it entirely ourselves and almost from scratch, and the goal was to make it lightweight and easy to manufacture.

Development

As a result of searching for a solution that would fit all our requirements, we ended with several similar options. It was decided to make the guard in the shape of a semiregular polyhedron (examples include fullerene, carbon, or a pentakis dodecahedron) — it was chosen as the most pleasing to the eye. In addition, such a guard is easily scalable to the desired size.

In creating such a shape, two types of edges (beams) are used: short and long ones, their length is calculated based on the desired diameter of a polyhedron inscribed into a sphere. For better understanding, I will insert all necessary formulas from Wikipedia below.

If the "short" edges of the pentakis dodecahedron have the length of a, its long edges have the length of $\frac{1}{6}\left(9-\sqrt{5}\right)a \approx 1,13a$

The radius of the inscribed sphere (touching all faces of the polyhedron in their incenters) will be equal to

$$r = \frac{1}{2}\sqrt{\frac{1}{109}\left(477+194\sqrt{5}\right)}\,a \approx 1,4453319a,$$

The corner joints (fittings) were not very easy either. They are of two types as well: with five faces on the vertex (five beams protrude from the vertex) and with six faces (six rays protrude from the vertex).

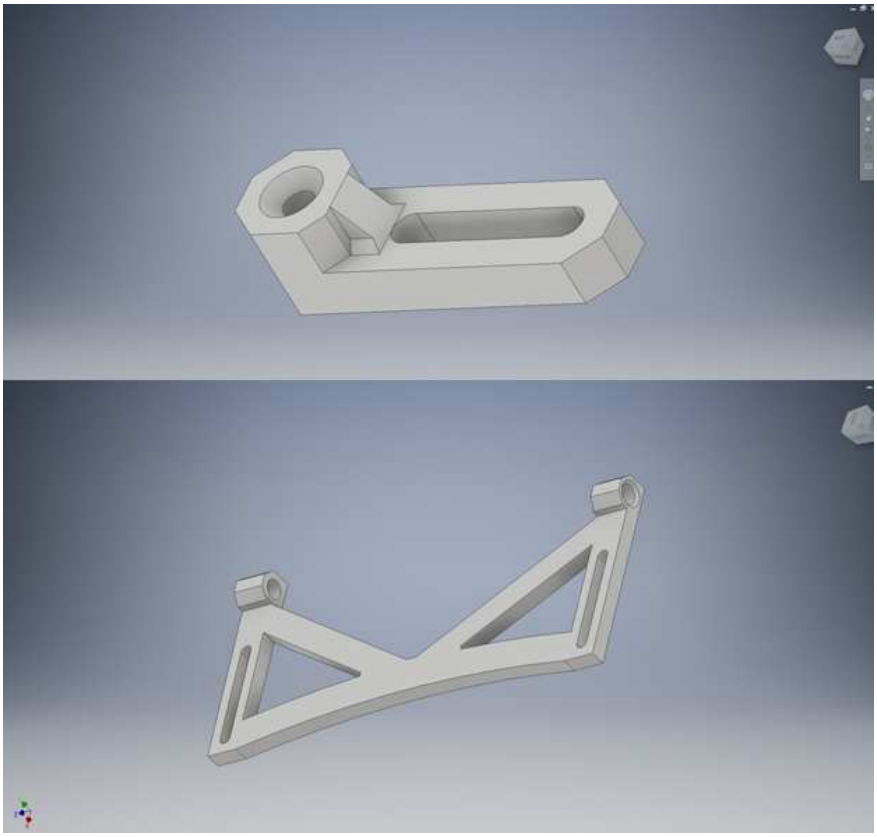
First models

A specification has been prepared for the ease of monitoring the manufacturing process, and we started modeling.

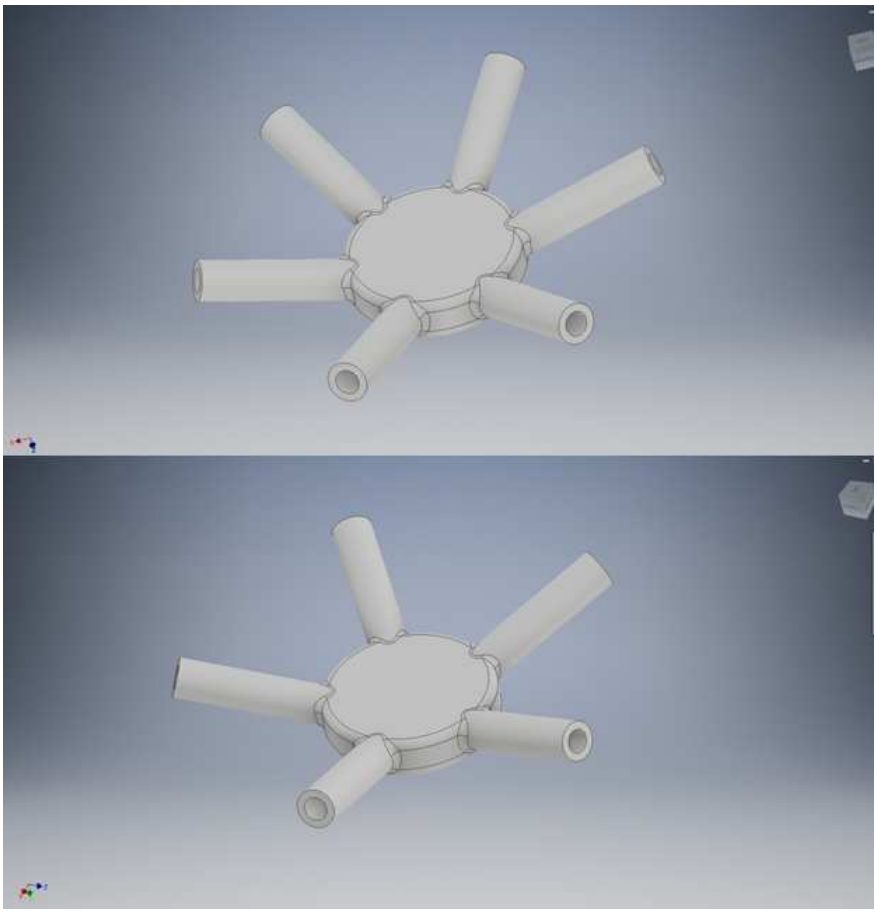
Making simple calculations for the required size, we built a model in Inventor CAD.

In the progress of designing, we faced problems in modeling corner joints, but they were solved by simplifying the design, and the differences of the angles were compensated for by flexible materials. Thus, all joints fit slightly tightly.

Designation	Name	Q-ty	Note
	5-beam 12	12	
	6-beam 20	20	
	Beam 12 5 cm 60	60	
	Beam 14 cm 30	30	
	Main fasteners 2	2	
	Additional fasteners 2	2	
	Copter beam with changes 2	2	
	Standard products		
	M3x16 mm screw 6	6	
	M3 nut 6	6	



(Elements of guard fasteners to the body)

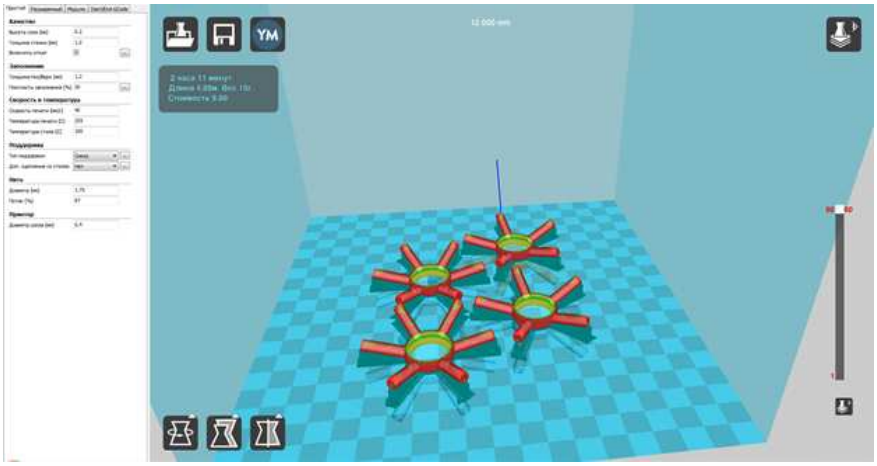


Materials

During the design process, the question arose about a lightweight and strong material to be used for the guard. The answer, as always, was unexpected. We saw bamboo sticks: they were thin enough to not affect the aerodynamics, had sufficient flexibility and were quite strong. Then, the question was about how to make fittings, and of what material. Surely, it should be 3D printing! A 3D printer is a generally indispensable thing, especially for those who like doing something themselves. In addition, due to their moderate price, they are widespread enough. Such a printer may be used to make items of almost any complexity. That is what we need!

Convert ready models to .stl, put them into a slicer (Cura in our case), enter the setting for a particular printer and plastic, and start printing.

To reduce the weight we chose ABC plastic, which is lightweight and available.



The sticks were cut to the calculated length and prepared for subsequent work.

Assembly and installation

After everything has been printed and cut, it is time to assemble the guard.



Assembly is the most crucial moment, as it requires a special algorithm.

From a five-beam fitting, only short beams protrude, while from a six-beam fitting — only every second long one.

Assembly:

1. First, assemble all five-beam vertices.
2. Put a six-beam vertice on every beam protruding from a five-beam vertice.
3. Interconnect the six-beam fittings with long sticks.
4. Connect assembled five-beam vertices to the six-beam vertices, bearing in mind that in a six-beam fitting, short and long beams alternate.
5. Repeat the process for each beam vertice, until the ball is assembled.

After assembly, divide the ball into 2 hemispheres and install the mounts on the copter making sure that everything fits.



(An example of fasteners installation)

Now, the hemispheres may be glued. The hemispheres are not to be glued to each other, as this is required for installing the copter inside. We used the Dichloroethane solvent for plastics as the glue, but you can use any quick-drying adhesive polymer with the same success.

After drying, the guard is ready for installation and for the first test flight!



(without fasteners yet)



First flights

We have made a guard for the Clover 2 copter, which is a learning aid for quadcopters assembly and setup and is installed on it without modification. The guard weighs 70g more (139 grams) than the standard one, and almost does not affect controllability and flying time.

It should also be said that excessive vibrations, if any, may be removed by more rigid attachment to the copter.

Eventually, we've got an unusual guard for the copter that is lightweight and has an interesting design and opens new horizons for flying in the locations where flying copters was dangerous before.

Face recognition system

Introduction

Recently, face recognition systems have been getting a wider use, the application scope of this technology is really expansive: from regular selfie drones to police drones. Everywhere it is being integrated into various devices. The recognition process itself is really fascinating, and that's what inspired me to create a project associated with it. The purpose of my internship project was to create a simple open source system for face recognition with a Clover quadcopter. The program takes images from the quadcopter's camera and processes it on a PC. Therefore, all other instructions are executed on a PC.

Development

The first task was finding a recognition algorithm. As a solution to the problem, a ready API for Python was chosen. This API combines several advantages: recognition speed and accuracy, and ease of use.

Installation

First, you have to install all the necessary libraries:

```
pip install face_recognition
pip install opencv-python
```

Then download the script from the repository:

```
git clone https://github.com/mmkuznecov/face_recognition_from_clever.git
```

Code explanation

Enable libraries:

```
import face_recognition
import cv2
import os
import urllib.request
import numpy as np
```

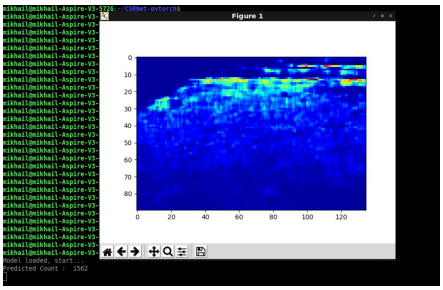
This part of the code is intended for Python 3. In Python 2.7, enable urllib2 instead of urllib:

```
import urllib2
```

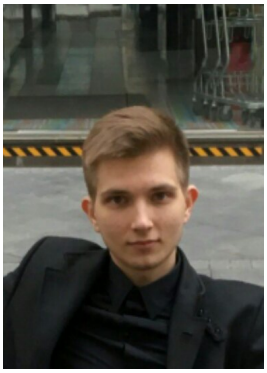
Create a list of encodings for images and a list of names:

```
faces_images=[]
for i in os.listdir('faces/'):
    faces_images.append(face_recognition.load_image_file('faces/'+i))
known_face_encodings=[]
for i in faces_images:
    known_face_encodings.append(face_recognition.face_encodings(i)[0])
known_face_names=[]url
for i in os.listdir('faces/'):
    i=i.split('.')[0]
    known_face_names.append(i)
```

Addition: all images are stored in folder faces in format name.jpg







Initialize some variables:

```
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True
```

Get the image from the server, and convert it to format cv2:

```
req = urllib.request.urlopen('http://192.168.11.1:8080/snapshot?topic=/main_car
arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
frame = cv2.imdecode(arr, -1)
```

For Python 2.7:

```
req = urllib2.urlopen('http://192.168.11.1:8080/snapshot?topic=/main_camera/ima
arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
frame = cv2.imdecode(arr, -1)
```

Further explanation of the code is available at GitHub of the used API in the comments to the next script

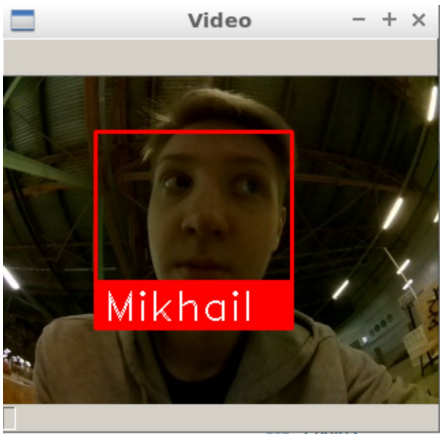
Using

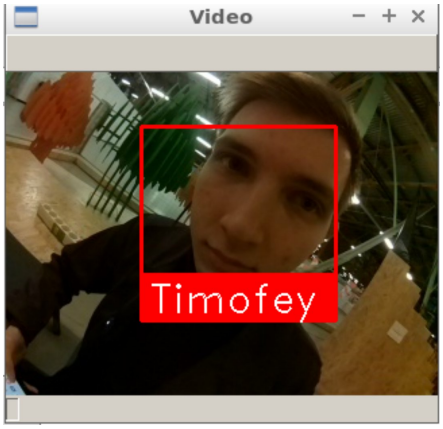
It is enough to connect to "Clover" via Wi-Fi and check whether the video stream from the camera is working correctly.

Then just run the script:

```
python recog.py
```

And the output:





Possible difficulties

When the script is started, the following error may pop up:

```
    known_face_encodings.append(face_recognition.face_encodings(i)[0])
IndexError: list index out of range
```

In this case, try to edit the images in folder faces, perhaps the program cannot recognize faces in the images due to poor quality.

Using the calibration

To improve recognition accuracy, you can use camera calibration. The calibration module may be installed using a special package. Instructions for installation and use are available in the camera calibration article. The program that uses the calibration package is named recog_undist.py

Code brief explanation:

Enable installed package:

```
import clever_cam_calibration.clevercamcalib as ccc
```

Add the following lines:

```
height_or, width_or, depth_or = frame.shape
```

This way, you will obtain information about image size, where height_or is the height of the initial image in pixels, and width_or is the width of the initial image. Then correct distortions in the initial image, and get its parameters:

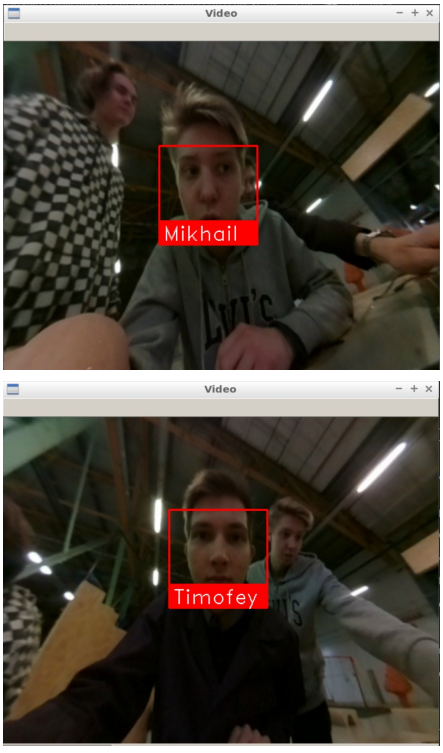
```
if height_or==240 and width_or==320:
    frame=ccc.get_undistorted_image(frame,ccc.CLEVER_FISHEYE_CAM_320)
elif height_or==480 and width_or==640:
    frame=ccc.get_undistorted_image(frame,ccc.CLEVER_FISHEYE_CAM_640)
else:
    frame=ccc.get_undistorted_image(frame,input("Input your path to the .yaml f
height_unz, width_unz, depth_unz = frame.shape
```

In this case, we pass argument ccc.CLEVER_FISHEYE_CAM_640, since the resolution of the image in this example, is 640x480; you can also use ccc.CLEVER_FISHEYE_CAM_320 for resolution 320x240, otherwise you will have to send the path to the .yaml calibration file as the second argument.

Finally, return the image to its initial size:

```
frame=cv2.resize(frame,(0,0), fx=(width_or/width_unz),fy=(height_or/height_unz)
```

This was, you can significantly improve recognition accuracy since the image processed will not be so badly distorted.



An Android transmitter

As early as in the frosty January 2018, all owners of Apple mobile devices got a nice Wi-Fi piloting app for iOS. And now, a year later, such an application is available for another operating system. The latest version may be downloaded **here**.

Introduction

In this article, I will tell you how to write your own or modify an existing transmitter for Android yourself. We will use the popular language *Kotlin*, and we will use *Android Studio* for an IDE. For those who never used it, I recommend reading the following *materials*. The entire application code can be found **here**. If you want to immediately get an app to further tuning, run the following command:

```
git clone https://github.com/Tennessium/android
```

However, to make you fully understand the application, I will tell you about each stage of the project, as if you were building it from scratch.

Wrapper

Let's start with the simplest thing — the appearance of our application. At **GitHub**, you can find *HTML*, *CSS* and *JavaScript* files, which make up the web page to be used for controlling the copter. To have this page displayed in our application, do the following:

1. Create folder **assets** in the main folder of the app named **app**
2. Add to it all files from here

If you reached this stage, you already have the web page you want, congratulations! Now we have to display it somehow in the app. To do this, in class *activity* in method **onCreate**, write the following code:

```
main_web.loadUrl("file:///android_asset/index.html")
```

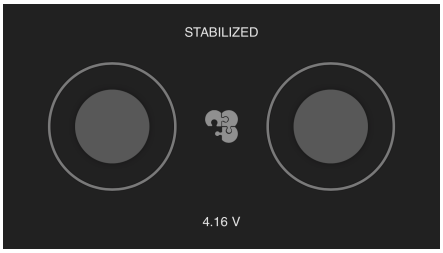
Where *main_web* is the ID of your *WebView*, which is in the *xml* file of the *activity* selected by you.

Unfortunately, the quadcopter transmitter requires the entire screen of the device, while the interface elements of the system interfere with full-fledged use of the program. For this purpose, at the beginning of method **onCreate**, call the following function:

```
private fun fullScreenCall() {
    window.setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.L
    if (Build.VERSION.SDK_INT < 19) {
        val v = this.window.decorView
        v.systemUiVisibility = View.GONE
    } else {
        //for higher API versions.
        val decorView = window.decorView
        val uiOptions = View.SYSTEM_UI_FLAG_HIDE_NAVIGATION or View.SYSTEM_UI_F
        decorView.systemUiVisibility = uiOptions
    }
}
```

This feature allows getting rid of the system interface elements. Let's go ahead.

This is how the transmitter looks at this stage:



If you run your application, you will see that the sticks are not functioning. This is due to the fact that *JavaScript* is disabled in our page. To enable it, write the following code:

```
    main_web.settings.apply {
    domStorageEnabled = true
    javaScriptEnabled = true
    loadWithOverviewMode = true
    useWideViewPort = true
    setSupportZoom(false)
}
```

This piece of code allows the page to use *JavaScript* and at the same time prepares for the next stage - **logics**.

Receiving data from the web page

To let your phone receive data from the *HTML page*, create a class for interacting with the web interface

```
class WebAppInterface(c: Context) {
    @JavascriptInterface
    public fun postMessage(message: String) {
        val data = JSONObject(message)
        send("255.255.255.255", 35602, pack(
        data.getInt("x").toShort(),
        data.getInt("y").toShort(),
        data.getInt("z").toShort(),
        data.getInt("r").toShort()))
    }
}
```

This class will receive messages from the web page sent by the *postMessage* where argument *message* is the message from the page.

Now we have to link classes **WebAppInterface** and **MainActivity**. For this you have to add just one line to method **onCreate**:

```
main_web.addJavascriptInterface(WebAppInterface(this), "appInterface")
```

Sending data to the copter

Important! For working in Internet in the *Android* platform, add the following line to tag *manifest* in file **AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

It will grant your application access to the Internet, and the ability to send data via **Wi-Fi**. And you will now learn how to do that. Let's go ahead.

You have probably noticed function *send* in class **WebAppInterface**. It is this function that sends data to the copter. Let's declare it **outside classes**:

```
fun send(host: String, port: Int, data: ByteArray, senderPort: Int = 0): Boolea
    var ret = false
    var socket: DatagramSocket? = null
    try {
        socket = DatagramSocket(senderPort)
        val address = InetAddress.getByName(host)
        val packet = DatagramPacket(data, data.size, address, port)
        socket.send(packet)
        ret = true
    } catch (e: Exception) {
        e.printStackTrace()
    } finally {
        socket?.close()
    }
    return ret
}
```

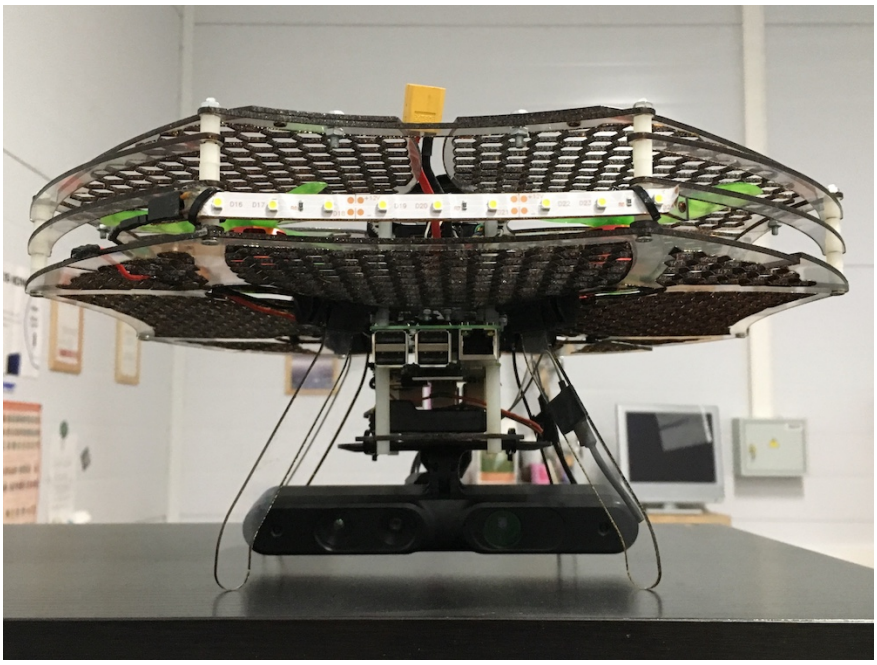
This function sends data via the *user datagram protocol* to the copter. The program sends **bytes**, so it would be a good idea to declare the function for creating an array of **bytes** from four variables:

```
fun pack(x: Short, y: Short, z: Short, r: Short): ByteArray {
    val pump_on_buf: ByteBuffer = ByteBuffer.allocate(8)
    pump_on_buf.putShort(r)
    pump_on_buf.putShort(z)
    pump_on_buf.putShort(y)
    pump_on_buf.putShort(x)
    return pump_on_buf.array().reversedArray()
}
```

Summary

Now your app has the full functionality of its analog for **iOS**. You can customize it as you wish. For any questions about the app, contact us in Telegram @Tenessinum.

3D-scanning drone



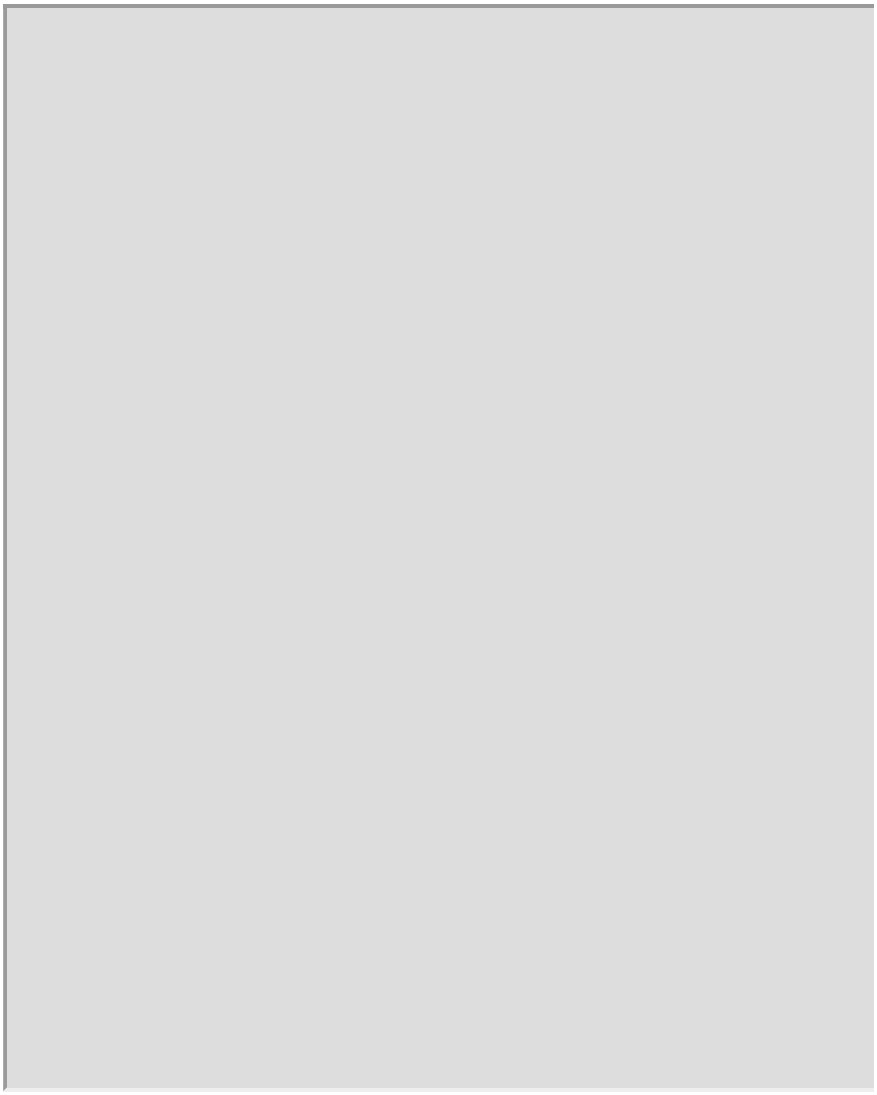
The project was created in collaboration with Texel inc. that develops 3D-scanners for scanning people.

Our fellows from Texel provided a module consisting of a Raspberry Pi and a PrimeSense 3D-sensor.

We provided a Clover 3 drone that's capable of autonomous flight and wrote a flight program.

To make it all work we conducted many tests, made changes in the drone's design and tuned the drone properly.

Video



Team

The project was made by:

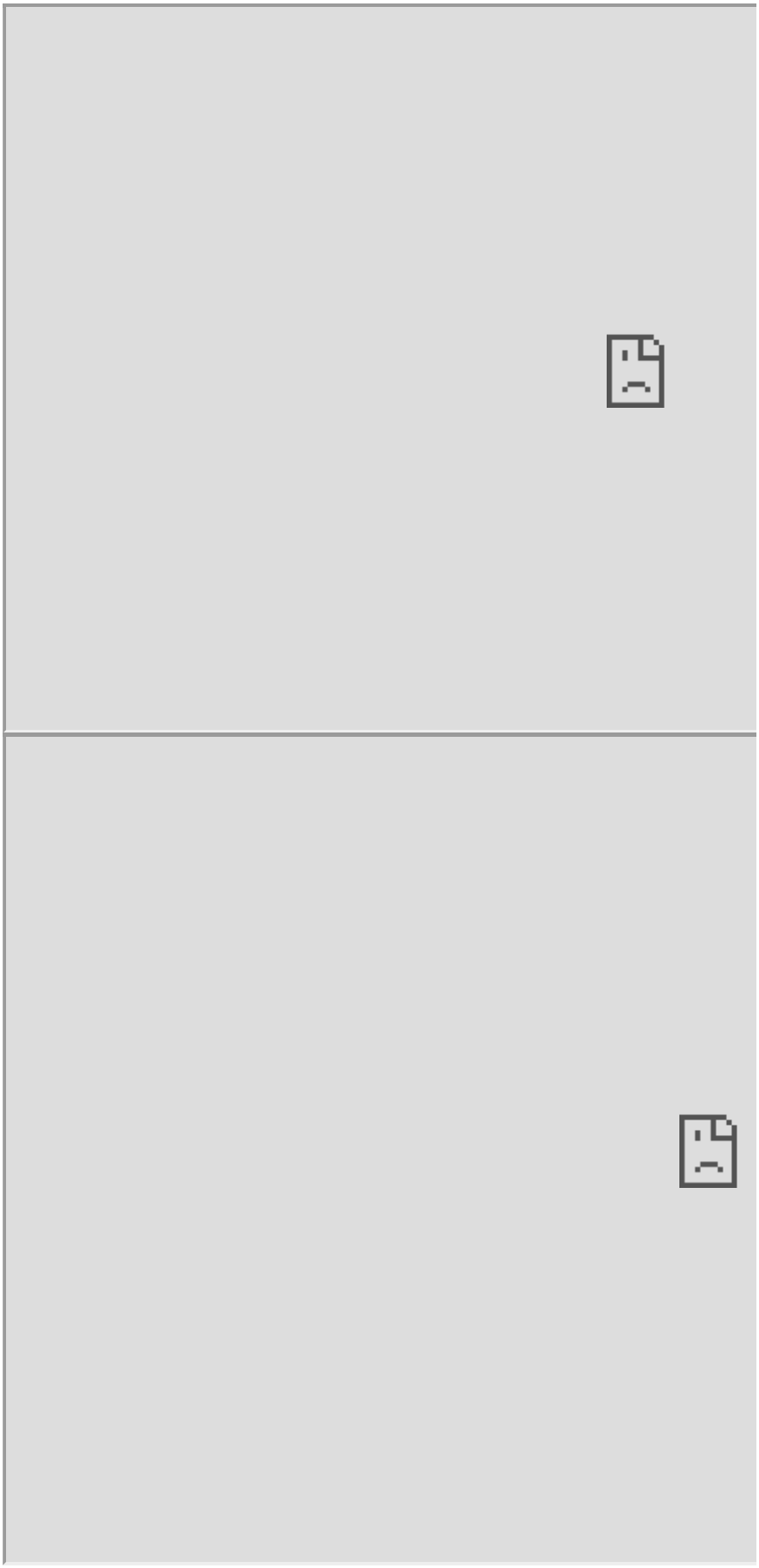
- Timofei Kondratiev [Copter Express] - drone assembly, writing and debugging the program, conducting tests;
- Anton Maltsev [Copter Express] - modeling of the protection of the propellers;
- Andrei Poskonin [Texel] - modifying the Texel's software to work on Raspberry Pi.

Human Pose Estimation drone control

Introduction

Human pose estimation is one of the computer vision applications in order to estimate all the joints and the different poses of the human body through a special camera and a special hardware or process the images from a regular camera by machine learning and deep learning techniques. This project is about controlling the drone through the poses of a person in front of regular camera without needing to an external hardware or hard build dependencies on your computer.

Demo



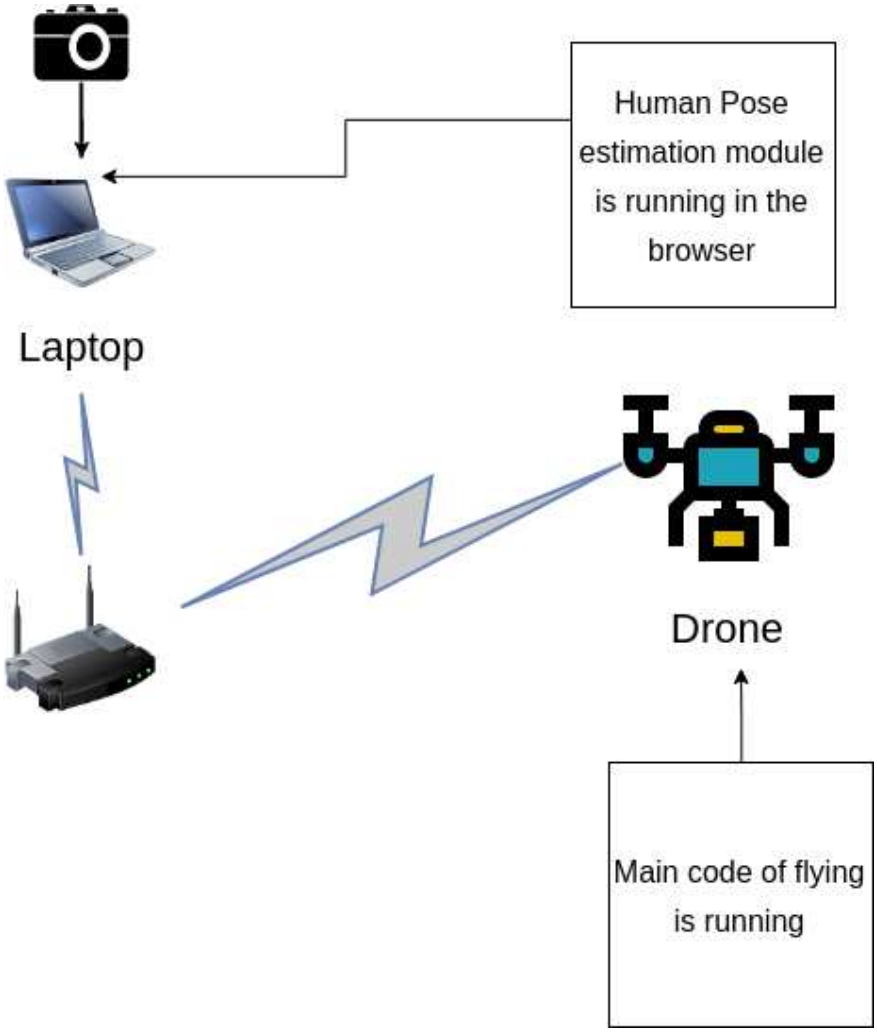
In the demo video, we were using Ubuntu 18.04 and clever4 drone.

Development

We used posenet from tensorflow.js as our human pose estimation module because it is easier to use, build, fast and compatible with different environments(Hardware and OS). You can find the work of posenet for this project here. Websockets were used as communication protocol between the browser and a running server on the drone.

Architecture

The image below is a visualization of our architecture for the project.



This figure is made from here

Dependencies

Before you test it you need to install on your laptop:

- Install Nodejs from here. For Ubuntu installation

- Install Yarn package manager from here. Usual problem while installing and using yarn with Ubuntu.
- Have an experience in manual control on the drone in case of any weird behavior happen.
- Worked before with COEX drones, if this is your first time to work with COEX drones check this.

and you are ready to build and use the required codes.

Setup & installation

In your main laptop

(It has been tested until now only on Ubuntu 18.04)

- Clone the repo of posenet in your computer or download it if you are using Windows without GitHub

```
git clone https://github.com/hany606/tfjs-posenet.git
```

- Do the steps of running and setup as it is described in the README here

In the Raspberry Pi of the drone (Main controller)

- Access the Raspberry Pi
- Switch to Client mode and ensure that the network has internet connection.

Notice: I have already made a bash script based on that tutorial, it is in COEX-Internship19/helpers/ called .to_client.bash To run it:

```
chmod +x .to_client.bash
./.to_client <NAME_OF_NETWORK> <PASSWORD>
```

- Install the tornado library to make a WebSocket server

```
sudo pip install tornado
```

- Clone the main repo on the Raspberry Pi of the drone

```
git clone https://github.com/hany606/COEX-Internship19.git
```

- Go to the project directory

```
cd COEX-Internship19/projects/Human_pose_estimation_drone_control/
```

- Run the server to test that everything is correct and run the posenet, you should see a lot of data is printed in the terminal (if you are running the human pose estimation code on your main computer, just refresh the page in the browser after running the below command in Raspberry Pi)

```
python websocket_server_test.py
```

- Close the server using Ctrl+C
- To run the main file

```
python main_drone.py
```

How to use it

- Run the server first from the Raspberry Pi from the correct directory

```
python main_drone.py
```

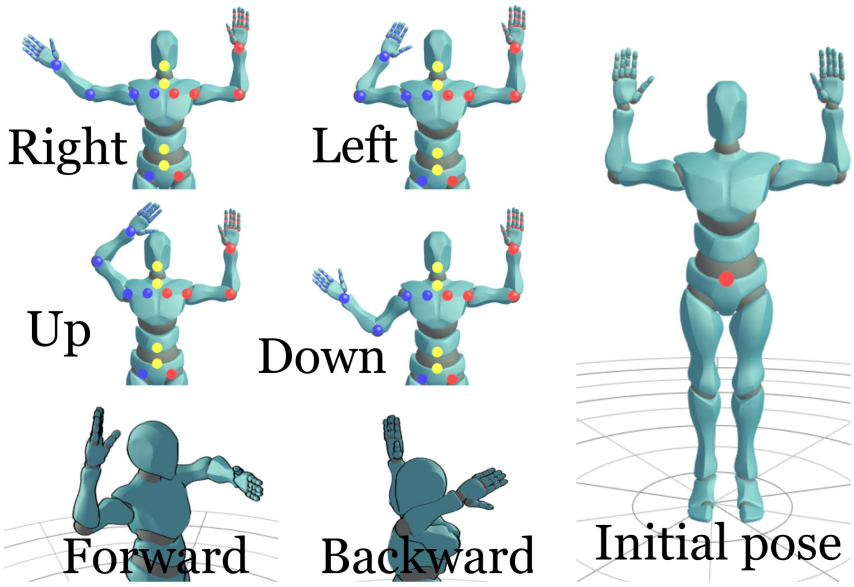
- Run Human pose estimation module on your laptop with WebSocket by

```
yarn websocket
```

Or refresh the page if you already run it.

- You should see the instructions on the screen of the terminal of the Raspberry Pi right now.
- Firstly, you should be visible for the camera and it is better to have a clear background without many details.
- Secondly, you should do initial pose as it is described in the images below.
- You can perform any pose and try to keep it until your drone finish doing this move that is corresponding to the pose.
- After you do the pose return to the initial pose in order to give the drone the command to listen to another pose.
- If you want to stop the program, land the drone and don't return to the initial pose and press Ctrl+C to stop the drone.

Poses



Animation is created by this

Notes

- Websockets are used to communicate between the page on the browser that runs posenet and the drone.
- As the model of posenet is already pre-trained and using tensorflow.js. So, it is quite fast and can run on different computers without any problems thanks to yarn, parcel and tensorflow.js, and we have configured the code of posenet to the minimal configuration to not require a lot of computation power.
- This project has been built in 1 week of working, it took a lot of time trying to make openpose and google colab working but unfortunately I had many errors and one I decided to move to posenet everything was pretty easy.
- If you have any comments about the codes to try to improve it, I will be happy if you can contact me through telegram: @hany606 or email: h.hamed.elanwar@gmail.com or do pull requests.
- If you have implemented any of the applications below, or do some improvements, we will be very happy for that.

Future application

- Drone wars: Control the drone during the drones battle using human poses. It requires high speed interaction and more precise control.
- Control a drone that draw graffiti using human poses and draw in real-time.
- Playing with balls like ping pong game with the drones. It may require 3D Human Pose estimation Algorithms.
- Control two drones by your arms and do some task together.

Acknowledgments

- This project was part of an internship in COEX in July 2019. if you found any bugs or problems, you can contact me through telegram: @hany606 or email: h.hamed.elanwar@gmail.com.
- The above applications were thought by me and my internship supervisor Timofey.

References

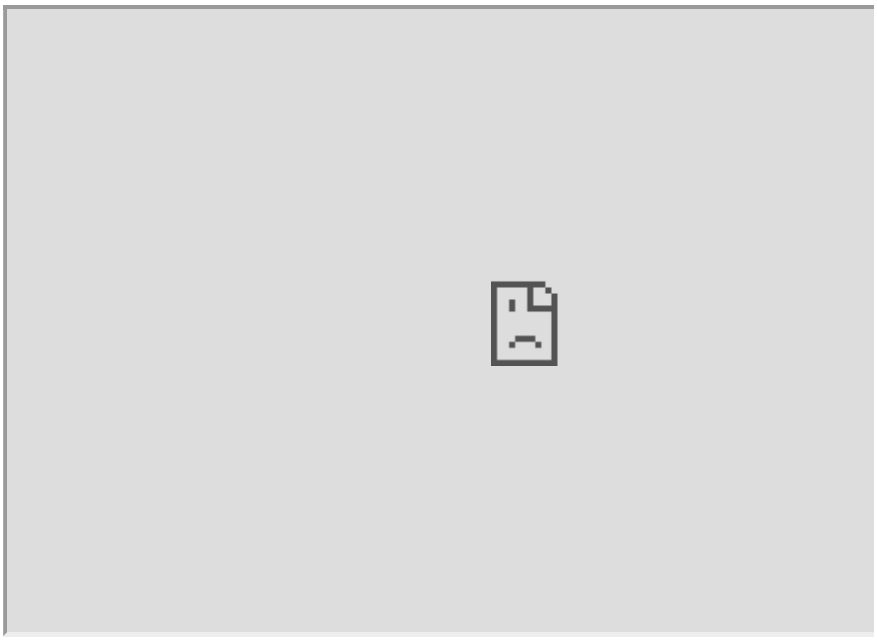
- Human pose estimation guide
- Clover drones tutorials
- Posenet GitHub repo
- Posenet meduim article
- Tensorflow.js demos
- Posenet overview

Robocross-2019

On July, 2019, for the fourth time in a row, the team Copter Express won the annual tests of unmanned vehicles "Robocross". Tests are held at the GAZ test site near Nizhny Novgorod.

The main objective of the tests in the UAV category was to localize and destroy the target - the red balloon - autonomously.

Video



Implementation

The team used an F450 frame based quadcopter and Clover software platform. The final source code is available on GitHub.

`robocross2019` ROS package is divided into two parts: `red_dead_detection` ROS nodelet recognizes the red ball, `ball.py` implements high-level flight logic.

red_dead_detection

The `red_dead_detection` nodelet recognizes the red ball on the image from the forward looking quadcopter camera (`/front_camera/image_raw` and `/front_camera/camera_info` topics). The simplest method of filtering the image by color is applied. Then the nodelet calculates the geometric center of the detected segments, and performs camera distortion compensation (`cv::undistortPoints`).

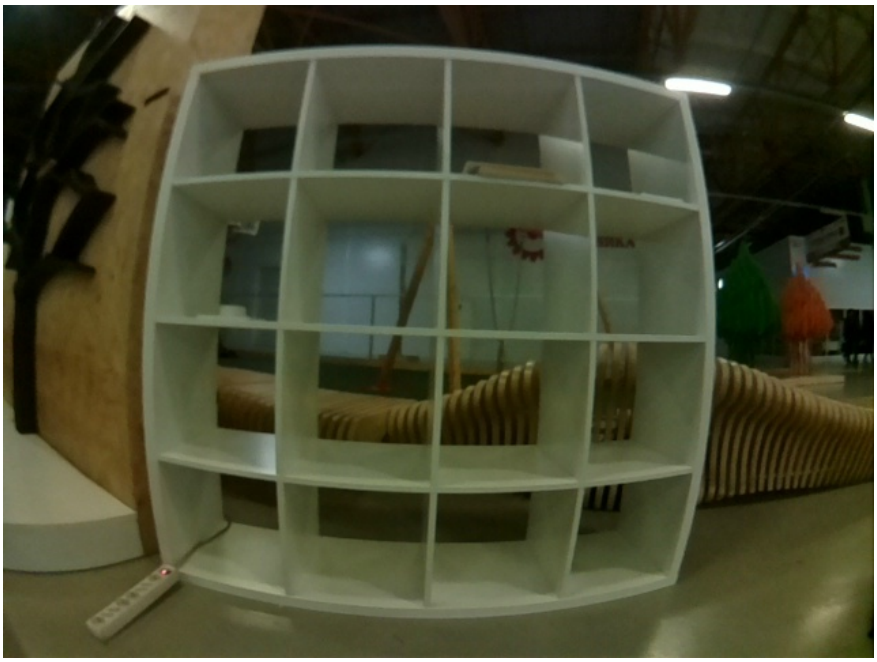
Using the known focal lengths of the camera (from `camera_info`), the nodelet calculates the vector directed towards the target. The resulting vector is published to the topic `/red_dead_detection/direction`; its coordinate system (`frame_id` is associated with the front camera `front_camera_optical`).

balloon.py

To fly towards the ball, the direction vector `red_dead_detection/direction` is used, which is set as a setpoint for the velocity of the drone. The yaw angle is also set towards the ball. The target is considered destroyed when the total area of red pixels is less than the threshold for a certain amount of camera frames.

Camera calibration

Computer vision is becoming more and more widespread. Often, computer vision algorithms are not precise and obtain distorted images from the camera, which is especially true for fisheye cameras.



> The image is "rounded" closer to the edge.

Any computer vision algorithm will perceive the picture incorrectly. To remove such distortion, the camera that receives the image is to be calibrated in accordance with its own peculiarities.

Script installation

First, you have to install the necessary libraries:

```
pip install numpy
pip install opencv-python
pip install glob
pip install pyyaml
pip install urllib.request
```

Then download the script from the repository:

```
git clone https://github.com/tinderad/clever_cam_calibration.git
```

Go to the downloaded folder and install the script:

```
cd clever_cam_calibration
sudo python setup.py build
sudo python setup.py install
```

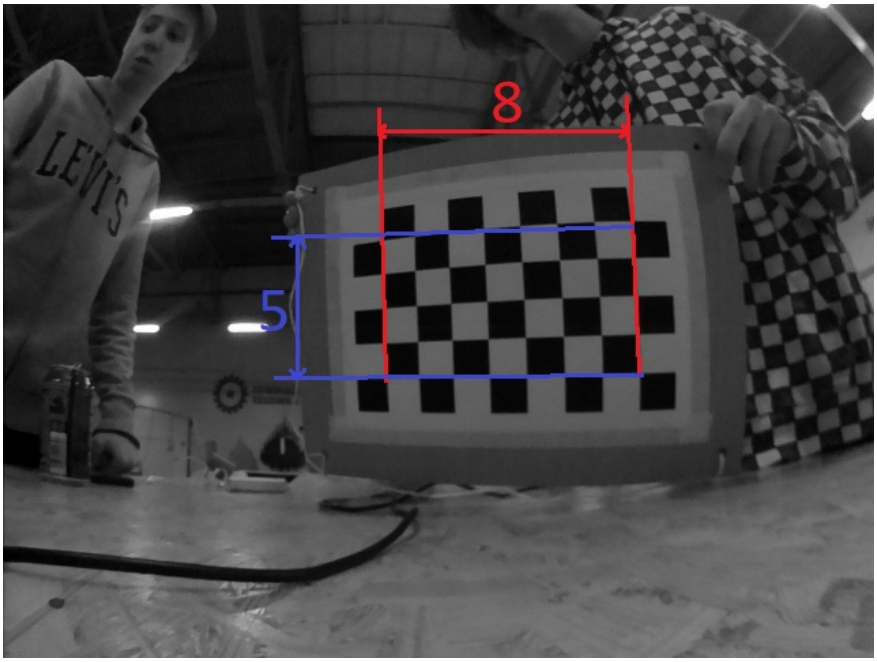
If you are using Windows, download the archive from the repository, unzip it and install:

```
cd path\to\archive\clever_cam_calibration\
python setup.py build
python setup.py install
```

> path\to\archive – path to unpacked archive.

Preparing for calibration

You will have to prepare a calibration target. It looks like a chessboard. The file is available for downloading here. Glue a printed target to any solid surface. Count the number of intersections on the board lengthwise and widthwise, measure the size of a cell (mm).



Turn on Clover and connect to its Wi-Fi.

> Navigate to 192.168.11.1:8080 and check whether the computer receives images from the image_raw topic.

Calibration

Run script ***calibrate_cam***:

Windows:

```
>path\to\python\Scripts\calibrate_cam.exe
```

> path\to\Python – path to the Python folder

Linux:

```
>calibrate_cam
```

Specify board parameters:

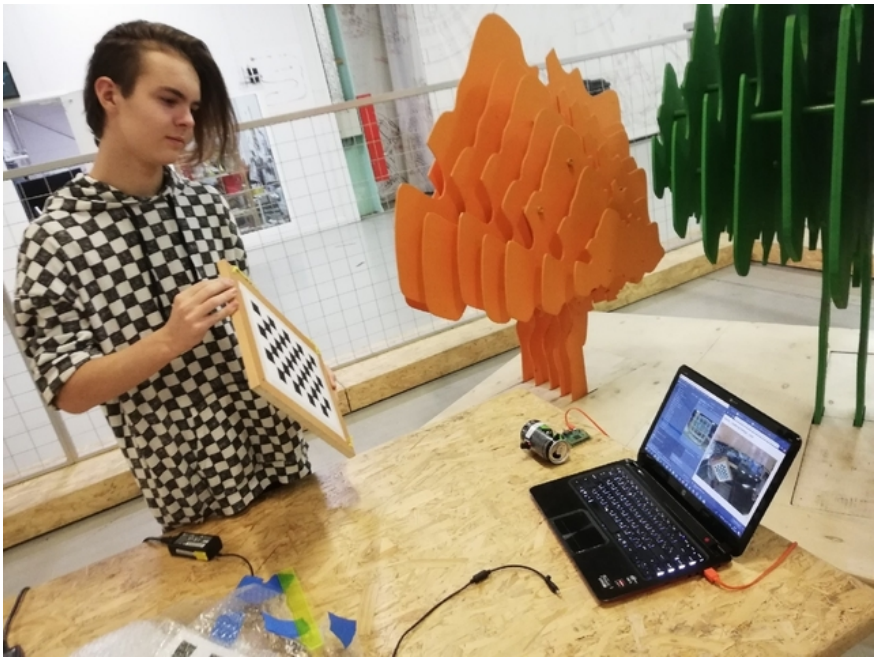
```
>calibrate_cam
Chessboard width: # Intersections widthwise
Chessboard height: # Intersections heightwise
Square size: # Length of cell edge (mm)
Saving mode (YES - on): # Save mode
```

> Save mode: if enabled, all received pictures will be saved in the current folder.

The script will start running:

```
Calibration started!
Commands:
help, catch (key: Enter), delete, restart, stop, finish
```

To calibrate the camera, make at least 25 photos of the chessboard at various angles.



To make a photo, enter command ***catch***.

```
>catch
```

The program will inform you about the calibration status.

```
...
Chessboard not found, now 0 (25 required)
>  # Enter
---
Image added, now 1 (25 required)
```

> Instead of entering command each time, you can just press ***Enter*** (enter a blank line).

After you have made a sufficient number of images, enter command ***finish***.

```
...
>finish
Calibration successful!
```

Calibration by the existing images

If you already have images, you can calibrate the camera by them with the help of script ***calibrate_cam_ex***.

```
>calibrate_cam_ex
```

Specify target characteristics and the path to the folder with images:

```
>calibrate_cam_ex
Chessboard width: # Intersections widthwise
Chessboard height: # Intersections heightwise
Square size: # Length of cell edge (mm)
Path: # Path to the folder with images
```

Apart from that, this script works similarly to ***calibrate_cam***.

The program will process all received pictures, and create file ***camera_info.yaml*** in the current folder. Using this file, you can equalize distortions in the images obtained from this camera.

> If you change the resolution of the received image, you will have to re-calibrate the camera.

Correcting distortions

Function ***get_undistorted_image(cv2_image, camera_info)*** is responsible for obtaining a corrected image:

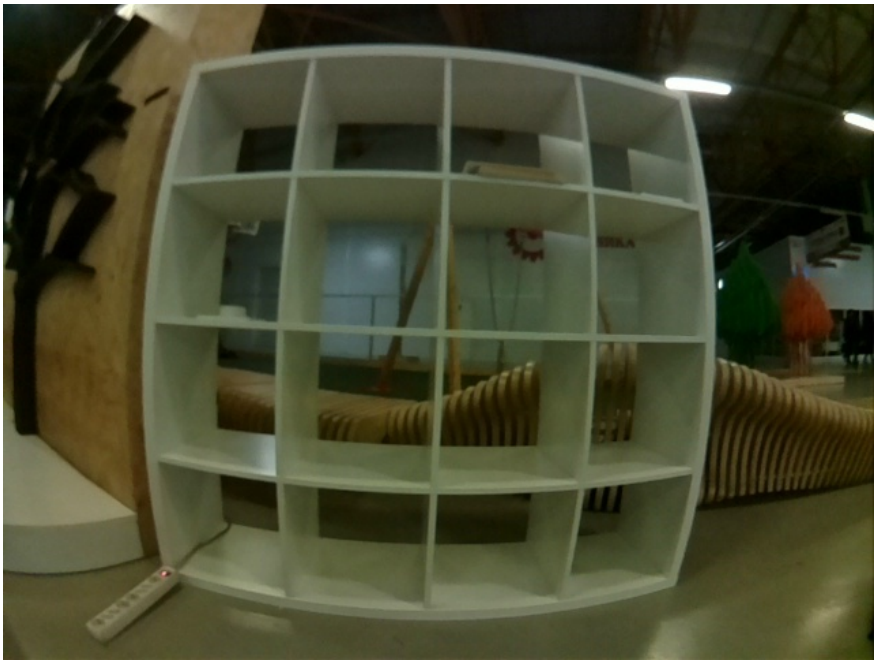
- ***cv2_image***: An image encoded into a cv2 array.
- ***camera_info***: The path to the calibration file.¬

The function returns a cv2 array, into which the corrected image is coded.

If you are using a fisheye camera provided with Clover, for processing images with resolution 320x240 or 640x480, you can use the existing calibration settings. To do this, pass parameters or ***clever_cam_calibration.clevercamcalib.CLEVER_FISHEYE_CAM_640*** as argument ***camera_info***, respectively.

Examples of operation

Source images:





Corrected images:



An example of usage

Processing image stream from the camera.

This program receives images from the camera on Clover and displays them on the screen in corrected for, using the existing calibration file.

```
import clevercamcalib.clevercamcalib as ccc
import cv2
import urllib.request
import numpy as np
while True:
    req = urllib.request.urlopen('http://192.168.11.1:8080/snapshot?topic=/mair
    arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
    image = cv2.imdecode(arr, -1)
    undistorted_img = ccc.get_undistorted_image(image, ccc.CLEVER_FISHEYE_CAM_(
    cv2.imshow("undistort", undistorted_img)
    cv2.waitKey(33)
cv2.destroyAllWindows()
```

The usage for ArUco

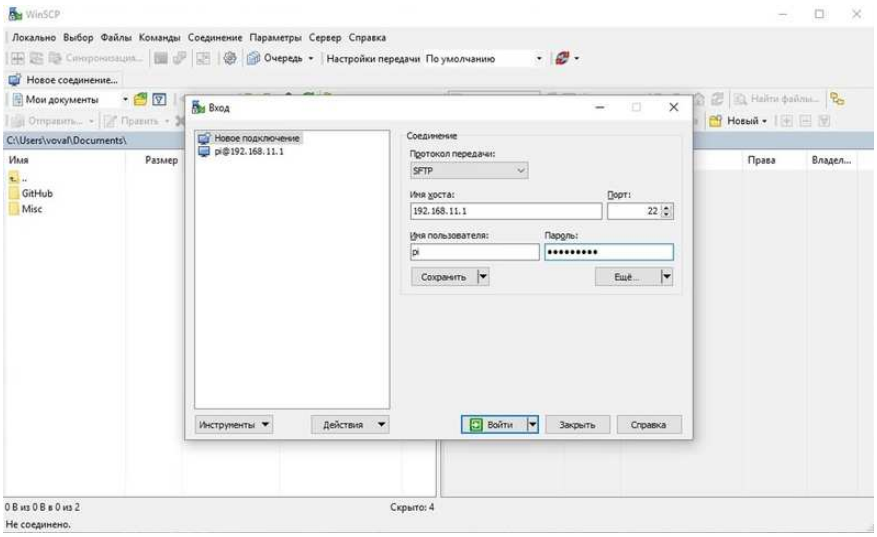
To apply the calibration parameters to the ArUco navigation system, move the calibration .yaml file to Raspberry Pi of Clover, and initialize it.

> Don't forget to connect to Wi-Fi of Clover.

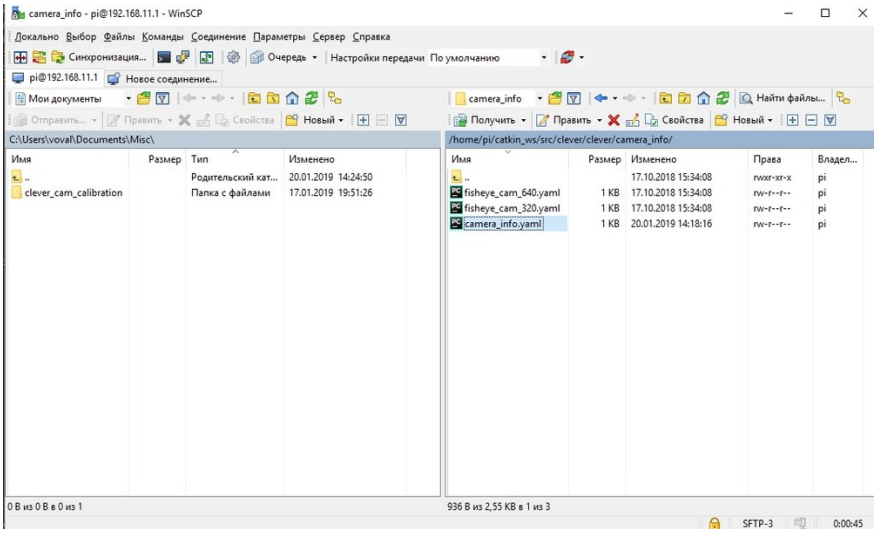
The SFTP protocol is used for transferring the file. This example, WinSCP program is used.

Connect to Raspberry Pi via SFTP:

> Password:

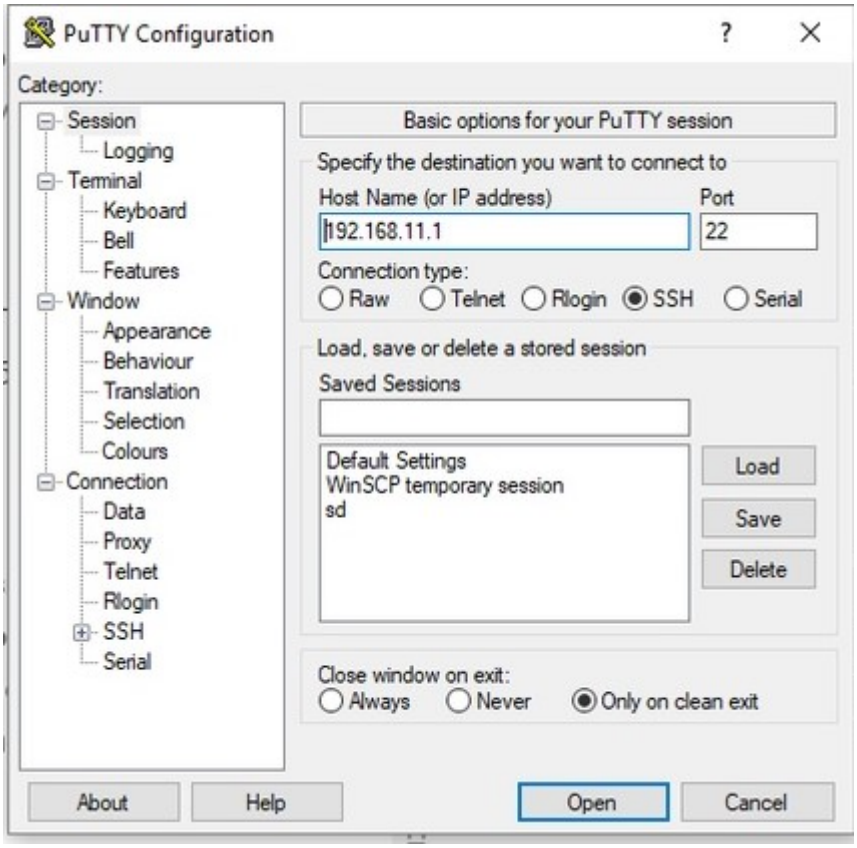


Press “Enter”. Go to ***/home/pi/catkin_ws/src/clever/clever/camera_info/***, and copy the calibration .yaml file to this folder:

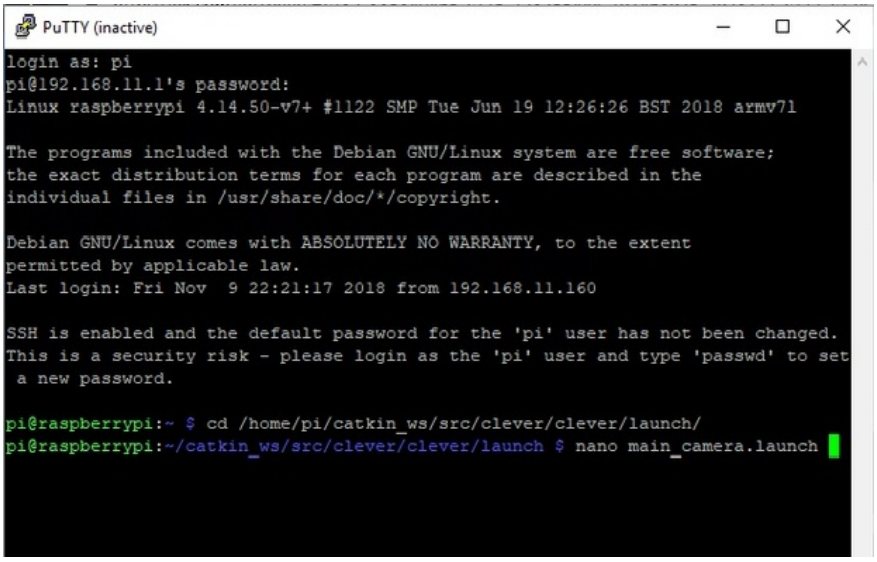


Now we have to select this file in ArUco configuration. Connection via SSH is used for this purpose. This example, PuTTY program is used.

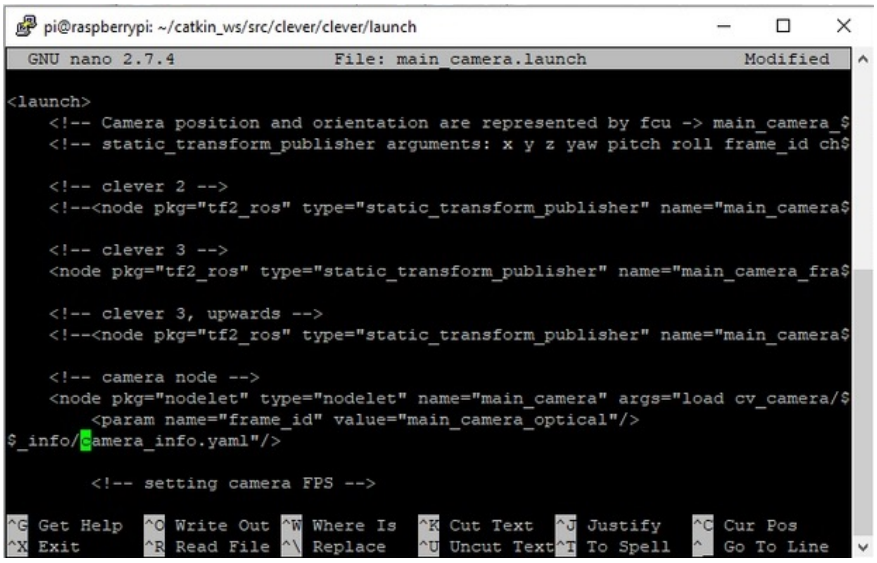
Connect to Raspberry Pi via SSH:



Log in with username ***pi*** and password ***raspberry***, go to directory ***/home/pi/catkin_ws/src/clever/clever/launch*** and start editing configuration ***main_camera.launch***:



In line ***camera node***, change parameter ***camera_info*** to ***camera_info.yaml***:



> Don't forget to change camera resolution.

Recognition of crop types in mass agricultural production

Introduction

Modern agriculture in many countries is becoming one of the shining examples of the rapid and successful introduction of new technologies. Unmanned aerial vehicles are capable of performing a wide range of tasks, among which monitoring of agricultural land has now become a common tool for increasing the efficiency of agriculture. The goal of my project is to write a code for recognizing crop types in mass agricultural production. In the future, from the recognition results, you can design a map of sown areas.

Monitoring

In agriculture, monitoring is necessary to obtain information on the state of land and crops. Based on the monitoring results, farmers or specialists can understand whether crops are sprouting normally, whether there is a threat from weeds and/or insects - pests, what is the degree of moisture in individual areas or entire areas, etc.

Explanation of the code

Import libraries:

```
import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import numpy as np
```

Create some variables:

```
rospy.init_node('computer_vision_sample')

bridge = CvBridge()

color = 'undefined'
shape = 'undefined'
culture = ""
```

To implement computer vision algorithms, it is recommended to use the OpenCV library preinstalled on the Clover image. Create a subscriber for the topic with the image from the main camera for processing using OpenCV:

```
def image_colback_color(data):
    global color, shape

    cv_image = bridge.imgmsg_to_cv2(data, 'bgr8') # OpenCV image
    img_hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV) #[118:119,158:159]

    #detected color
    #print(img_hsv[0][0])
```

Each culture has its unique shade (wheat is golden, buckwheat is light brown).





We describe color ranges for certain crops:

```
#wheat
yellow_orange_low = (38, 110, 150)
yellow_orange_high= (52, 110, 150)

#buckwheat
brown_low = (23, 50, 50)
brown_high= (37, 50, 50)

yellow_orange_mask = cv2.inRange(img_hsv, yellow_orange_low, yellow_orange_high
brown_mask = cv2.inRange(img_hsv, brown_low, brown_high)

if yellow_orange_mask[119][159] == 255:
    shape = shape_recog(yellow_orange_mask)

elif brown_mask[119][159] == 255:
    shape = shape_recog(brown_mask)

else:
    shape = 'undefined'
    color = 'undefined'

if shape == 'brown':
    culture = "greshiha"
if shape == 'yellow_orange':
    culture = "pshenitsa"

image_sub = rospy.Subscriber('main_camera/image_raw', Image, image_colback_colo
```

The script will take up to 100% CPU capacity. To slow down the script artificially, you can use throttling of frames from the camera, for example, at 5 Hz (`main_camera.launch`):

```
<node pkg="topic_tools" name="cam_throttle" type="throttle" args="messages main
```

The topic for the subscriber, in this case, should be changed for `main_camera/image_raw_throttled` .

```
print (culture)
while not rospy.is_shutdown():
    print("color: {}".format(color))
    print("shape: {}".format(shape))
    rospy.sleep(0.2)
```

This program will recognize the culture by its shade. We can use more color ranges to improve the accuracy of the recognition so the drone can recognize more crops.

Examples of color ranges for other colors:

```
red_low1 = (0, 110, 150)
red_high1 = (7, 255, 255)

red_low2 = (172, 110, 150)
red_high2 = (180, 255, 255)

red_orange_low = (8, 110, 150)
red_orange_high = (22, 110, 150)

orange_low = (23, 110, 150)
orange_high = (37, 110, 150)

yellow_orange_low = (38, 110, 150)
yellow_orange_high = (52, 110, 150)

yellow_low = (53, 150, 150)
yellow_high = (67, 255, 255)

yellow_green_low = (68, 150, 150)
yellow_green_high = (82, 255, 255)

green_low = (83, 150, 150)
green_high = (97, 255, 255)

blue_green_low = (98, 150, 150)
blue_green_high = (113, 255, 255)

blue_low = (114, 150, 150)
blue_high = (127, 255, 255)

blue_violet_low = (128, 150, 150)
blue_violet_high = (142, 255, 255)

violet_low = (143, 150, 150)
violet_high = (157, 255, 255)

red_violet_low = (158, 150, 150)
red_violet_hugh = (171, 255, 255)
```

Note that there are two ranges for red because red is at the edges of the HSV color space.

Drones to fight Corona

CopterHack-2021

team: **Drones to fight Corona**

Team

- Daria Miklashevskaya d.miklashevskaya@innopolis.ru
- Yuriy Sukhorukov y.suhorukov@innopolis.ru

Innopolis University, B17-DS-II, B17-RO-I

Introduction

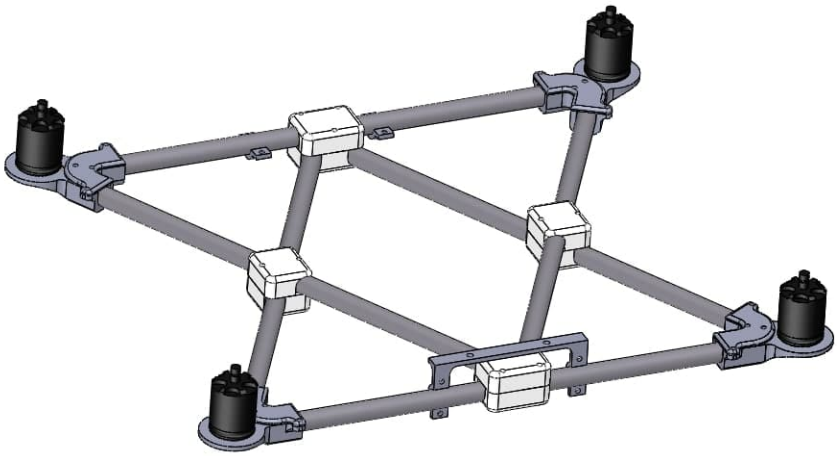
The world faces the worst pandemic of XXI century, which affects lives and well-being of millions of citizens. To slow down the spread of the disease and give the healthcare system to react, people are obliged to wear masks, however, people sometimes ignore those rules, which puts lives of others under the threat.

There are laws in place to enforce mask wearing, but there is just not enough cops to monitor the situation in every mall, bus station etc and fine the law-breakers. We want to contribute to the solution of this problem, but instead of punishing people, we want to provide them with mask with our super-friendly and cute drone.

Custom airframe

Since we do not have the Clover drone, we will use an airframe built and designed by us. Our system is designed to be platform-independent, so it can be installed on almost everything, even VTOL aircrafts.

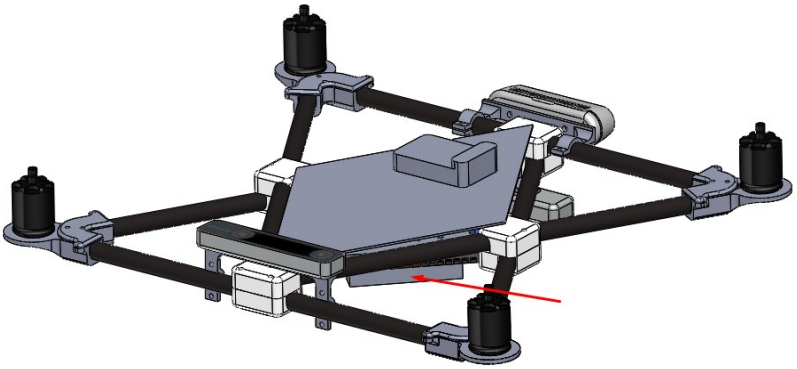
The main idea is to use truss structure, because it works well against twisting and warping, besides, it can be assembled out of carbon tubes relatively easily.



The main advantage of such a system is that it distributes the impact between beams and effectively dissipates it. Engine mounts, however, are not impact-proof because they are specifically designed to break but save much more expensive and not-readily-available engines. This is why mounts are quickly-replaceable (only 3 screws) and made of cheap PLA plastic.

The space inside the central rhombus is occupied by the on-board equipment: batteries, PX4 flight controller, Jetson Xavier NX / AGX, power electronics, sensory equipment.

As it is shown on this picture, computers can be mounted on the bottom and completely protected by legs and the truss structure from any collision damage. Jetson AGX is marked with arrow. Almost invisible, isn't it?



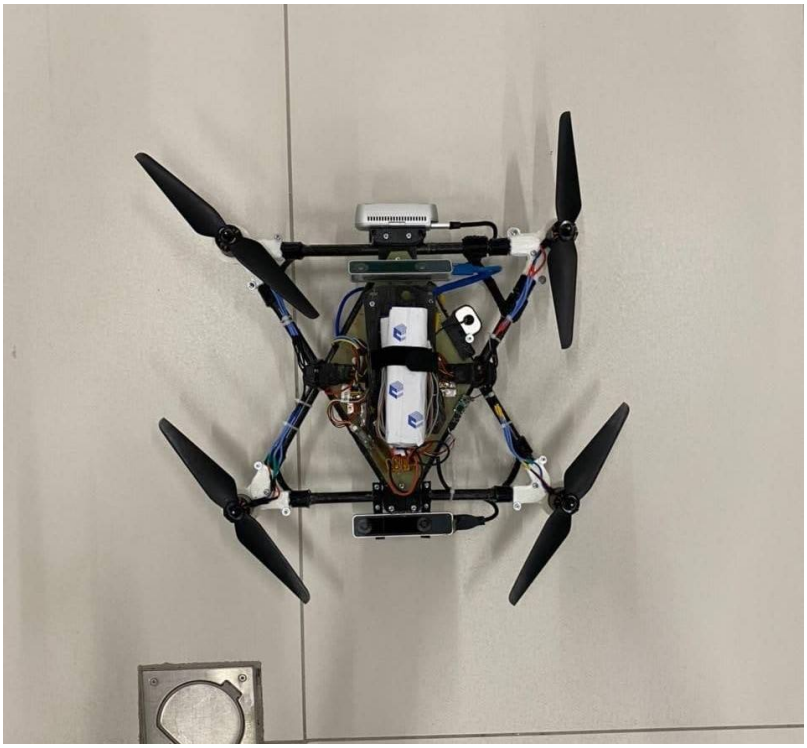
All sensory equipment, like cameras, rangefinder, etc can be easily mounted on the beams with special bracket connectors. This type of connection provides flexibility because you can fine-tune camera angle or position before tightening screws and fixing it firmly in place, which is especially relevant for tracker-cameras.

We used one T-265 camera for visual odometry and one D-435 depth camera for both video input for neural net and for map-building (collision avoidance). T-265 suffers from "odometry drift" especially when engines are beat-up, which

eventually happens after a number of crashes, so we have incorporated dampers to solve this problem.



Finally, the drone with all equipment installed looks like this



Software

Thus far we discussed things which are specific to our custom airframe. Things we are going to discuss next are applicable for Clover drone as well. Our software is containerized so it can be launched on every platform that supports Docker, be it Windows machine, Linux machine, Jetson or Raspberry Pi.

We have split our drone software into two modules:

- Pipeline, which manages communication with ROS.
- Neural net module, which talks with the pipeline via sockets.

The reason for this is simple - ROS supports only Python 2, and I do not feel like I'm building ROS with Python 3 because it is a bit troublesome. Our neural net, on the other hand, uses Pytorch which is Python 3 only, so the only way to run them both is to use inter-process communication of some sort.

Apart from compatibility issue, such an arrangement allows us to run inference module anywhere we want, e.g. on more powerful desktop PC or even somewhere in the cloud (google collab? Why not!).

This means that we can make our drone lighter by excluding heavy on-board computer and replacing it with something light like Raspberry Pi. Pipeline image is made as lightweight as possible, so it should be runnable even on really weak computers.

More detailed instructions on how to build and run our software are available in our Gitlab repo.

Neural net [1]

We use 3rd version of YoLo neural network, pretrained on custom dataset for 50 epochs.

It runs 10-15 FPS on Jetson NX, which is enough for our task

Exploration and collision avoidance

To make our drone useful and to operate it safely, we should somehow make the drone aware of its current position and surrounding objects. There are two ways we can solve this problem

- Use pre-built map as a ground truth and then calculate the position with e.g. Particle Filter
- Build map on-the-fly, while avoiding collisions and moving towards the goal

The second approach is more robust, because it does not rely on any external map, which can be erroneous or just missing and hence we opted for it.

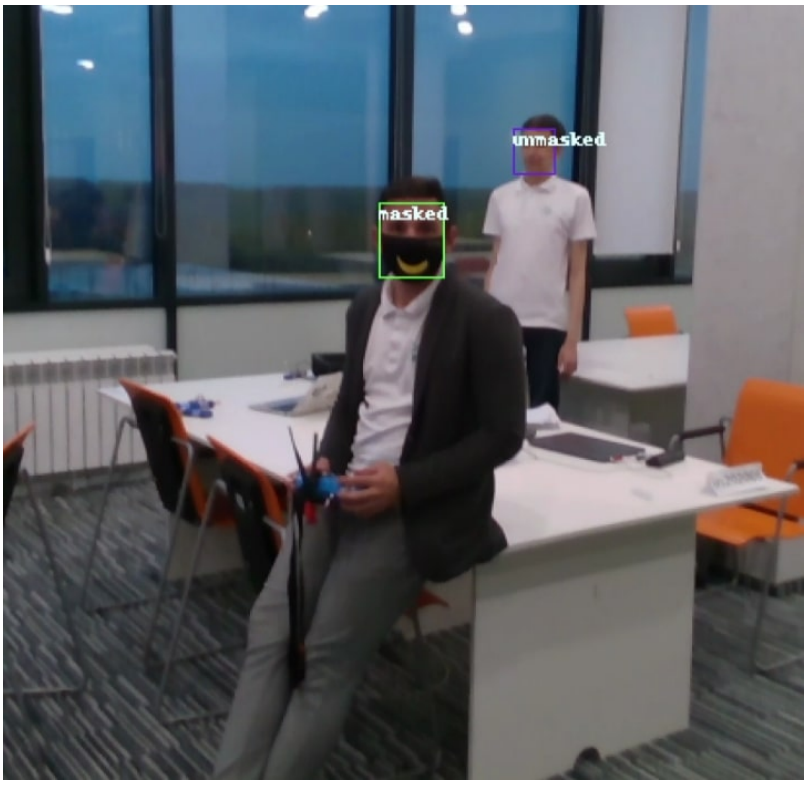
We use a path planner, described in [2]. In this paper Receding Horizon Next-Best-View Planner is presented, which uses Rapidly-exploring Random trees to navigate and explore the environment. It yields the following results, here is the occupancy map and the corresponding tunnel as it is seen by human being:



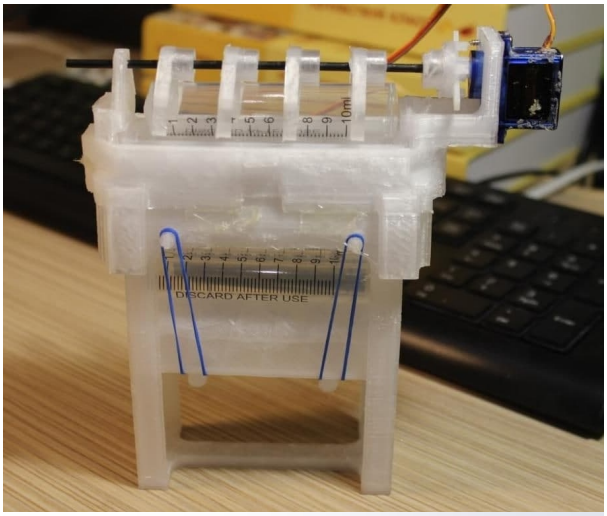
The algorithm is lightweight, so even the small computer like Latte Panda can run it with high enough frequency, and since it is CPU-bound, it will not compete for resources with the neural net, which is almost entirely GPU-bound.

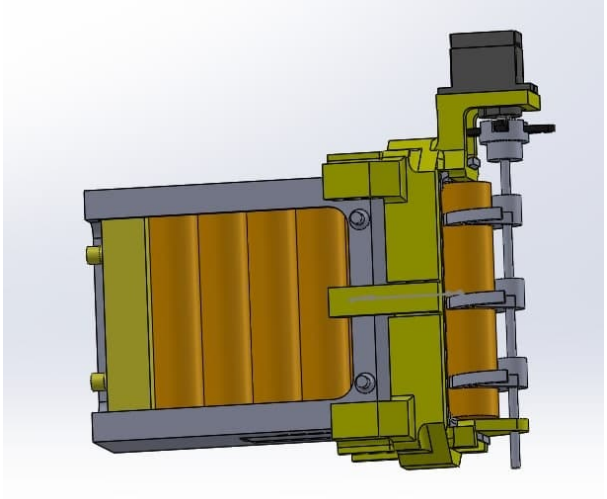
Mask release

Detecting people without masks is cool, no doubt.



But we want not only to detect them but to give him a mask as well, so, we have built this system that can give a mask to person.





This device looks like a regular firearm mag, and functions exactly in the same way. Masks can be loaded into containers made out of 20ml syringe barrels.

This device needs further engineering because current iterations are too fragile and unreliable, probably the best solution will be to use linear actuator and push the "casing" out of the action, like in actual firearm.

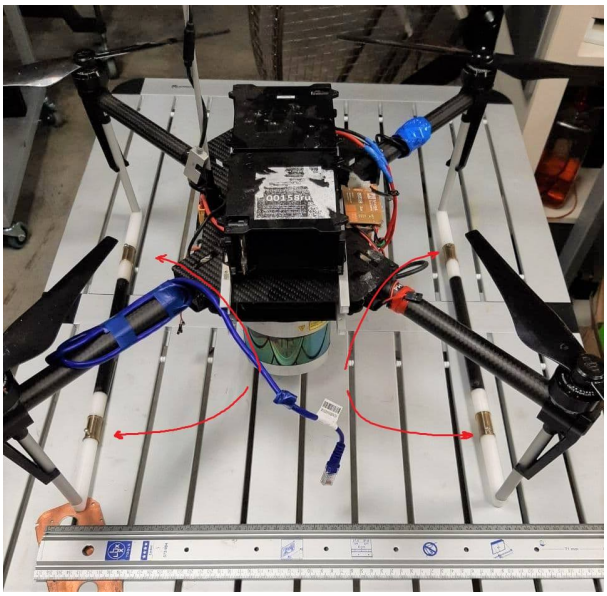
3D models

All the 3D models used to build this cute drone can be found in our gdrive.

Final thoughts

We all hope that Corona crisis will soon be over, and when it will be finally over, our drones will be still useful. We can deliver some small objects, like cosmetics or shaving blades to the customers' door, the task that currently is done by a human courier. This service (with shaving blades), when a guy comes and brings a new set of shaving blades every week is very popular in US and UK, so why not try to automate it.

This is an MVP, so some improvements are to be done. For example, payload refill and battery swaps are done manually for now, but this task should be automated. Actually, some work is already done in this direction:



Drone lands on a landing platform with special hooks which will connect to four metallic contacts (highlighted) and charge the battery. In future we want to change battery, not just charge it, but the mechanism is very complex and requires making custom batteries and battery mounts, also, it constraints placement of cameras, payload and so on.

For the general-purpose delivery drone, the working principle and hardware will remain the same, but software (neural net detector) will need an update.

Stay safe folks!

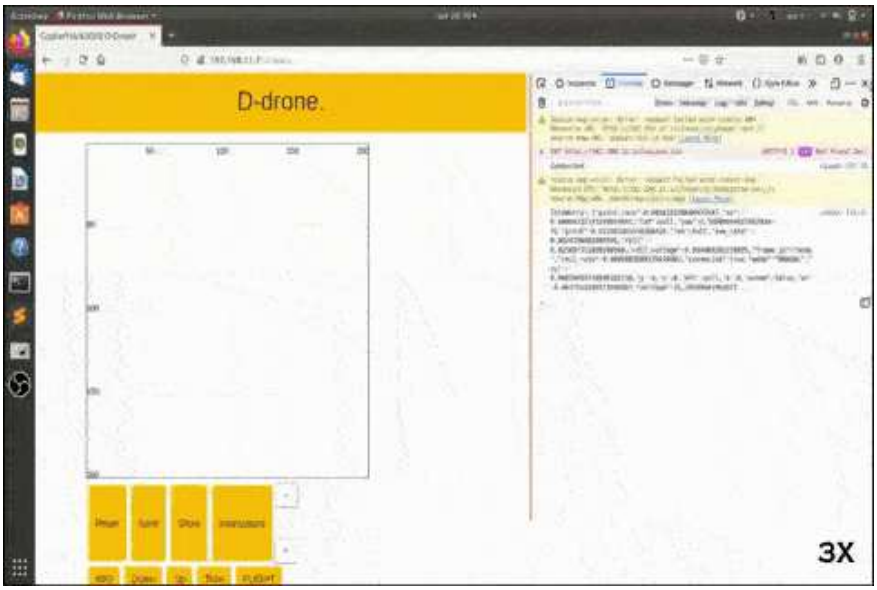
References

> [1]. Massagué Respall, Victor & Devitt, Dmitry & Fedorenko, Roman. (2020). Unmanned Aerial Vehicle Path Planning for Exploration Mapping. 1-6. 10.1109/NIR50484.2020.9290232. ↩

> [2]. Nisarg Pethani & Harshal Vora. (2020) https://github.com/NisargPethani/Face-Mask-Detection-using-YOLO-v3 ↩

D-drone Copter Hack 2021 by AT Makers

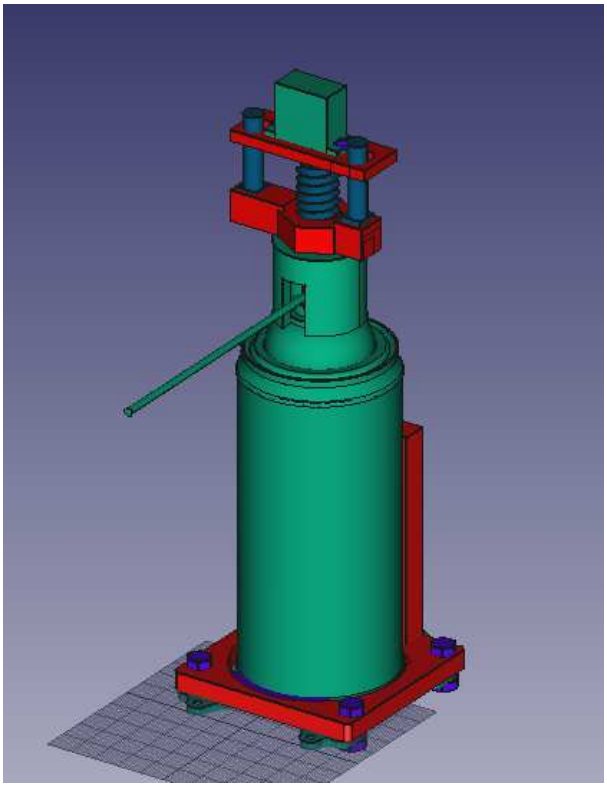
CopterHack-2021, team **AT Makers**.



Intro

People strive to teach artificial intelligence everything they can do themselves. We are taught to draw from childhood. And why not teach the drone to draw? At the moment, copters and graffiti are gaining popularity. So we decided to combine them.

Models and assembly





To complete the project you need to have in stock:

- spray paint
- clover 4 kit
- servo MG90S
- 3D printer

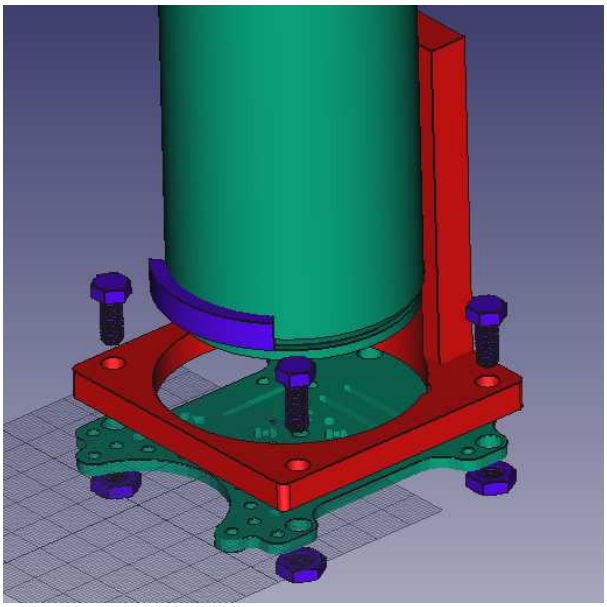
- spray can extension straw
- velcro
- 4 long screws and nuts M4 or M3
- 2-4 short self-tapping screws M4 or M3.

Download and 3D-print details:

- holder
- screw
- rack_holder_with_nut
- rack x2
- servo_holder.

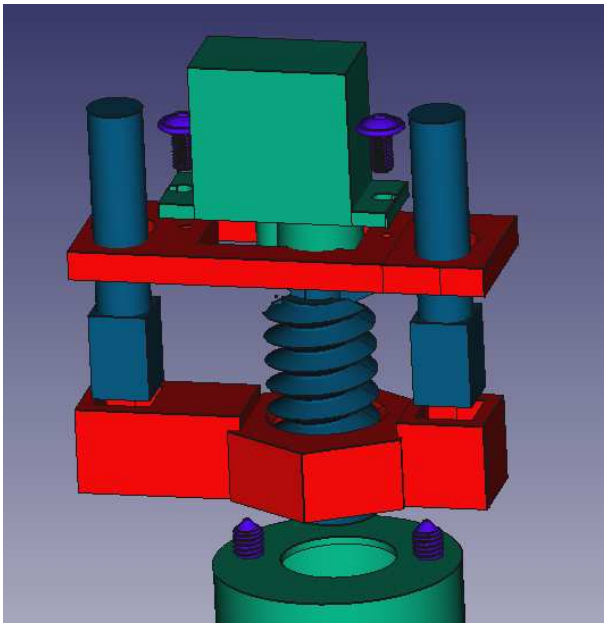
Spray holder. The spray holder is attached to the deck with 4 screws and nuts. To fasten the can to the holder, we used a tape with velcro. With 4 nuts and screws, we fix the drone's upper deck with spray holder.

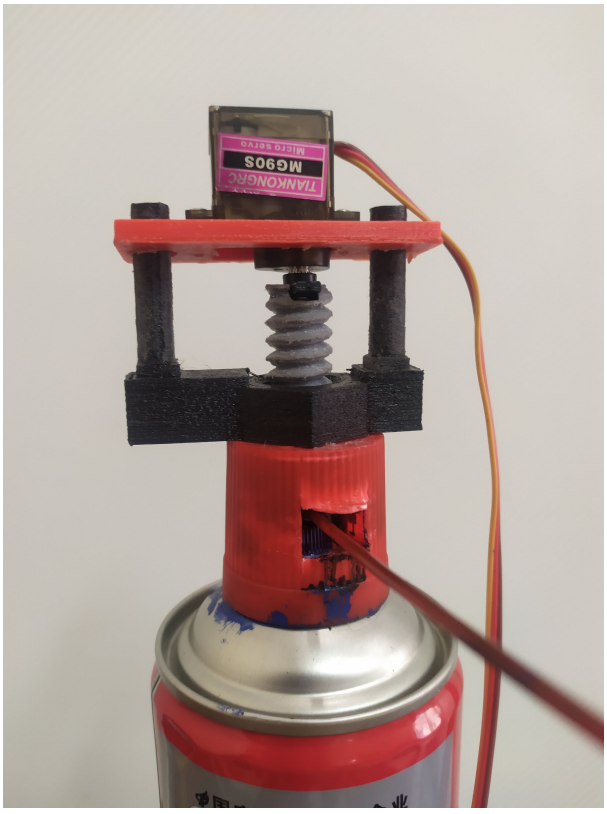
Holder weight: 90g.



If the diameter of the can is less than the diameter of the holder, we use the part in the form of an arc, with the size of the difference between them. This helps us to fix the spray can firmly.

Pressing mechanism. To push the valve, we will use a screw drive with a fixed nut. A bar with holes will be attached to the servo, which will include the racks attached to the nut. This helps the servo to move only on one axis, up and down. We also modeled the cap for the spray can button, since the surface of the nozzle is uneven.





Before launching

Configuring the servo scripts

Before starting the copter, you need to download servo.py and move it to RPi. You can simply copy and paste using the clipboard. Or copy it using the scp command. For instance:

```
scp servo.py pi@192.168.11.1:/home/pi
```

Then run the following commands remotely on the Raspberry Pi:

```
sudo pigpiod
python servo.py
```

Configuring the Web interface

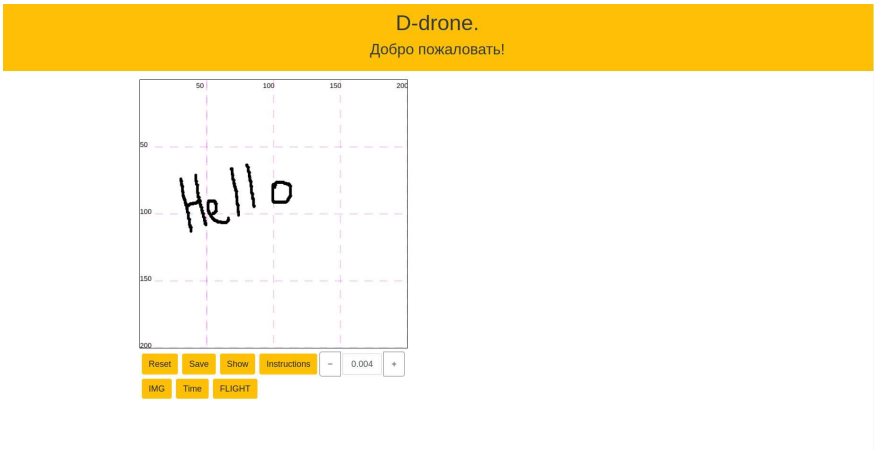
Download this repository in .zip format. Copy to RPi and unpack using the following commands:

```
scp visual_ddrone-master.zip pi@192.168.11.1:/home/pi
cd catkin_ws/src/clover/clover/www
unzip /home/pi/visual_ddrone-master.zip .
mv visual_ddrone-master ddrone
```

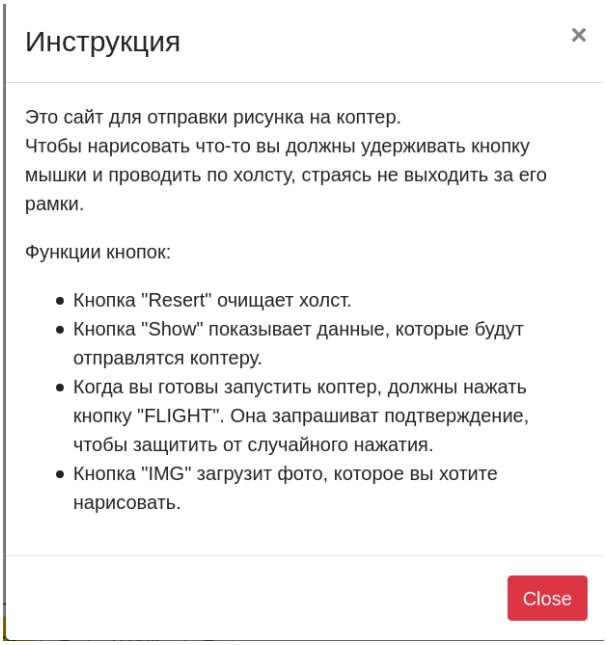
Now to open the web interface, click on the link http://192.168.11.1/clover/drone.

Web interface

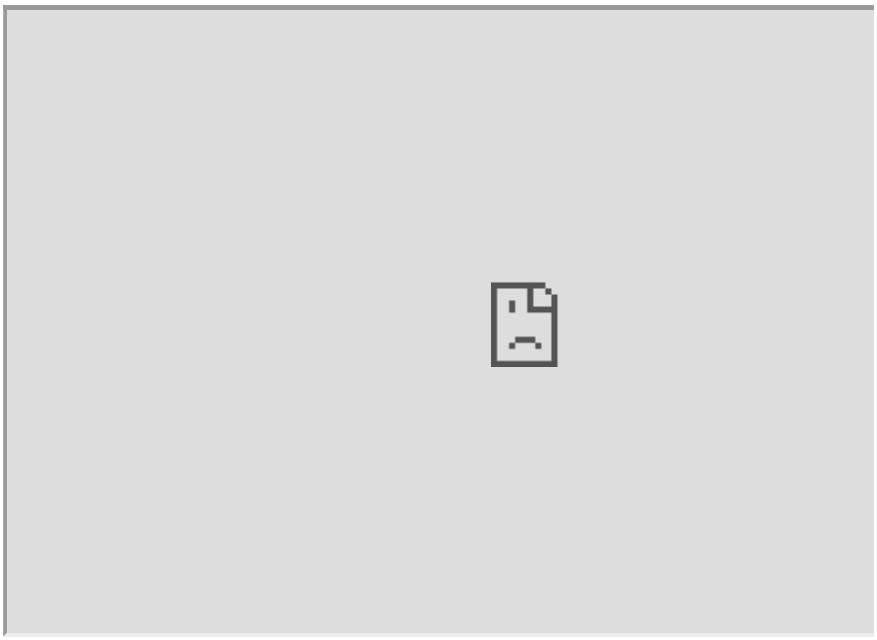
Our drone is launched via website. The web interface allows you to draw and encode what you draw in G-code. The coordinate data will be transmitted for further processing and execution by the copter.



We pick the web interface to control the copter because it is easier for the user.



Flights



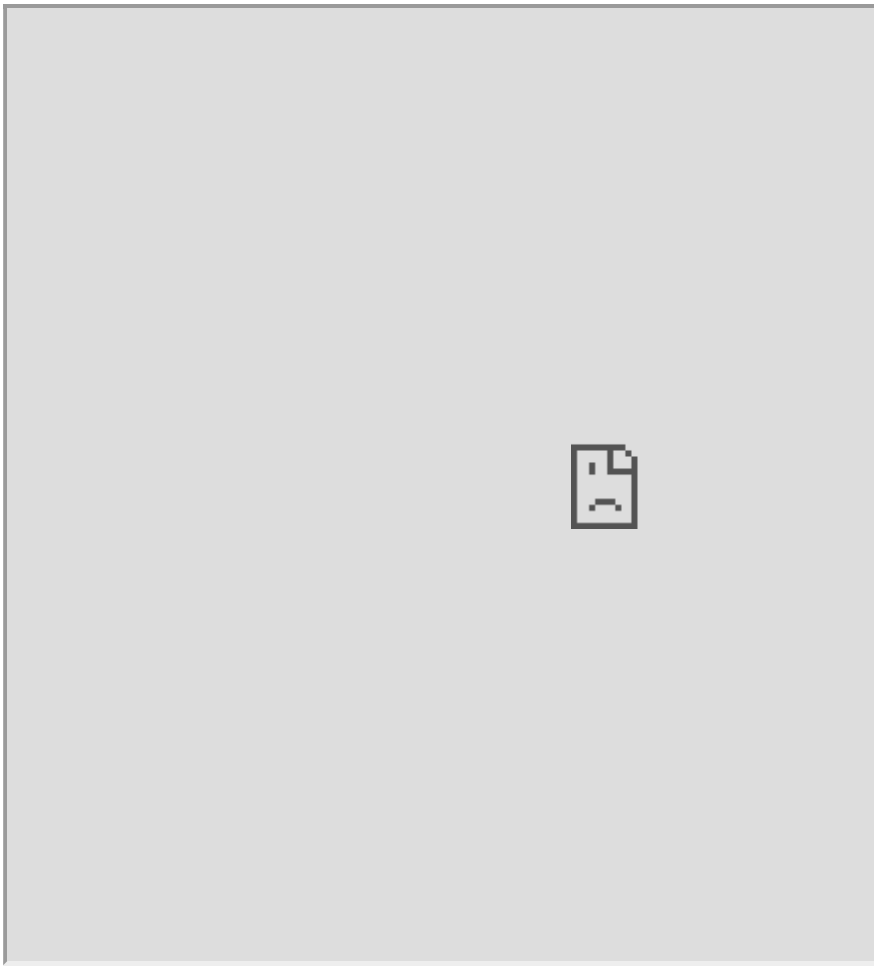
Special Thanks

Project was created with financial support of International Ala-Too University.

ALA-TOO INTERNATIONAL UNIVERSITY

3D-printed Generative Design Frame

ADDI Copterhack 2021 Project



Contact

Website, Mail, Telegram: @danielhonies.

Introduction

At the Aachen Drone Development Initiative we aim to develop a new frame for the clover drone by implementing the latest state of the art CAD-Design techniques as well as advanced manufacturing methods.

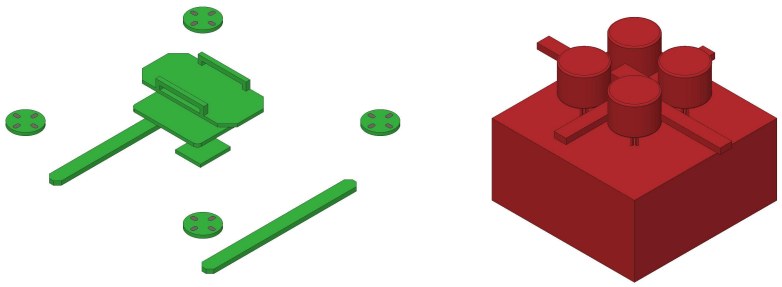
Three main goals have to be taken into consideration when designing a new frame:

- Decreasing the Weight
- Improving Durability
- Increasing Safety

For the first stage of the design we will focus on the first two points.

Software

For designing our drone we use Autodesk Fusion 360. It comes with a generative design feature. This makes it possible to create rule-driven designs. First the preserved geometry is defined. Usually this includes all kinds of mounts like motor mounts, flight controller mounts, RPi mount etc. Then obstacle geometry is defined. This for example includes space for the propellers and the airflow. After that forces are defined. Then Fusion 360 will automatically calculate optimal connections for the aforementioned preserved geometry.



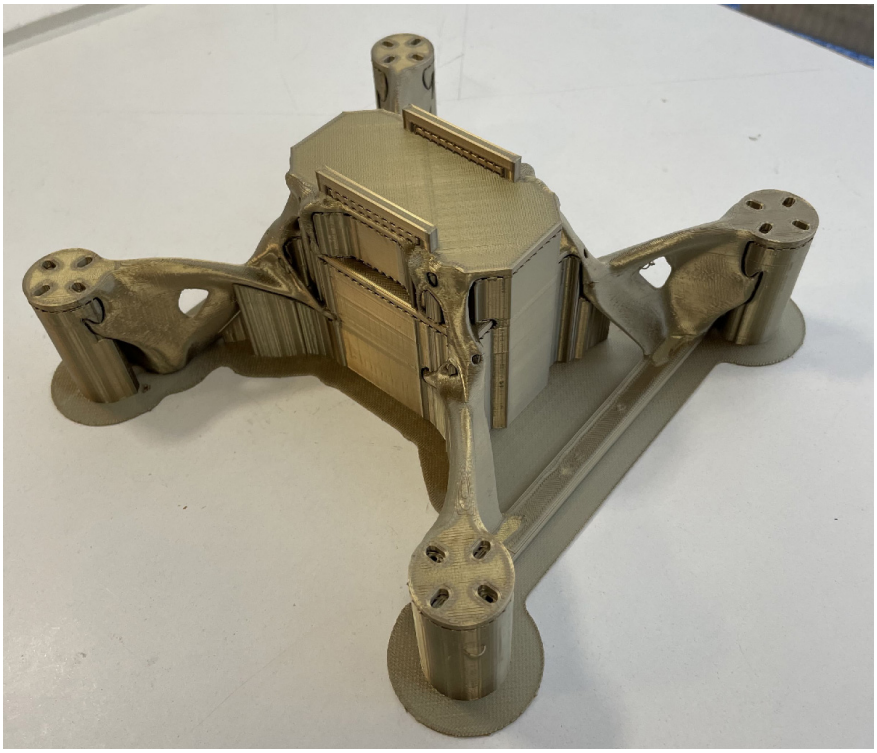
Prototypes

Prototype 1

After printing the first version of the frame we discovered the following problems:

- Bad Filament: Layer Adhesion of the Filament was quite bad resulting in a not very rigid model
- Support Structure: The support structure for the frame is very complex and the parameters used in the slicer resulted in it being unable to be removed without destroying the model
- Arm Strength: Some parts of the arms to the motor mounts were very thin, resulting in them breaking easily and removing the support structure resulted in breaking them

To conquer those problems we made several changes. We increased the minimal thickness for the generated structures and generated a new model. We changed the settings in the slicer so that the support structure could be removed easier as well as changed the infill structure. Finally we changed the filament and increased the printing temperature. Further we concluded that printing with a water dissolvable support structure would be optimal, however as of right now we don’t have access to a printer capable of that.



Prototype 2

This prototype took 48 hours of printing and used 277 grams of filament including 100 grams for the support. Installation of the components is very easy as no other tools than a screwdriver are needed. This prototype was the first to take flight in January 2021. Please see this video.

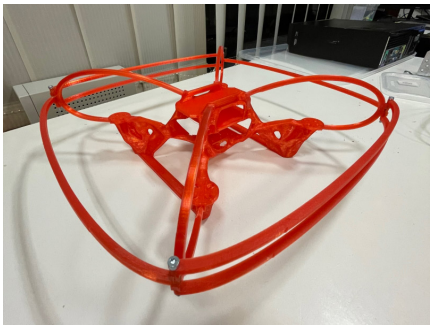


Prototype 3

This prototype is even more optimised than the last one. Excluding support the model only weighs 141 grams. For this version we have also developed a prop guard, which weighs around 80 grams. Weight of the drone with the prop guard and a 2200mah battery is under 700 grams. Flight testing in the following videos shows the effectiveness of the prop guard. We also did some drop tests with this model and figured that a drop height of around 1 meter can be sustained. We plan to optimise this while sacrificing a bit of weight in February.

Videos:

- Flight Test
- Prop Guard Test




Final Prototype

In this final prototype we have changed the preserved geometry on the bottom to form a rectangle for added stability. We have also changed some of the forces on the points we observed breakings in our previous tests. We have also updated the prop guard to make it more stable and increased the area around the screws, so it would break harder. The frame without the prop guard weighs only 150g making it significantly lighter than the default frame.



Benefits

We see the following benefits with our design over the traditional clover frame design:

- Tools needed for production: only a 3D printer is needed compared to laser cutter, cnc and 3D printer
- Single material reduces supply chain complexity and reduces cost. Filament is cheap and only around 400 grams are needed for the full frame and the prop guard. This should reduce the cost to under $5.
- The unibody design saves weight and much less screws are needed. This also reduces costs.

- Easier adaptability: Anybody can change the frame to their desire and include their own adapters and mounts. It can be printed by any standard sized 3D printer. (Ender-3, Prusa etc.)
- Manual labor reduced: Printing is easy to automate, packing is significantly reduced as less screws need to be counted etc.

Conclusion

In our work for this years competition we presented a new way to design and manufacture drone frames. By utilizing state of the art methods of CAD programs we are able to optimize the weight and shape of drone frames in a way that for the first time it is feasible to 3D print them. In total we have printed around 10 frames and presented a few of them above and the challenges we faced with them. This iterative process to frame design was only possible due to the fact that only a 3D printer is needed and the filament being very cheap. Due to the open source nature of the project and this report giving a short introduction into generative design, we hope that many people feel inspired to check out this new method of designing and producing drone frames.

CAD Files

You are welcome to test out these frame yourself and you are free to modify them in any way, shape or form. We would appreciate feedback and encourage submitting your modifications in a pull request so other people can benefit from this open source development. The CAD Files can be found on our GitHub Page.

Retail Drone - CopterHack 2021 article

CopterHack-2021, team: Bennie and the Jetson TX2.

My project is a drone that can scan a store shelf and determine if the representation of a specific brand is adequate for what the vendor paid for. In most retail stores, a brand will pay for space on a shelf in order to be more eye catching to the customer and eventually get more sales, which is why bigger brands are generally seen more. However because of either employee ineptitude or general non-compliance up to 15% of a brand's space on a shelf that it paid for can be lost, which can lose the brand upwards of 5% loss in overall sales.

Several apps have been released to fight this such as Clobotics, however they all require an employee to go around manually and take snapshots. I wanted to build a system where a shop owner could simply place the drone at a starting point, click a button and have their inventory done automatically within minutes.

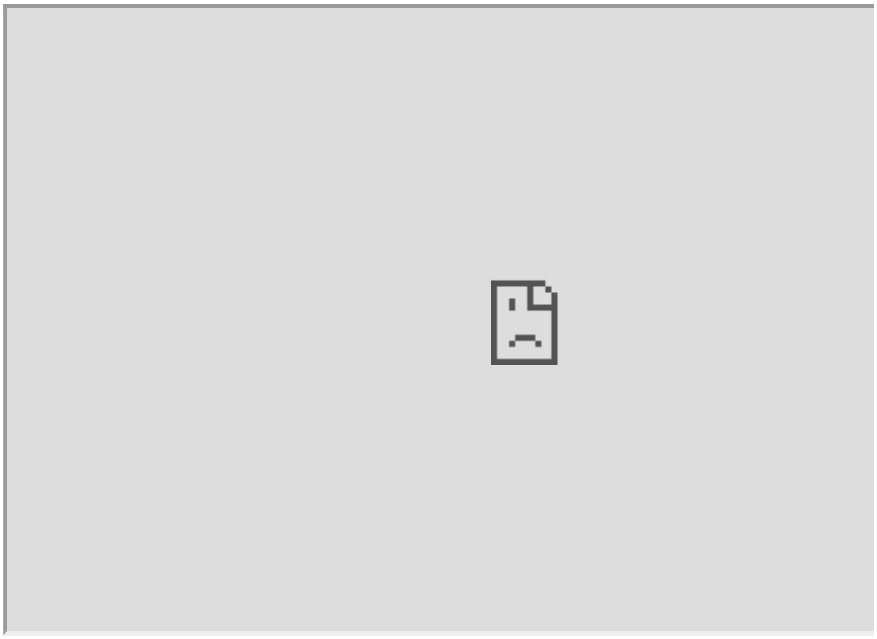
Since all inventory in a shop is usually on the shelves and I wanted to not have the drone rely on GPS I fitted the Clover's Pi camera in front for object detection and navigation. Rather than using map-based navigation since one can't expect a shop owner to place markers perfectly on the ground, opencv2 has a built in ArUco marker detection method that identifies and draws a bounding box around all markers in frame. So once the drone takes off from the starting position, it identifies where the marker is in frame and uses the Clover's `navigate()` method to center the marker and its area to place the Clover approximately 1 metre from it.

What's expected is that the shop owner places markers at every different brand's section in ascending order down an aisle. The drone has a specific marker ID it's looking for and once it's centered itself on the nearest marker, it determines its ID and moves either left or right depending if the identified marker is lower or higher than the target. This way the Clover moves down an aisle stopping at each marker until it reaches the target.

In order to identify the number of items of each brand, I've trained a TensorFlow model and stored its metadata in a caffemodel file for the Python script to pull from. Every time the drone stops at a marker, a method is run that applies a convolutional neural net to the frame and determines how much of a specific item (right now the model is trained to recognize soda cans but can easily be taught other items) it at said marker (remember that each marker is to be placed at each brand's individual section). In the video below, you'll see a POV video of my drone with two markers placed on the wall in my garage. What happens is that the drone takes off, centers itself on the first marker and as its ID is 140 and the target ID is 138, it's driven left to the second marker which has a soda can hung next to it. It then identifies the soda can, conveying the number of cans of that brand to the shop owner before centering itself on the marker and landing due to it reaching the target marker.

My main goal with this project was accessibility. I wanted to make a drone that could navigate purely based on its camera with computer vision tools that aren't exclusive to MAVLink or ROS drones. By fitting the camera to the front not only would a user be able to get more functionality out if it than simply map-based navigation but would be able to make more useful applications such as shop keeping or security through facial recognition. With alterations to the left, right, front, back, up and down commands this script can be applied to any hackable drone and requires little to no prior experience to use.

Check out the video below to see it in action:



Contact info

- Email - jadenbh12@gmail.com
- Telegram - @jadenbh12

DroMap: The Indoor Mapping Drone

CopterHack-2021, team: **DroMap**. E-mail: officialdromap@gmail.com.

Team:

- Shouq AlQahtani
- Ameena AlMansouri
- Noof AlMarri

Abstract

In the modern era, the world is witnessing a magnificent development in the field of architecture and interior design. Due to architectural development, the current measuring tools such as metal tapes and leaser meters became insufficient for assisting both architects and interior designers in taking the measurements for buildings and facilities. Because the accuracy of obtained readings depends on the professionalism of the users and the nature of these tools is unidirectional, the measurement taking process becomes less efficient in terms of time and labor. Since drones have played an essential role in revolutionizing the world of science and automation due to their use in a huge number of daily life applications, an introduction of indoor drones in the field of mapping and architecture is indispensable. Hence, the DroMap project proposes an autonomous indoor drone that can navigate autonomously and create a map of the indoor environment along the way. For the aforementioned purpose, a LiDAR sensor is used to collect data of the indoor place which is sent to a host computer. Afterward, simultaneous localization and mapping algorithm utilizes these data to pave the way for creating a 2-dimensional map. This autonomous indoor mapping drone system is not prone to inefficiency and human errors like in manual mapping and has the potential to take indoor mapping to the next level in the near future.

Motivation

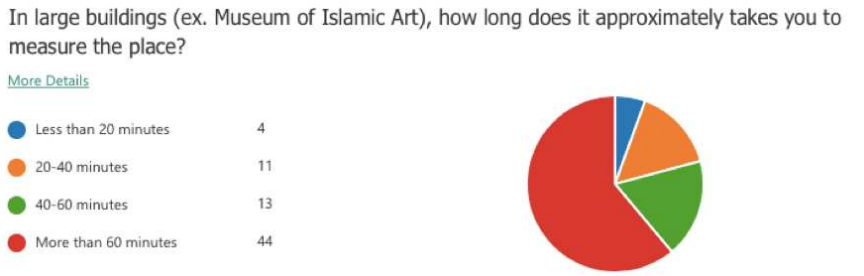
Problem Statement

Architects lives are constantly in danger due to the nature of their work as they are supposed to enter buildings without knowing their structure, these dangers could potentially threaten their lives and can lead to many issues. According to [1], there are 1.2 deaths per 100,000 architects and that job is ranked 19th among the most dangerous jobs in the United States. One example of a fatal accident is the accident of Bruno Travalja which happened in 2016, this architect fell from the 48th floor of a building while taking measurements. In addition to being dangerous, the process of mapping an indoor environment is time-consuming, especially in transferring the raw measurements into a 2-dimensional map [2]. Therefore, the need for robot assistance in mapping and measurement taking

processes reaches the peak. Autonomous indoor mapping using drones or robots is considered an important tool where the drone can reach different places which are inaccessible to humans due to space constraints or security reasons [3].

The use of robots has increased dramatically within the past decade due to their enormous potential in both civil and architectural applications. Specifically, in designing and building robots for mapping enclosed buildings. Even though most of the works were implemented on unmanned ground vehicles (UGV), the current experimental use of unmanned Ground vehicles for indoor mapping suffers from a few shortcomings. Particularly, most implementations suffer from low performance regarding time consumption and have difficulty accessing narrow places. Since UGVs have limitations in terms of time consumption and navigation rigidity, in the DroMap project we decided to use drones as a replacement for UGVs to map indoor sites. This is because drones are unique in their ability to traverse any 3D interior space without any restrictive concerns regarding space architecture. Additionally, since these vehicles are not required to remain on the ground, aerial vehicles can fully explore the extent of the indoor space, regardless of their interiors. Furthermore, unmanned aerial vehicles can access difficult to reach areas.

A questionnaire was conducted for this project to study the need for an indoor mapping drone which involved 72 architects and interior designers. The following question was asked to have an estimate of the time taken by the targeted category for taking the measurements of a large building. According to the survey results, 61% of the sample consume more than 60 minutes to measure a large building. This shows that the measurement taking process is time-consuming.



Technical Challenges

- The positioning system calculates the odometry data based on the laser scanner poses. This might misestimate the drone's position with respect to the surroundings.
- The LiDAR readings could be infinite if the distance between the LiDAR and the surrounding walls exceeds the LiDAR range.
- The communication between the Raspberry Pi and the PC relies heavily on Wi-Fi. Therefore, any loss in the Wi-Fi signal would terminate the communication between the drone and PC.

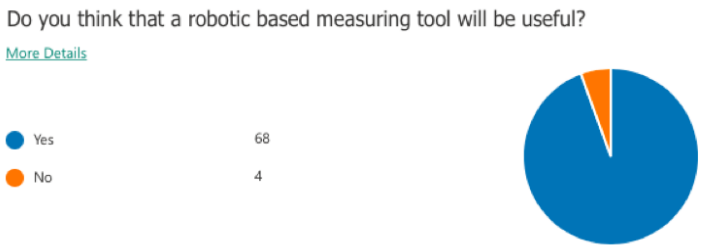
Non-technical Challenges

- The indoor environment could be full of obstacles, which impedes path planning process.
- The mirrors, windows, and glass doors may affect the accuracy of the map as they are not detected correctly by the laser pulses.

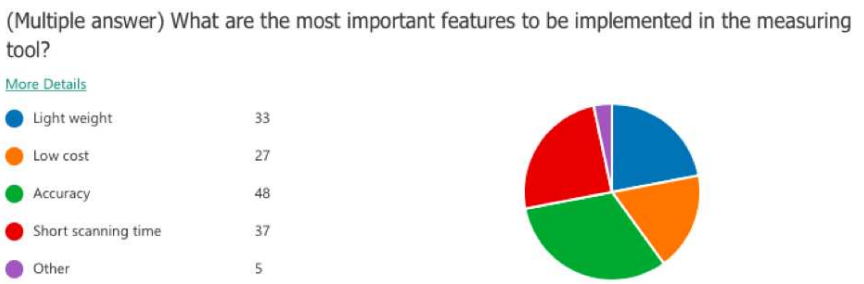
Project Significance

Measuring a room or a full building along with transforming the collected data to a full map is time-consuming and requires massive effort. DroMap helps architects and interior designers to measure and generate a fully constructed 2D map with less time and effort. To help us understand the problem better, we conducted a survey to assess the need for an indoor mapping drone. This project will provide a great advantage for architects and interior designers as it would save time and effort in the map construction process. In addition, it will assure great cooperation from both the computer and architecture fields.

Generally, the process of mapping an indoor environment is composed of two phases; the first phase is the measurement taking phase and the second phase is the map drawing phase. However, the project introduces another way to create a map that is faster and requires less effort; as the measurements of the surroundings will be taken by the system once it is activated and processed by Simultaneous Localization and Mapping algorithms (SLAM) for building and updating maps as well as positions of an unknown environment in robotics in real-time. According to the survey, 94% of the sample agreed that it would be useful to have a robotic based measuring tool. Therefore, the proposed solution will successfully assist architects and interior designers in mapping indoor areas.



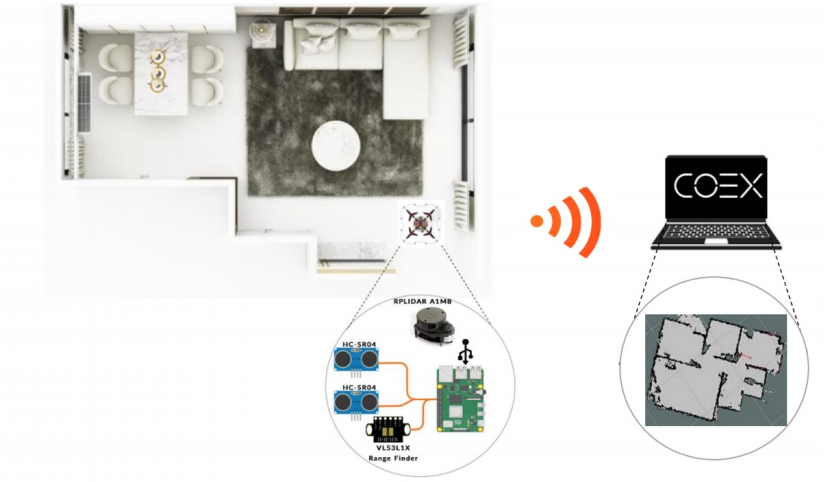
Based on the conducted survey, accuracy is the most important characteristic to be satisfied with the project. The below figure demonstrates that 48 of the sample sizes care about having high accuracy. Moreover, the second most important feature to be reached is having a short scanning time, which highlights the importance of the project.



Furthermore, since the project employs multiple concepts related to indoor robots and indoor data processing, it can be extended to assist other fields in Qatar rather than the architecture field only. For example, this project could be a great step towards training drones to handle different tasks related to search and rescue such as entering buildings on fire or finding a missing person in indoor places. This will serve to develop more technologies to process the indoor data in various environments and conditions, also to develop drones that are capable to operate in indoor areas with different functionalities.

Proposed solution

DroMap project consists mainly of two major components: the drone and the drone add-on. The drone is responsible for the physical movement of the entire system. The drone add-ons consist of necessary sensors for mapping, path planning, and mounting equipment such as Raspberry Pi 4, RPLiDAR A1M8, Sonar, and range finder. The Raspberry Pi collects the data from the sensors. While the data is being collected by the Raspberry Pi, the Hector SLAM will process these data in real-time to formulate 2-dimensional maps. After that, the map will be sent wirelessly to a remote PC and visualized through RVIZ software tool.



Hardware/software to be used

Hardware selection

COEX Clover Drone kit

Clover is a complete STEM educational programmable drone kit which includes unassembled quadcopter with four propellers and open-source software.

- Limitless possibilities of a fully programmable drone (Open Source).
- Drone can operate stably without GPS.
- The Clover platform exploits the ROS framework.
- Made especially for Indoor flights.

Slamtec RPLiDAR A1M8

LiDAR is low cost 2D 360° 12m scanning sensor.

- Omnidirectional Laser Range Scanner 360°.
- Compatible with ROS.
- Very high sampling Rate 8k times, Considered as one of the Highest in the Current LiDAR industry.
- Ideal for indoor Navigation and Localization using UAVS.

Raspberry Pi 4 Model B

Raspberry Pi is a single-board computer which is used as a companion computer.

- Low energy consumption.
- Connect the drone over Wi-Fi.
- Responsible for flight autonomy.
- Access and issue commands to peripherals.

VL53L1X RangeFinder Sensor

Laser Ranging Sensor Module Rangefinder. One of the smallest time-of-flight 940 nm laser VCSEL. Measuring absolute range up to 4 meters.

- The Range Finder Optical Ranging sensor is an integrated sensor with embedded infrared, eye-safe laser, advanced filters and high-speed photon detection arrays.
- Range finder Supports 400cm sensing range, suitable for many applications.

Software selection

Robot Operating System (ROS)

A framework which runs on Linux operating system, and will be used as a firmware to control and monitor the system.

COEX Virtual Machine

A Linux operating system that has a pre-installed ROS along with some necessary dependencies and packages in addition to a pre-configured Gazebo environment.

Gazebo

The simulation tool that will be used to test and try different mapping and automation approaches.

Visual Studio Code

A text editor to write python scripts to program the drone.

RVIZ

A visualization tool to visualize the LiDAR readings.

QGroundControl

QGroundControl supports full flight control and mission planning for any MAVLink enabled drone.

Implementation

The implementation divided into two parts. The first part is to work on the simulation software, and the second part is to work on the physical hardware components. The simulation software helped us to have an estimation of how the system will work in the physical world. Through the simulation software, we were able to identify some implementation challenges and finding solutions for them. In addition, the simulation software gave us the opportunity to process the sensors data and test the sensors before testing them physically, which speeded up the process of working on the physical components and testing them. Moreover, it was found that the results obtained from the simulated components and the physical components were close to each other. In this section, we demonstrate the progress that happened in both the physical and the virtual worlds.

The Simulation Software

The Mapping algorithm

The Hector SLAM algorithm was selected in this project due to its high efficiency in mapping indoor environments, its ability to work with drones efficiently, and its facility to be integrated with the selected LiDAR sensor. Moreover, it consumes less power in handling some cases where the indoor environment is dynamic, and the obstacles are moving [4]. Hector SLAM is an algorithm that is used widely in mapping unknown indoor environments. The algorithm is LiDAR-based, and it uses the Gaussian Newton equation to construct accurate maps from the laser scanner data [5]. Moreover, this algorithm does not use any odometry data to estimate the robot's position with respect to its surroundings. Instead, the algorithm utilizes the difference in the laser scanner locations to calculate the odometry [6]. This feature qualifies the Hector SLAM algorithm to work optimally with the unmanned aerial vehicles given that in most of the cases, the odometry data is calculated from processing the wheels motion and that is not the case with UAVs. In addition, the algorithm provides an accurate estimation of the robot's position with respect to its surroundings.

The Exploration Algorithm

The method used in this project to explore the indoor sites is selected to be the wall following algorithm due to its effectiveness and simplicity. The implementation of that algorithm can be summarized into three main functions which are: `left_side()` , `move_forward()` , and `take_stop_action()` which are represented in a while loop as following:

> The implementation of the wall following algorithm highly depends on the LiDAR used in the simulator which is Hokuyo laser scanner with 360 rotation angle and 720 readings per 32ms. However, the physical LiDAR used is RPLiDAR A1M8 which provides 360 readings per rotation.

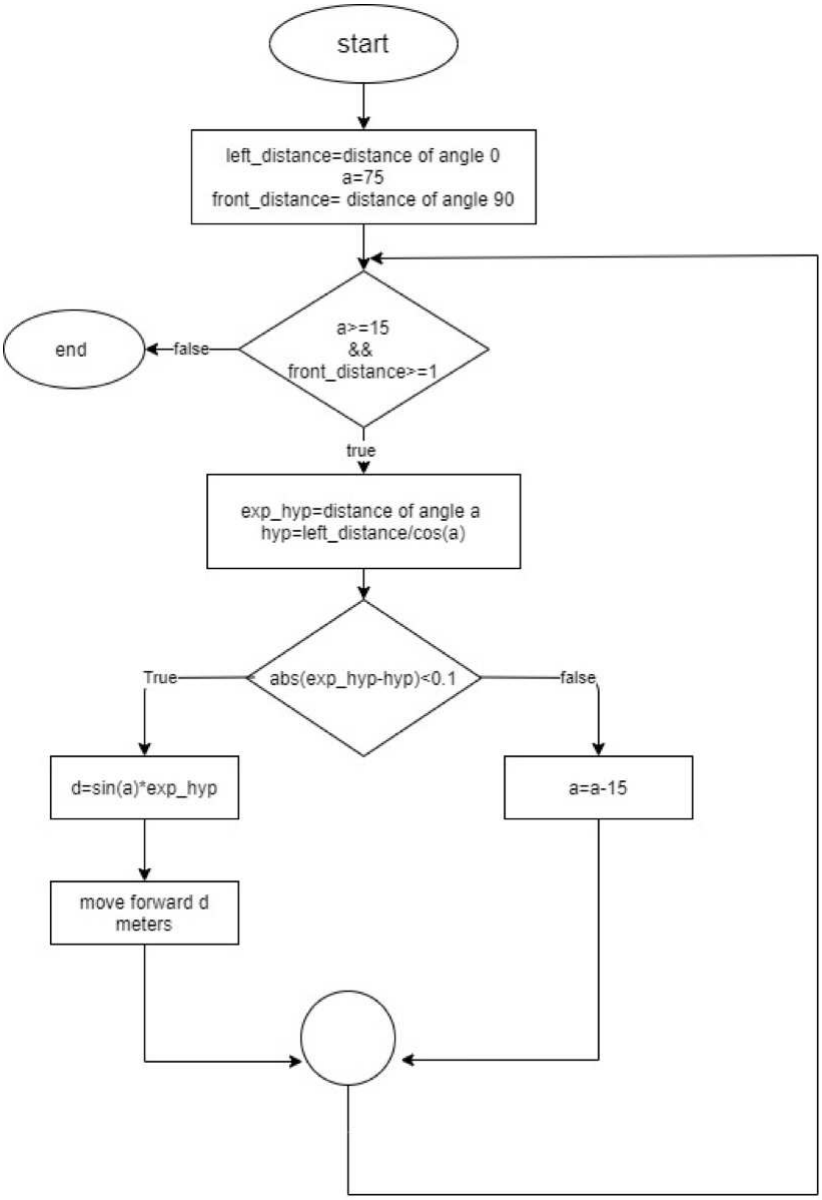
```
while(1):
    left_side()
    move_forward()
    take_stop_action()
```

```
left_side()
```

This function uses the LiDAR readings that are pointing exactly to the west of the drone, it measures how far the drone is from the left wall, then adjust the drone to it such that the drone is approximately 0.7 m away from the left wall. The reason of using 0.7 meters is because the drone has higher error that expected. Therefore, a while loop is used to ensure that the drone is far enough from the wall.

```
move_forward()
```

This function was implemented to safely move the drone forward without hitting a wall, or without skipping an outer corner. The logic behind this algorithm is that it uses the concept of the right angle, and multiple readings which correspond to different angles to measure the safe distances. The bellow flow chat demonstrates the logic of that algorithm in details.



When the function ends, the drone will either stop after an inner corner, or an outer corner.

```
take_stop_action()
```

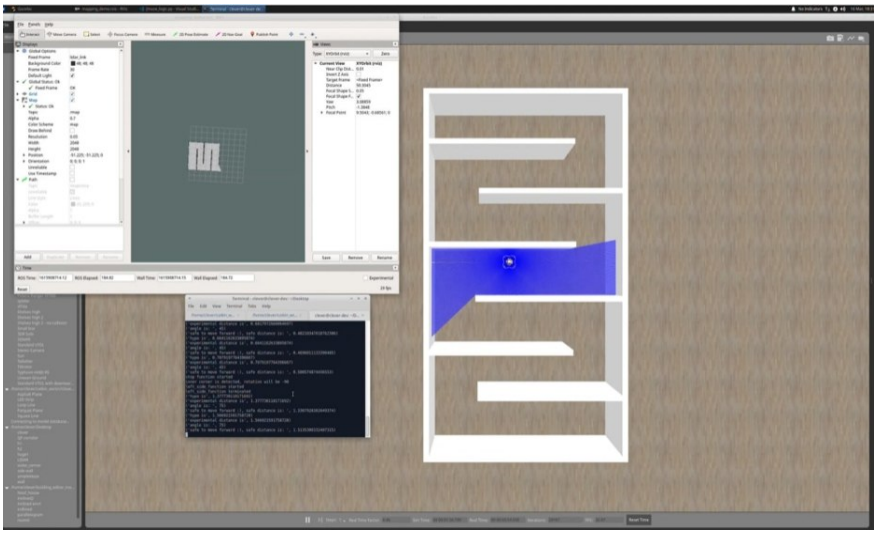
This function handles two different situations:

- The first situation is that the drone may stop when it faces an inner corner, this can be detected by measuring the distance from the front wall, then compare the current distance of the left LiDAR reading with the previously recorded one, if the comparison showed that there is a small difference between these two readings, then this means that the drone must rotate to the right and continue its path.
- The second situation is that the drone may stop when it detects an outer corner, the logic is exactly like the first situation except, that the drone must be away from the front wall (with distance greater than 1.5 meter). In addition, the difference between the current left LiDAR reading, and the previously

recorded reading must be greater than 0.5 m. If this is the case, then the drone has stopped because of an outer corner. Therefore, the drone must rotate to the left and continue its path.

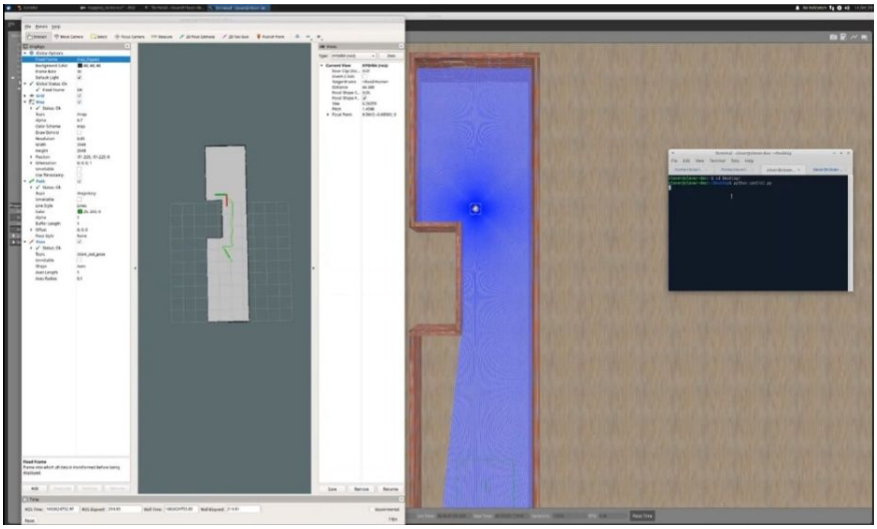
The following video illustrates a ROS Simulation test on Wall Following Algorithm:

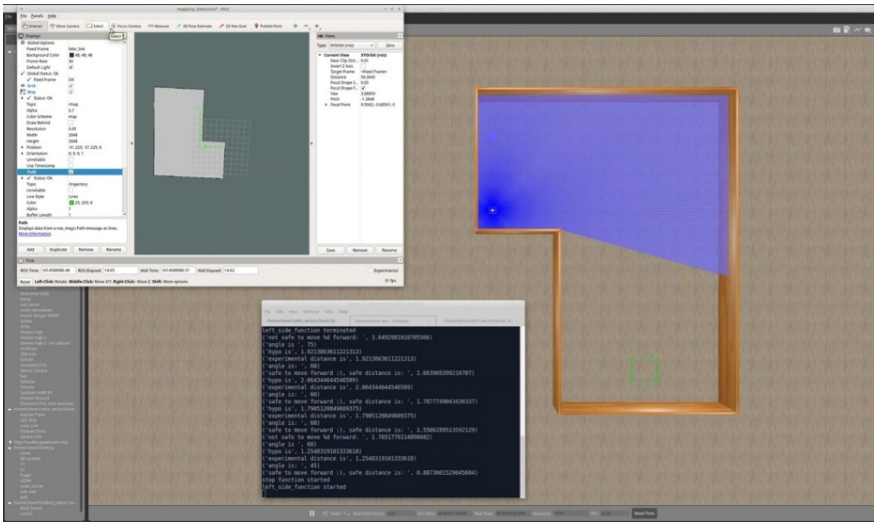
Testing



The above figure demonstrates the drone exploring a maze autonomously while constructing a 2D map in real-time. The terminal shows the safe distances to move forward, these distancing where calculated using the aforementioned flowchart.

The following figures show a constructed 2D map of different environments.





The following video demonstrates an autonomous maze exploration with Hector SLAM responsible for constructing a 2D map:

The physical hardware

This section illustrates the progress done regarding the hardware components. The first step done was to establish a Wi-Fi communication between the Raspberry Pi and the remote PC. The second step was to install the hector SLAM and robot Localization packages in the Raspberry Pi to visualize the maps remotely.

Initial Setup

The drone is assembled and configured correctly to accomplish the autonomous mapping mission. The RPliDAR A1M8 and all other necessary sensors are mounted on the drone as shown in the figure bellow.



To set up the drone ready for mapping, the raspberry pi image created by COEX was installed on the micro-SD card. COEX Raspberry Pi image, COEX pixracer image and COEX virtual machine were selected as they contain all the necessary tools and packages to work efficiently with clover platform. The installed platform is based on Raspbian operating system and ROS. After flashing the image on the SD, the next step is to connect clover to Wi-Fi.

Network Setup

The drone produces a map of an unknown indoor environment by sending data received from the sensors to a remote pc. The transmission takes place over a wireless channel to get a map in real-time. One of the essentials for DroMap is to setup the connection between the drone and the remote PC. In DroMap Project ROS network must satisfy the listed below requirements:

1. There must be a full bidirectional communication between all the nodes.
2. Every component in the network must advertise its name.
3. In ROS network one of the components must be declared as the ROS master. Specifically, the ROS master is the drone (Clover-6064).
4. All ROS packages needed in the project, must use the ROS master.

All these requirements are fulfilled in our design.

Required packages

After the installation of ROS, the drone was ready to install RPLidar ROS package and Hector SLAM. These packages are installed by cloning them in a catkin workspace src folder. Then build them by running catkin build. The following commands were entered in the terminal show the process of installing RPLidar package and hector SLAM in raspberry pi. The `rplidar_ros` package is responsible for retrieving the RPLidar data and hector SLAM package is responsible for building maps. `rplidar_ros` and `hector_slam` packages ware installed from GitHub.

Testing

The testing phase was divided into several stages in order to test the sensor and the SLAM algorithm in several closed places. This makes it possible to identify obstacles and risks that may face us in the future.

We did several of the following elementary tests:

- Firstly, we flew the drone to obtain maps using Hector Mapping with the remote control.
- Secondly, we have moved to the automation stage of implementing the codes applied in the simulator.
- Finally, From here we did some tests, for example, the drone flies to the wall, and then lands after getting a wall reading. And tests are still going on for a fully automatic flight.

References

[1]. “The 20 deadliest jobs in America, ranked,” CBS News. https://www.cbsnews.com/pictures/the-20-deadliest-jobs-in-america-ranked/4/. ↩

2. A. Kovalchenko, “How To Carry Out a Survey and Site Measure,” 2012. https://essenziale-hd.com/2012/10/28/how-to-carry-out-a-survey-and-site-measure/. ↩

3. D. Hähnel, W. Burgard, and S. Thurn, “Learning compact 3D models of indoor and outdoor environments with a mobile robot,” Rob. Auton. Syst., vol. 44, no. 1, pp. 15–27, 2003, doi: 10.1016/S0921-8890(03)00007-1. ↩

4. M. Eliwa, A. Adham, I. Sami, and M. Eldeeb, “A critical comparison between Fast and Hector SLAM algorithms,” / REST J. Emerg. trends Model. Manuf., vol. 3, no. 2, pp. 44–49, 2017, [Online]. Available: www.restpublisher.com/journals/jemm. ↩

5. J. M. Santos, D. Portugal, and R. P. Rocha, “An evaluation of 2D SLAM techniques available in Robot Operating System,” 2013 IEEE Int. Symp. Safety, Secur. Rescue Robot. SSRR 2013, 2013, doi: 10.1109/SSRR.2013.6719348. ↩

6. H. Gossett, “Building an Autonomous Indoor Drone System,” University of Mississippi, 2018. ↩

Seed spreading quadcopter

CopterHack-2021, team **MINIONS**.

Have you ever wondered what a world without trees would look like? Close your eyes, and try to imagine a desolate Earth. There'd be no more paper, and everyone would have to resort to technological use - that is, if anyone was left. Trees are a crucial factor to our existence not only because they produce paper, lumber and chewing gum, but because they serve an important role in the carbon cycle.

Ever since the industrial revolution between 1760 and 1840, the world has been in a never-ending carbon chaos. Trees and Plankton are our only saviours in terms of handling this problem, and we can only control one of them, trees.

We need to save trees by protecting them from the destructive human activities like clearance of forests, deforestation for urbanization, etc. Trees are the lungs for the earth. It is an important part of nature’s ecosystem. They balance the soil composition and also act as the barrier for wind and storm. Thus, they provide various uses to the ecosystem. For these reasons, it’s imperative that we save trees.

Since there are a lot of dangerous and difficult-to-reach landsides for humans to plant, the most viable alternative is to use drones for plantation in those regions.

Seed-firing drones will, as the name suggests, fire seeds into fertile soil to allow millions of trees to grow back after being cut down for industrial use. If the rate of planting exceeds the rate of cutting, eventually we will restore the trees we once felled.



Our Aim

We will make drones able to hold seeds onboard and drop them in an area which we drove in a special application. We can control the density of the seeds and the height of the drop. We also thought about protection of the seeds from insects, animals and dehydration. We choose the earth ball technique invented by

Masanobu Fukuoka, aka Fukuoka Technique. This earth ball contains all needed elements to grow, plant seeds and earth for protection. When we drop it on the ground, the earth ball will hold seeds until it gets the needed amount of water and seeds will begin to grow.

YouTube video link - https://www.youtube.com/embed/Nz1w59v451U.

We achieved to do small seeding missions but we faced some problem about autonomous flying with GPS.

We coated our battery to protect it from cold weather, seeding missions need to start in winter since apple seeds need to stay in a cold place for some time to break dormancy.

- Seed capsules
- How to assemble seeding mechanism to clover 4.2 drone
- How to control the seeding mechanism
- Programming

Files

Link for the all files used in this project:
https://github.com/Sahinysf/TreeSeedQuad.

Seed capsules

Fukuoka technique

In southern Japan, the Japanese farmer and philosopher Masanobu Fukuoka invented a seed ball planting technique. The method is regarded as a natural farming technique that requires no machines, no chemicals and very little weeding. By the use of seed balls, land is cultivated without any soil preparation.



Advantages of seed balls:

- It is simple and easier to make seed balls without machines.
- Easier for reforestation and plantation in difficult terrains.
- Contribute to protect soil, environment and livelihood.
- It is an organic technique and doesn't use any chemicals.
- It is a low-cost method compared to traditional afforestation/reforestation techniques.
- It requires low maintenance.

Which Seeds can be used?

Any seed which grows in your area (In our it's apple seed).

Size and weight of the seed capsule: size and Weight of seed capsules are very important for this project. After some experiments we decided that best size is 16-18mm diameter and maximum weight is 10 g.

Required materials for making seed balls:

1. 1 bucket of clay
2. 1 bucket of organic dark soil / compost
3. 1 bucket of water (amount of water may vary depending on the soil type)
4. ¼ bucket of seeds

Steps for making seed balls:

1. Collect same quantity of both clay and organic soil. For example, if you use one bucket of clay, then you should mix with one bucket of organic soil.

2. Make sure that clay and organic soil fine particles.
3. The clay and organic soil texture should be wet but not sticky
4. Take a bit of mixture and roll it into balls. Test the ball by throwing it on a flat surface. If the ball doesn't break easily, it means it has got good bonding.
5. Seed balls must be a perfectly round shape otherwise they will be stuck while throwing with the quadcopter
6. Insert seeds (1 to 2 seeds per seed ball for permanent trees such as mahogany, sandalwood, orange, moringa…) (± 5 seeds per seed ball for vegetables, flowers, grasses, clovers…)
7. Dry the seed balls for one to two days in a shaded area, if properly dry, the seed balls will be protected from external predators such as chickens, birds, rats…



Second technique - paper seed capsules.

This method was influenced by a Korean newspaper that contained seed that could be planted outside after reading it. Paper seed capsules :

Required materials:

1. Any kind of paper
2. Water
3. Blender
4. Seeds

Steps for making paper seed balls:

1. Shred all of your paper.
2. Put paper in blender and add water, after 2 minutes blend.
3. Squeeze all the water out with paper.
4. Add seeds and give round shape.
5. Let it dry overnight.



Advantages of paper balls:

- Easy to find materials.
- Environmentally friendly.

REREFENCES

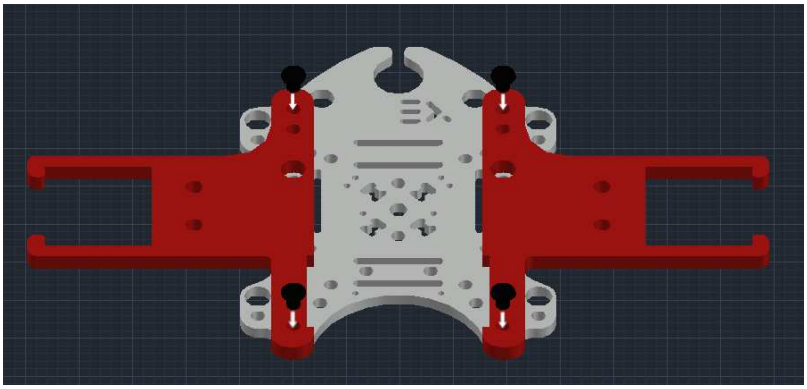
https://web.archive.org/web/20090115211020/http://www.rmaf.org.ph/Awardees/Biography/BiographyFukuokaMas.htm
http://www.guerrillagardening.org/ggseedbombs.html

How to assemble seeding mechanism to Clover 4.2 drone

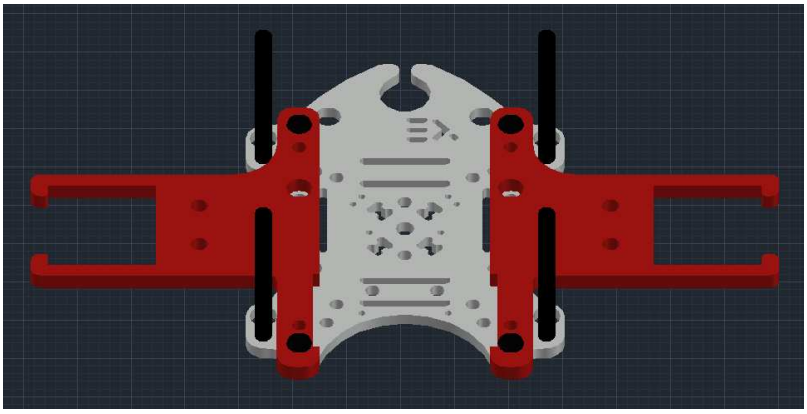
How to assemble seeding mechanism

After finishing step 4, at section Installing guard of Clover 4.2 assembly.

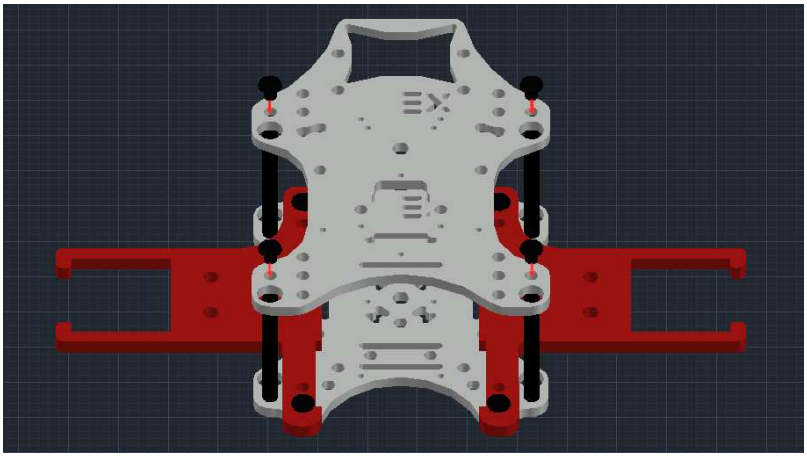
1. Install the Lower Tank Holders to top Deck mount and fix with the M3x8 screws.



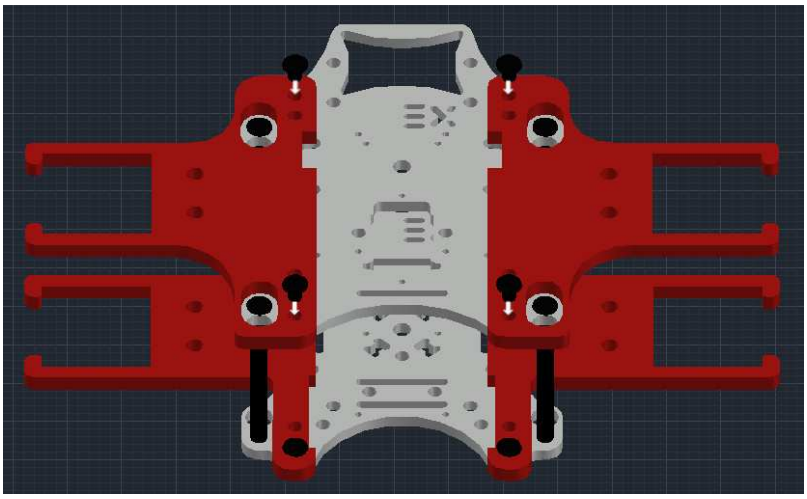
2. Install Nylon rack(40 mm) to 4 sides of the Deck mount.



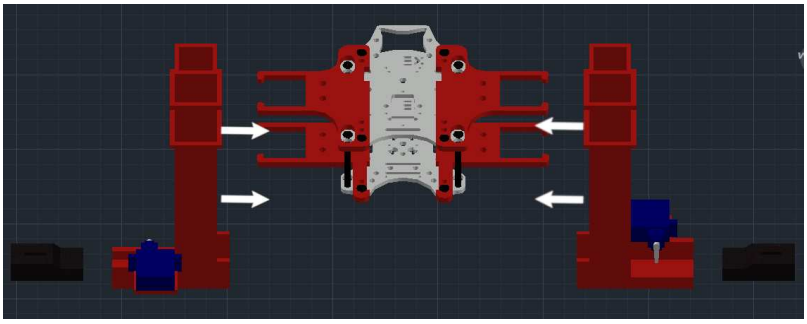
3. Install the Grab deck and fix with the M3x8 screws.



4. Install the Upper Tank Holders to top Grab mount and fix with the M3x8 screws.

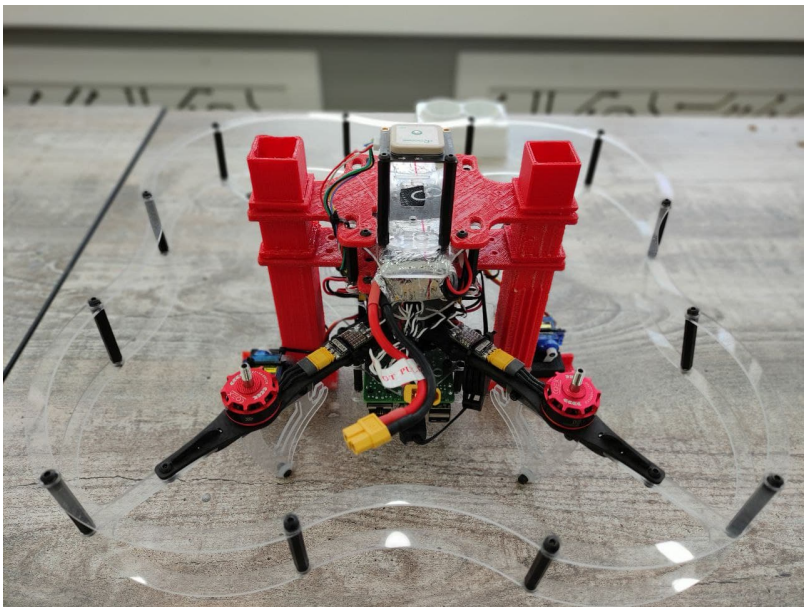


5. Connect the Tanks carefully to Tank Holders.



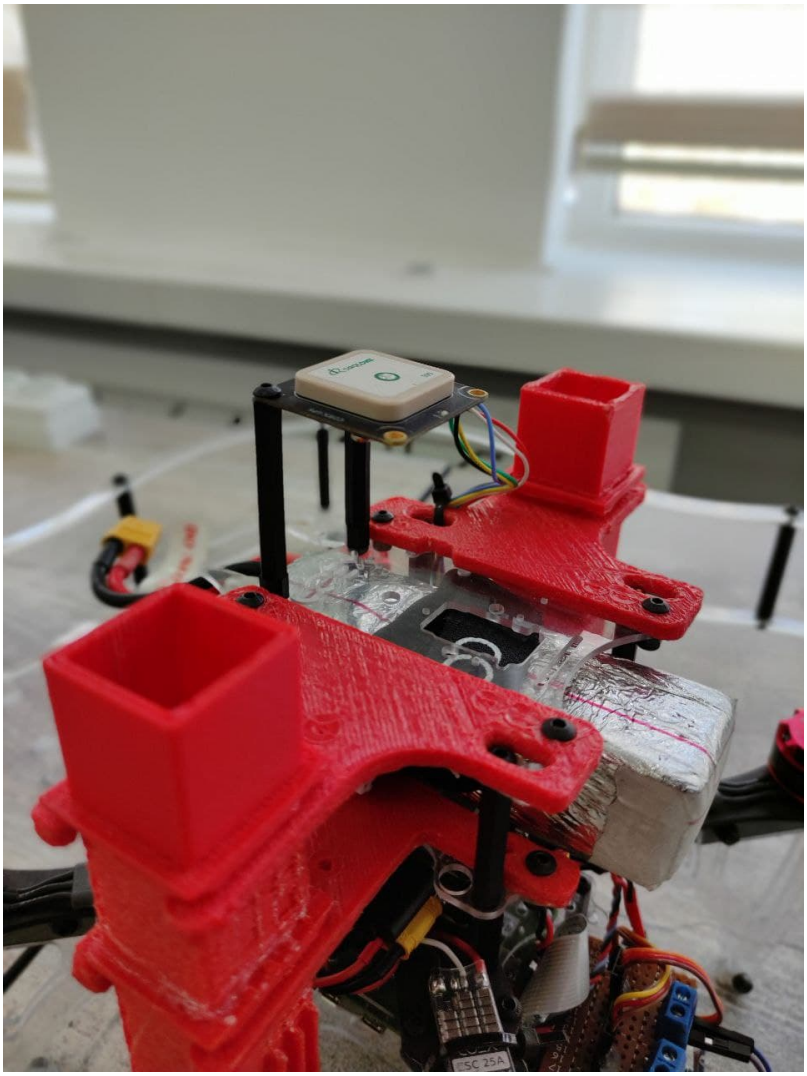
6. Connect SG90 servo motors to Tank using zip tie.

Final view of seeding drone:



GPS Module

We installed the GPS Module to the top using 2 Nylon rack (40 mm):



We coated the battery to protect it from the cold weather:

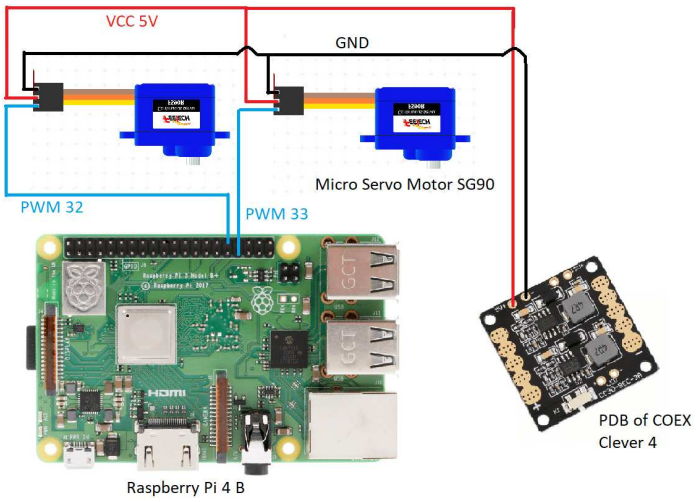


How to control the seeding mechanism

Electronic part of seed dropping mechanism consists of:

- Raspberry Pi 4 B of COEX Clover 4.
- 2 Micro Servo Motors SG90.
- PDB (Power Distribution Board) of COEX Clover 4.

Servo motor’s signal pins are connected to Raspberry Pi’s Hardware PWM pins 32 and 33, and power is taken from Power Distribution Board (5 V).



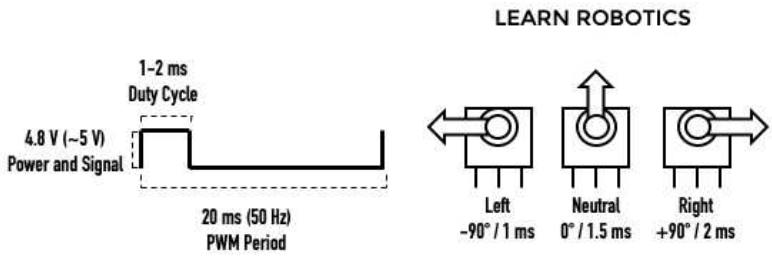
Explanation of code for controlling servo motors

Servo motors are controlled using a PWM (Pulse-Width Modulation) signal from Raspberry Pi. PWM controls the amount of time when signal is HIGH or LOW within a certain period of time. Duty Cycle – percentage of time when signal is HIGH.

In a table below it's presented the duty cycle of Servo Motor SG90 of each angle of servo motor. In order to use duty cycle in code we need to convert time to percentage by dividing duty cycle time by the total PWM period.

What we get is:

- -90° rotation angle or 2ms Duty Cycle => 1/20*100% = 5% Duty Cycle.
- 90° rotation angle or 2ms Duty Cycle => 2/20*100% = 10% Duty Cycle.
- 0° rotation angle or 1,5ms Duty Cycle => 1,5/20*100% = 7,5% Duty Cycle.



We'll do this by using the RPi.GPIO library and writing Python code on the Raspberry Pi.

First, import the RPi.GPIO library and the sleep function:

```
import RPi.GPIO as GPIO
from time import sleep
```

Then, setup the GPIO mode as BOARD:

```
servo = 33
GPIO.setmode(GPIO.BOARD)
GPIO.setup(servo, GPIO.OUT)
```

Next, create a variable for the servo, PWM. Then, send a 50 Hz PWM signal on that GPIO pin using the `GPIO.PWM` function. Start the signal at 0:

```
pwm=GPIO.PWM(servo, 50)
pwm.start(0)
```

Use the `ChangeDutyCycle` function to write duty cycle percentages to the servo motor.

```
pwm.ChangeDutyCycle(5) # left -90 deg position
sleep(1)
pwm.ChangeDutyCycle(7.5) # neutral position
sleep(1)
pwm.ChangeDutyCycle(10) # right +90 deg position
sleep(1)
```

Programming

In order for the mission to be achievable in the best way and within our reach, we were required to utilize the threading in Python.

Simple mission code:

```
import threading
import time
import rospy
from clover import srv
from std_srvs.srv import Trigger
import RPi.GPIO as GPIO

rospy.init_node('flight')

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)
navigate = rospy.ServiceProxy('navigate', srv.Navigate)
navigate_global = rospy.ServiceProxy('navigate_global', srv.NavigateGlobal)
set_position = rospy.ServiceProxy('set_position', srv.SetPosition)
set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)
set_attitude = rospy.ServiceProxy('set_attitude', srv.SetAttitude)
set_rates = rospy.ServiceProxy('set_rates', srv.SetRates)
land = rospy.ServiceProxy('land', Trigger)

servo1 = 33        # PWM pins
servo2 = 32

GPIO.setmode(GPIO.BOARD)    #set pin numbering system

GPIO.setup(servo1,GPIO.OUT)
GPIO.setup(servo2,GPIO.OUT)

pwm1 = GPIO.PWM(servo1,50)    #create PWM instance with frequency
pwm2 = GPIO.PWM(servo2,50)

pwm1.start(0)        #start PWM of required Duty Cycle
pwm2.start(0)


def servo_drop(seconds):   #function to drop seed capsules from 2 tanks
    print("Dropping")

    i = 1                          #variable to choose which tank
    for num in range(seconds/2):
        if(i == 1):                #first tank
            pwm1.ChangeDutyCycle(10) # release one seed capsule
            time.sleep(0.5)
            pwm1.ChangeDutyCycle(5) # push then drop the capsule
            time.sleep(0.5)
            i = 2                  #changing the variable for to use the secor

        elif(i == 2):              #first tank
            pwm2.ChangeDutyCycle(10) # release one seed capsule
            time.sleep(0.5)
            pwm2.ChangeDutyCycle(5) # push then drop the capsule
            time.sleep(0.5)
            i = 1                  #changing the variable for to use the first

        print(num)
        time.sleep(2)


if name == "main":
    # Take off and drone 10m above the ground
    navigate(x=0, y=0, z=10, frame_id='body', auto_arm=True)

    # rospy waits for 10 seconds to take off
    rospy.sleep(10)

    # Dropping starts simultaneously with flying forwards 5 meters
    d = threading.Thread(target=servo_drop, args=(18,))  # 18 is the sum of al
```

```
    d.start()

    navigate(x=5, y=0, z=0, frame_id='body')

    #rospy waits for 8 seconds to fly forward
    rospy.sleep(8)

    # Fly right 1 m
    navigate(x=0, y=1, z=0, frame_id='body')

    #rospy waits for 2 seconds to fly right
    rospy.sleep(2)

    # Fly backward 5 m
    navigate(x=-5, y=0, z=0, frame_id='body')

    #rospy waits for 8 seconds to fly backward
    rospy.sleep(8)

    # Perform landing
    land()

pwm1.stop()
pwm2.stop()
GPIO.cleanup()
```

References

- https://www.nationalgeographic.com/environment/article/deforestation
- http://www.fao.org/fileadmin/templates/rap/files/NRE/Forestry_Group/Landslide_PolicyBrief.pdf
- https://earthenginepartners.appspot.com/

Developed by Team MINIONS

Special thanks to International Ala-Too University for funding the Clover 4 kits.



ALA-TOO INTERNATIONAL UNIVERSITY

Designing a drone and a path planning algorithm

CopterHack-2021, team: **Blue Jay Eindhoven**.

Introduction

We at Blue Jay Eindhoven are a student team of the Eindhoven University of Technology. We are doing research on drones that fly indoors, are interactive, autonomous, safe and helpful.

We are participating in the Copterhack 2021, because COEX has a lot of knowledge about making a drone. With the help of COEX's expertise, we would be able to develop our drone further. However, the project with which we started the Copterhack turned out to be not that successful. We therefore also didn't get to have a more in depth discussion with COEX. The fact that we are not an open source company, added to this. We couldn't just share everything and when that one project failed, we first had to look at what we are going to share next. Because of these events, the collaboration part hasn't really lifted off.

The information that you are going to find in this project summary is therefore not that specific developed for the Clover drone. It can however certainly be used to customize the Clover drone. In addition, it can also give a first overview for beginners on how to design a path planning algorithm and how the design process of a drone looks like.

We are still planning on doing some research on the Clover drone itself. This will mainly be on the stability and movability of the our drone. But we are going to start with investigating it on the Clover drone. So there might be a nice further collaboration on this. In addition, we are also trying to implement the codes for the Clover drone onto our drones. By doing this, we will also be able to provide some feedback on it and develop it further.

Now we will give you a short summary of our results on the Path Planning Algorithm and the Hardware Research that we have done. For the full reports, you can go to this google drive: https://drive.google.com/drive/folders/1vfWjWD5Qx38mDta0PvMFvAv6jC-mxF7U?usp=sharing.

Path Planning Algorithm

It is investigated what the most optimal path planning algorithm is for the Clover drone. This is done since it was noted that this is not done in the base version of the drone. The path planning algorithm makes it possible that the drone flies autonomous in a much better way than without the algorithm.

In the documentation, we have set up a plan to put this path planning algorithm to work on your drone. This is a low level algorithm, so everyone should be able to implement it. The algorithm does however need some sort of map from which it can get information on the possible paths. So that part, you still have to implement yourselves.

Hardware Research

The report describes the internship project carried out at Blue Jay. As we focus on indoor drone application, we wishes to minimize the produced noise to improve user experiences. On the other hand, we also wish to improve the flying efficiency for benefiting flying time. As result, the project is about making the drone more efficient with less noise emitted during the operation. For producing design to approach the problem, design methodology has been applied. In the end the ducted fan design has been chosen through studied theory and experimenting. However, there are two additional requirements, an increase in amounts of sensors and an increase in propeller numbers to increase safety. These additional requirements result the drone has less flying time due to the increase in weight. However, the selected design still improved the efficiency of selected propeller.

This hardware part is more of a general research that can be applied to all drones, including the Clover drone. If a group of high school or university students would like to do their own research on the Clover drone, they can use the research for ideas. They can for instance perform a project in which they design their own 3D printed ducted fans to use on the Clover drone. The research in the report would then be a good first read on how to design such a thing and what the performance results could be.

In addition, the kind of propellers that are used on the Clover drone can also be adjusted. Maybe a group wants get a little smaller or larger propellers or even a different shape. The research done can then also help as a guide to decide what kind would be best for that specific use with respect to size and shape.

It is also useful when one wants to use the code that the Clover drone uses, but also wants to develop their own drone. The hardware research can then be used as a guide on how you can do this. It states all kind of factors that should be taken into account and in what way you can do this.

Full Project Information

To see what our project at Blue Jay is all about, you can watch the following video of our interim event: https://www.youtube.com/watch?v=E_8TTQN92pU&t=0s. We state our user case, explain what we have achieved so far and what the plans are for the future.

If you have questions or ideas, feel free to ask! You can contact us at info@bluejayeindhoven.nl.