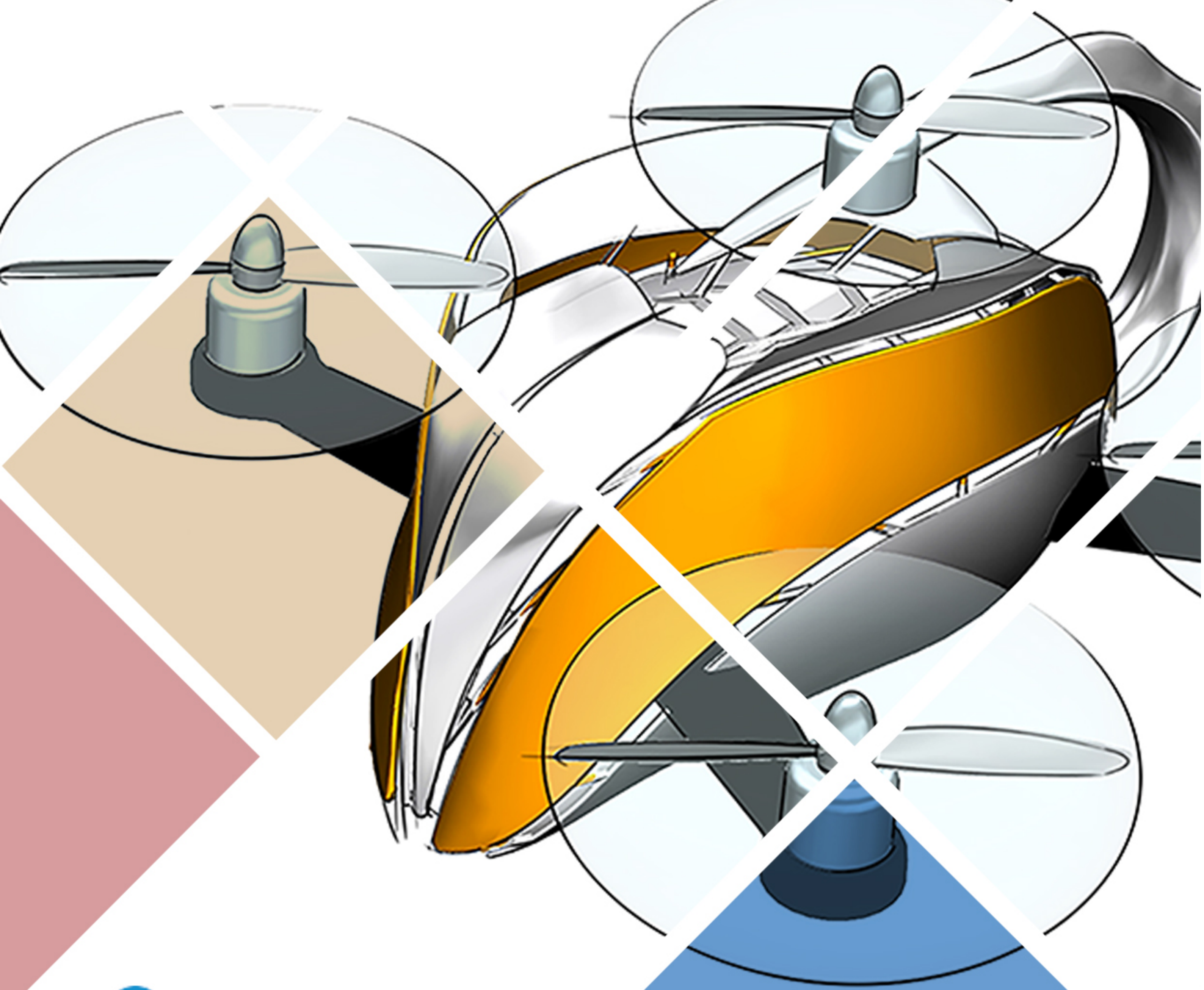


DRONE TEAM

Drone It Yourself!

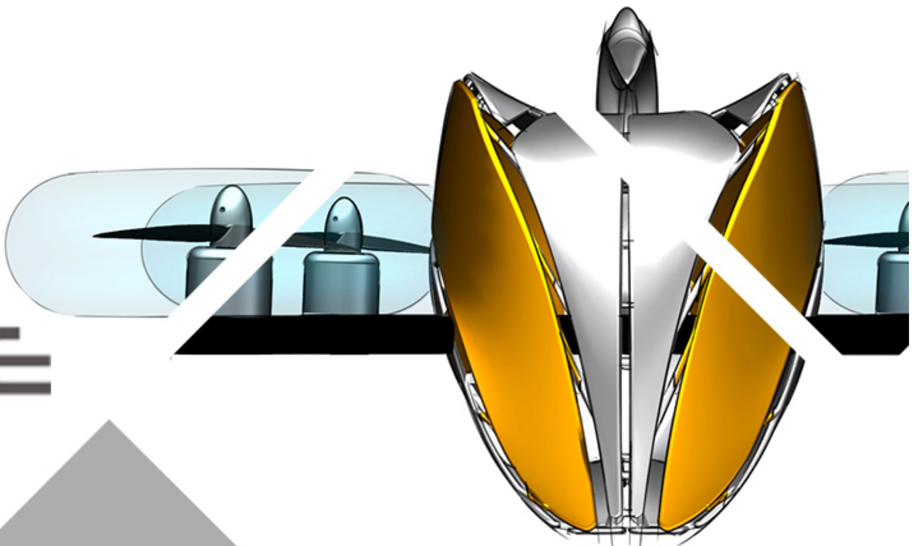
MAKING AND DESIGNING A TOY DRONE THROUGH MULTIDISCIPLINARY COLLABORATIVE WORK

Project no. 2015-1-ES01-KA202-015925

Co-funded by the Erasmus+ Programme of the European Union

Drone It Yourself! consists of the following modules:

0. INTRODUCTION TO THE DRONETEAM PROJECT
1. BASIC TOY DRONE FRAME
2. MODULE OF FLIGHT CONTROL
3. MODULE OF COMMUNICATION CONTROL
4. MODULE OF ADVANCED FRAME
5. MODULE OF GPS-COMPASS CONTROL
6. MODULE OF PROBLEM MANAGEMENT
7. MODULE OF FLIGHT STABILIZATION SYSTEM
8. MODULE OF FIRST PERSON VIEW
9. DRONETEAM E-LEARNING PLATFORM
10. OTHER DEVELOPMENTS
11. GLOSSARY



This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

MODULE OF COMMUNICATION CONTROL

2015-1-ES01-KA202-015925

Co-funded by the Erasmus+ Programme of the European Union

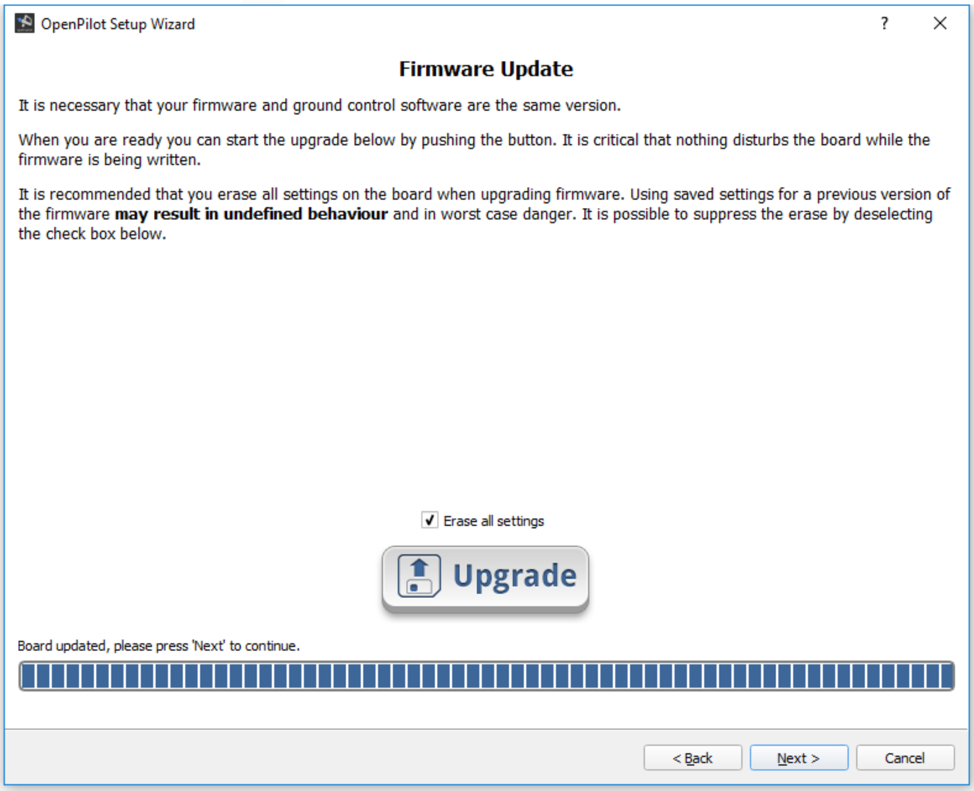
Index

1. LIBREPILOT- SOFTWARE CONTROL FOR BASIC DRONE WITH CC3D CONTROLLER

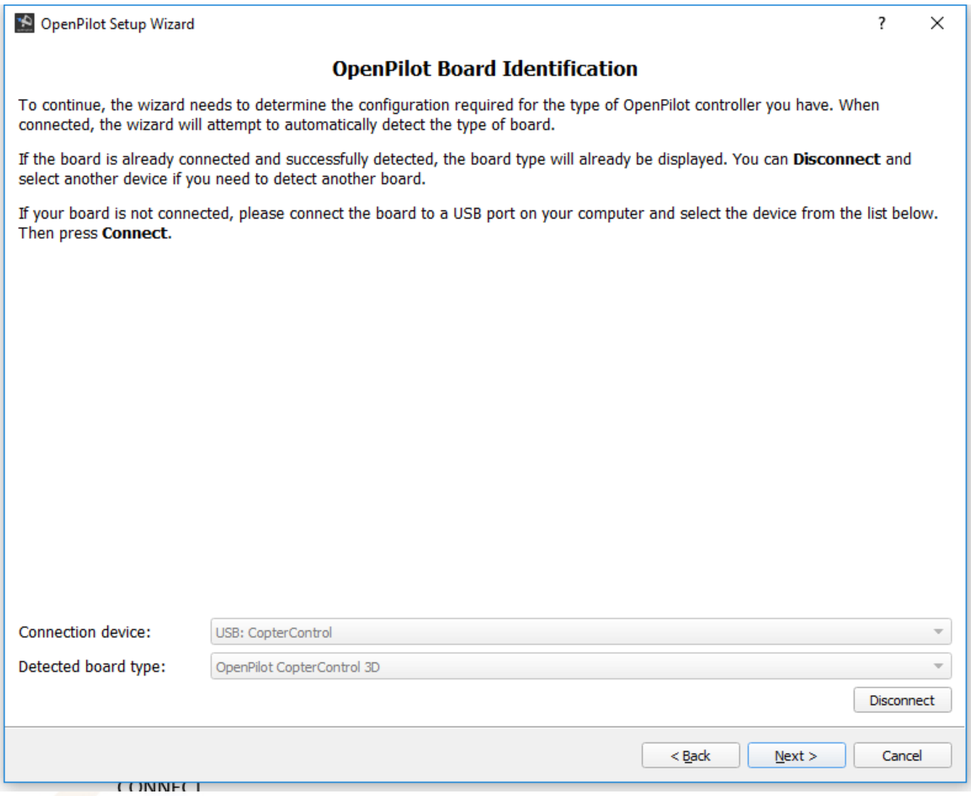
LibrePilot is an open source software to drone control. LibrePilot project start in July 2015. In DroneTeam Project we used for basic drone, for drone control and stabilization.

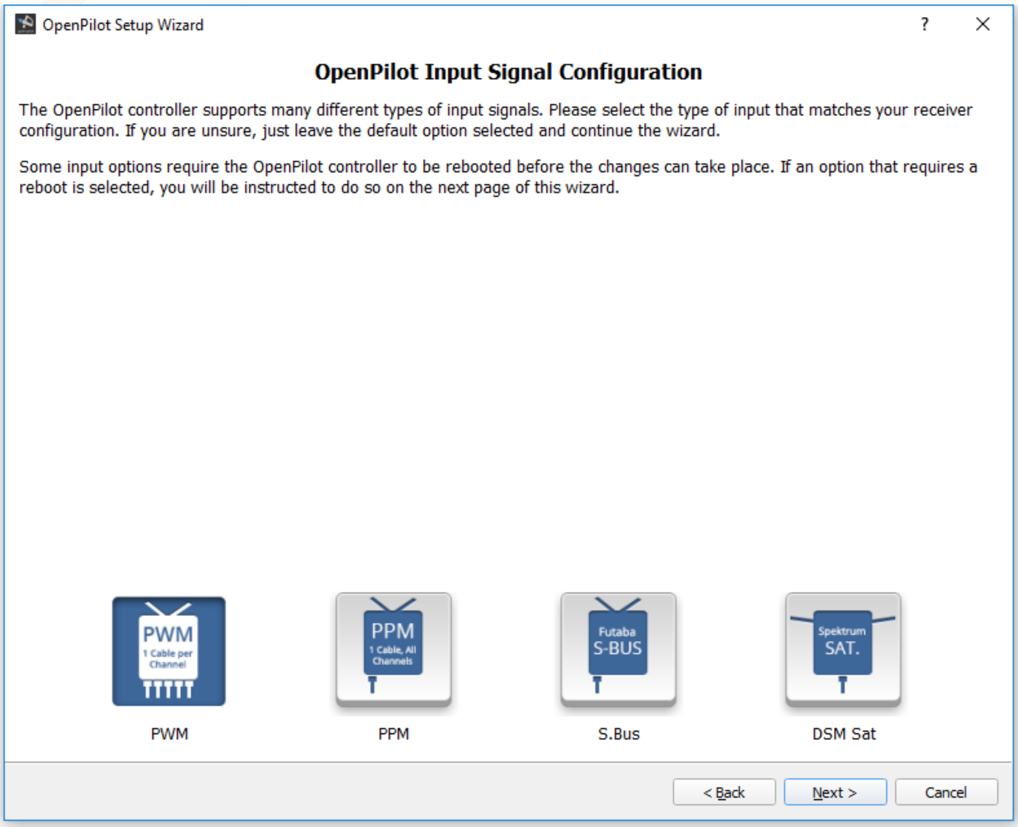
To know more about LibrePilot visit the official webpage (https://www.librepilot.org) and the source code is available in Bitbucket (https://bitbucket.org/librepilot/) and Github (https://bitbucket.org/librepilot/).

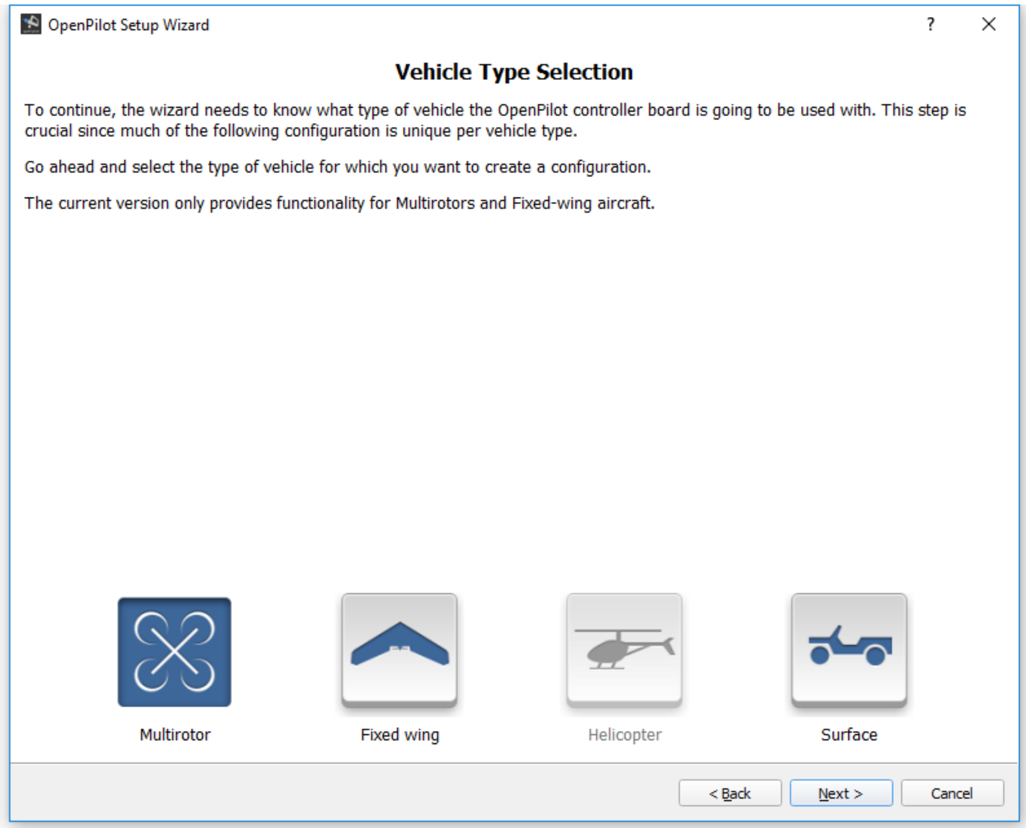
Following steps are collected to show how to configure LibrePilot in DroneTeam basic drone proposed:

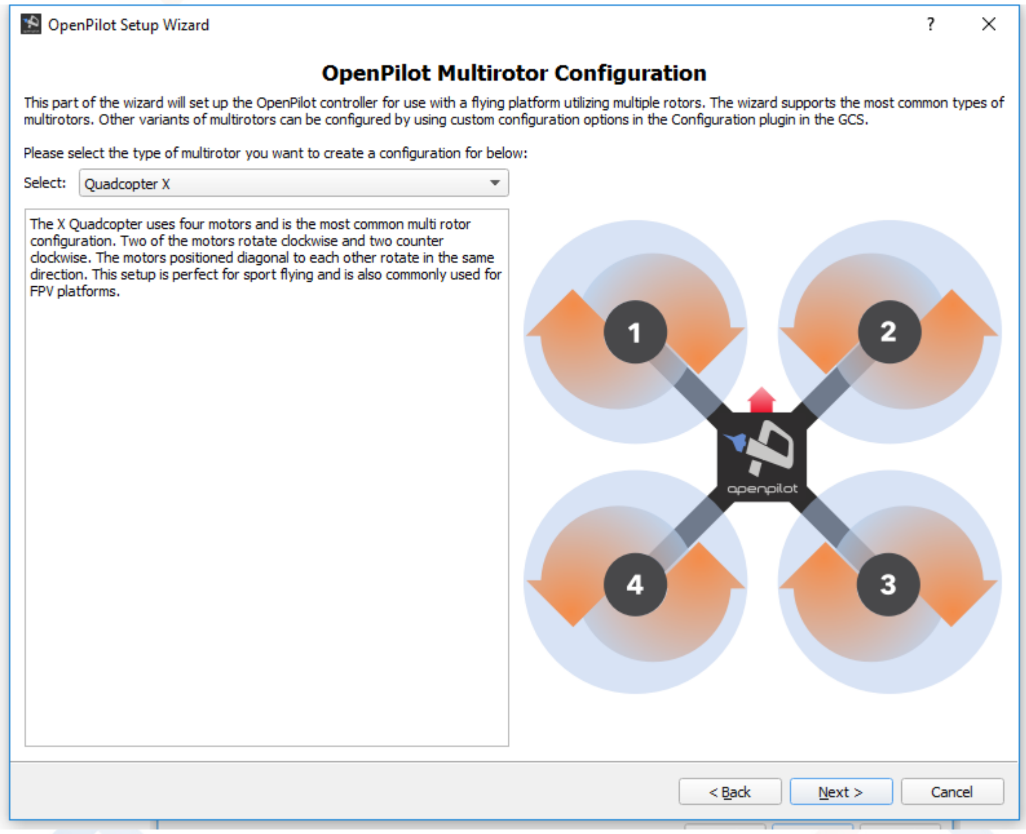


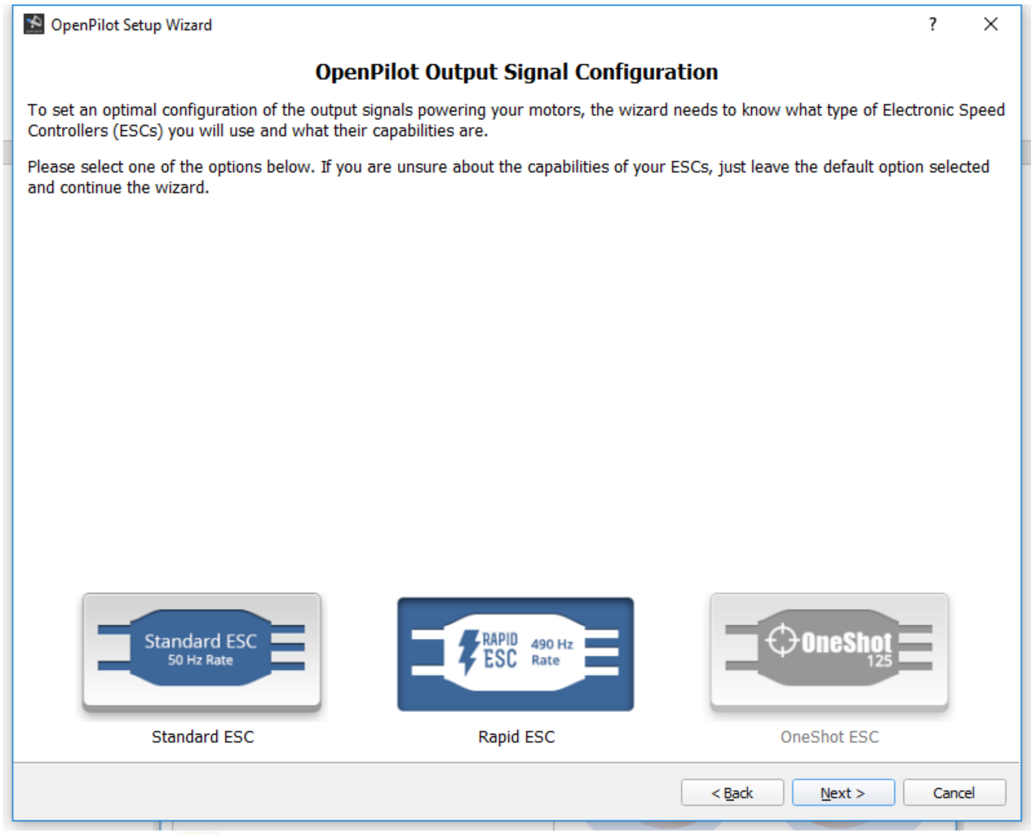
1.1. CONNECT MINIUSB FROM CC3D TO PC WITHOUT BATTERY CONNECTION ON CC3D

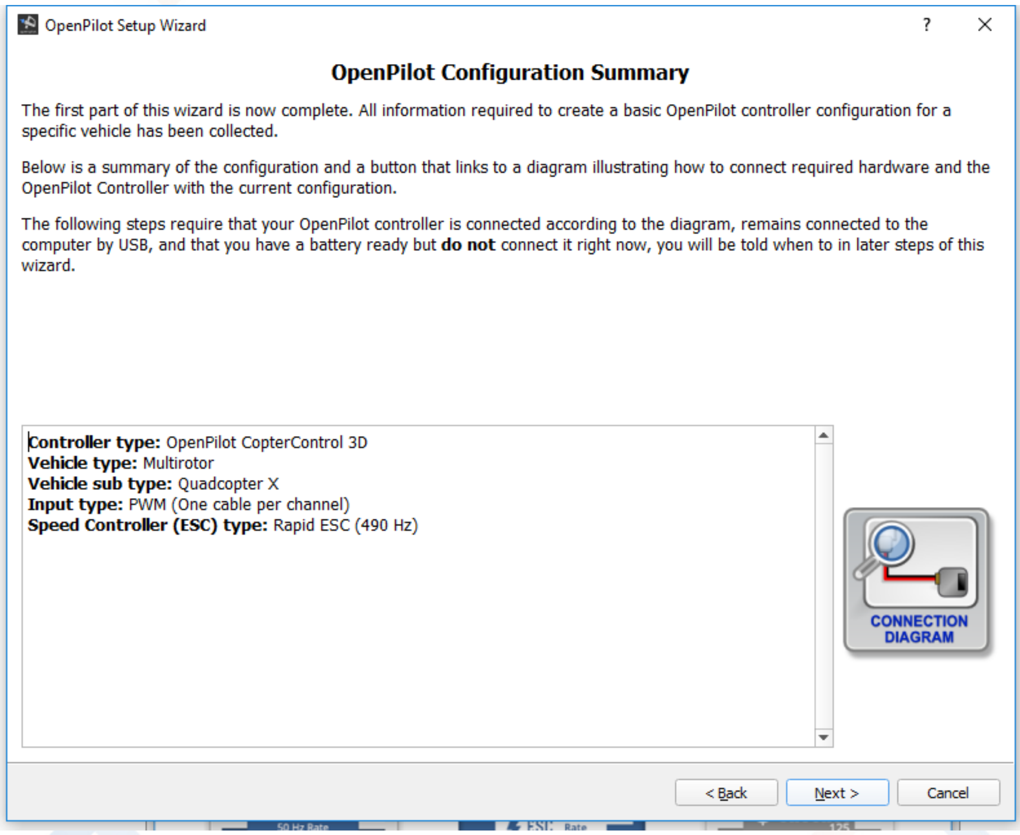


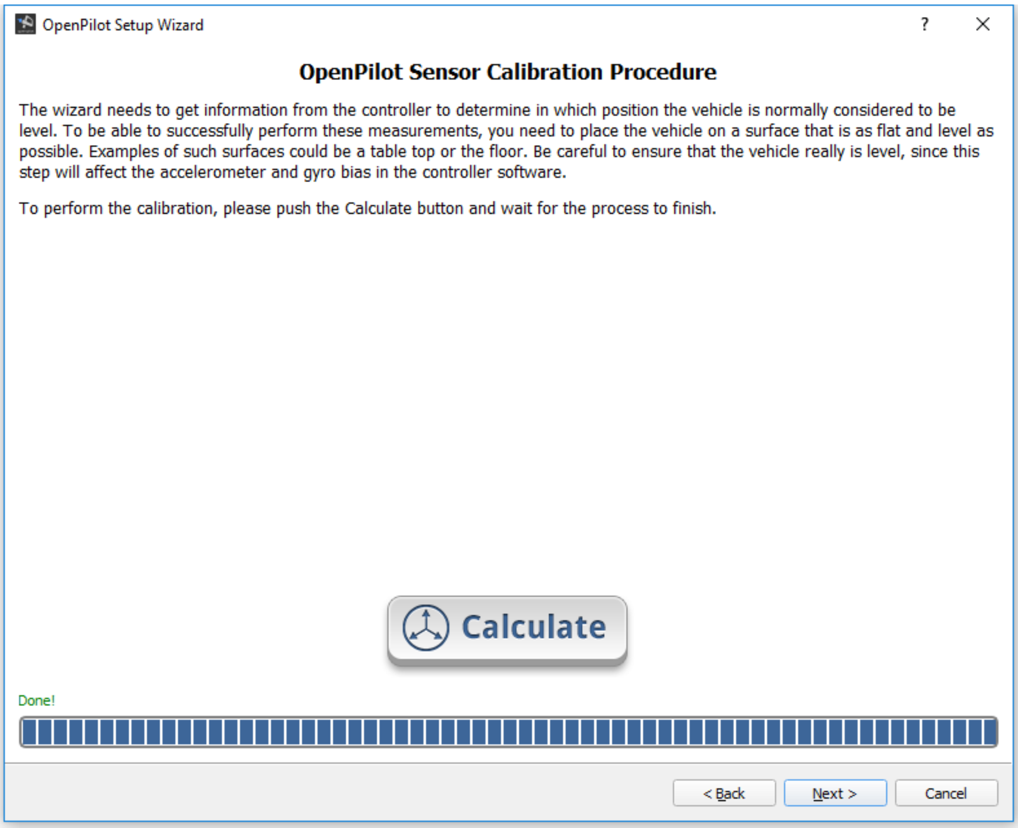


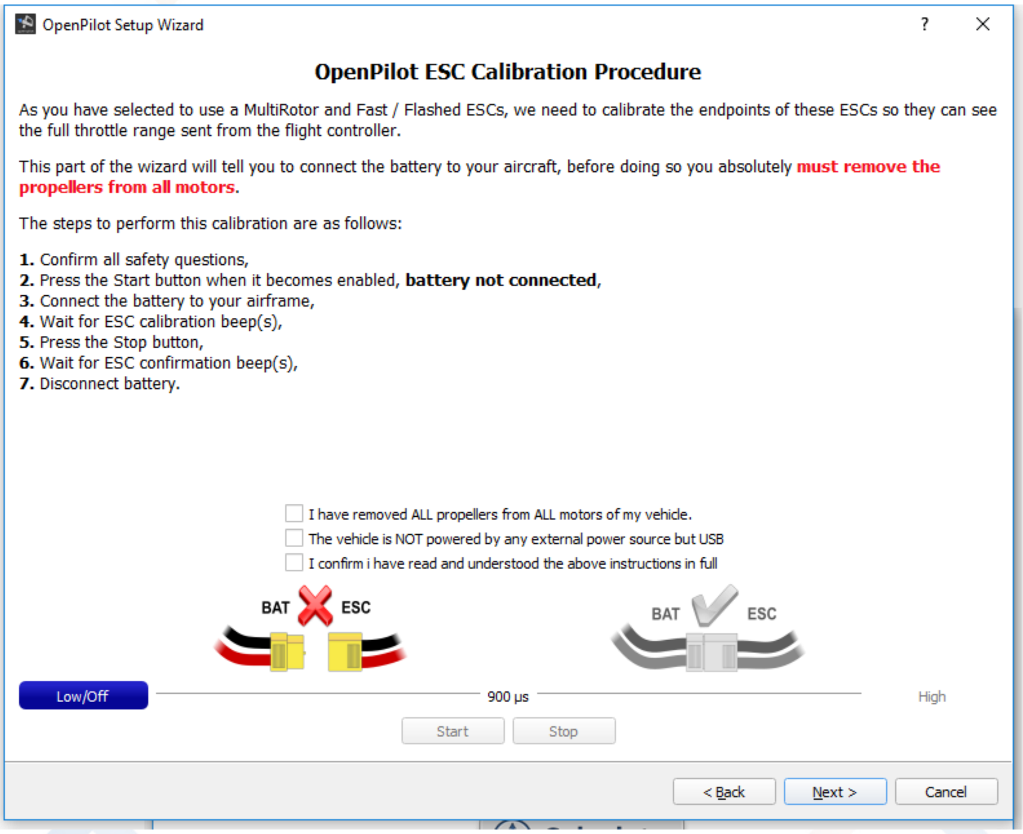

OpenPilot Setup Wizard
Vehicle Type Selection
To continue, the wizard needs to know what type of vehicle the OpenPilot controller board is going to be used with. This step is crucial since much of the following configuration is unique per vehicle type.
Go ahead and select the type of vehicle for which you want to create a configuration.
The current version only provides functionality for Multirotors and Fixed-wing aircraft.
Multirotor
Fixed wing
Helicopter
Surface
< Back
Next >
Cancel

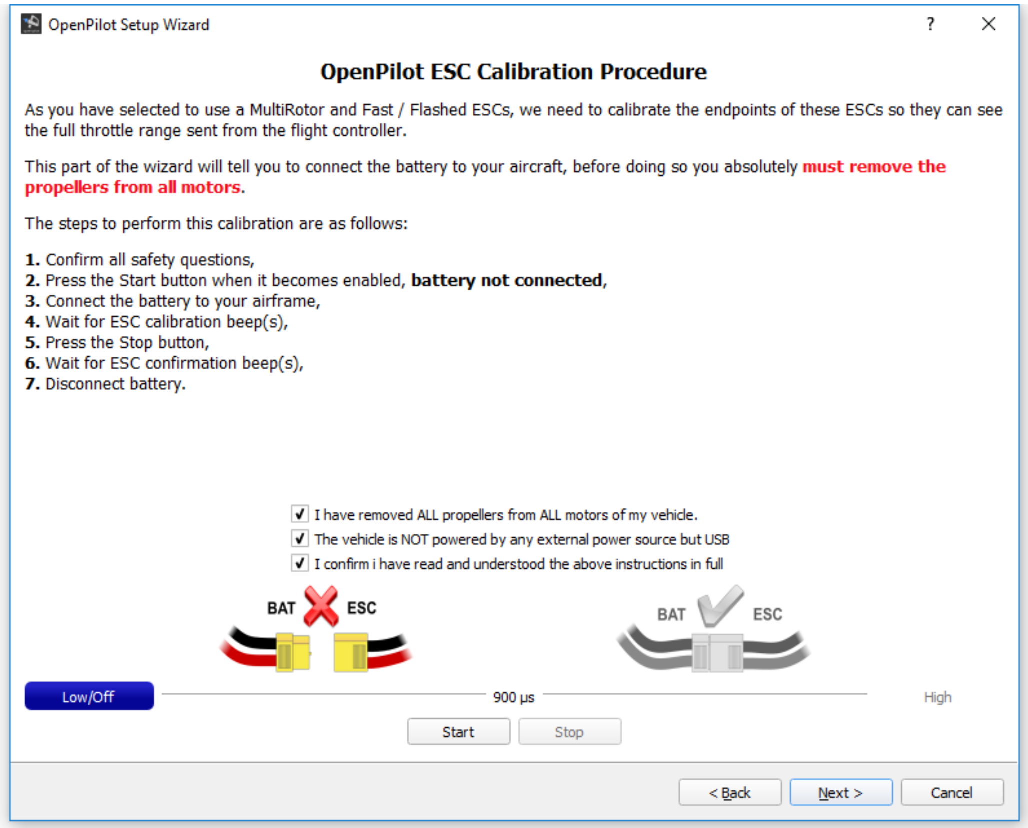

OpenPilot Setup Wizard
OpenPilot Multirotor Configuration
This part of the wizard will set up the OpenPilot controller for use with a flying platform utilizing multiple rotors. The wizard supports the most common types of multirotors. Other variants of multirotors can be configured by using custom configuration options in the Configuration plugin in the GCS.
Please select the type of multirotor you want to create a configuration for below:
Select: Quadcopter X
The X Quadcopter uses four motors and is the most common multi rotor configuration. Two of the motors rotate clockwise and two counter clockwise. The motors positioned diagonal to each other rotate in the same direction. This setup is perfect for sport flying and is also commonly used for FPV platforms.
1
2
3
4
openpilot
< Back
Next >
Cancel


OpenPilot Setup Wizard
?
OpenPilot Output Signal Configuration
To set an optimal configuration of the output signals powering your motors, the wizard needs to know what type of Electronic Speed Controllers (ESCs) you will use and what their capabilities are.
Please select one of the options below. If you are unsure about the capabilities of your ESCs, just leave the default option selected and continue the wizard.
Standard ESC
50 Hz Rate
RAPID ESC
490 Hz Rate
OneShot
125
Standard ESC
Rapid ESC
OneShot ESC
< Back
Next >
Cancel

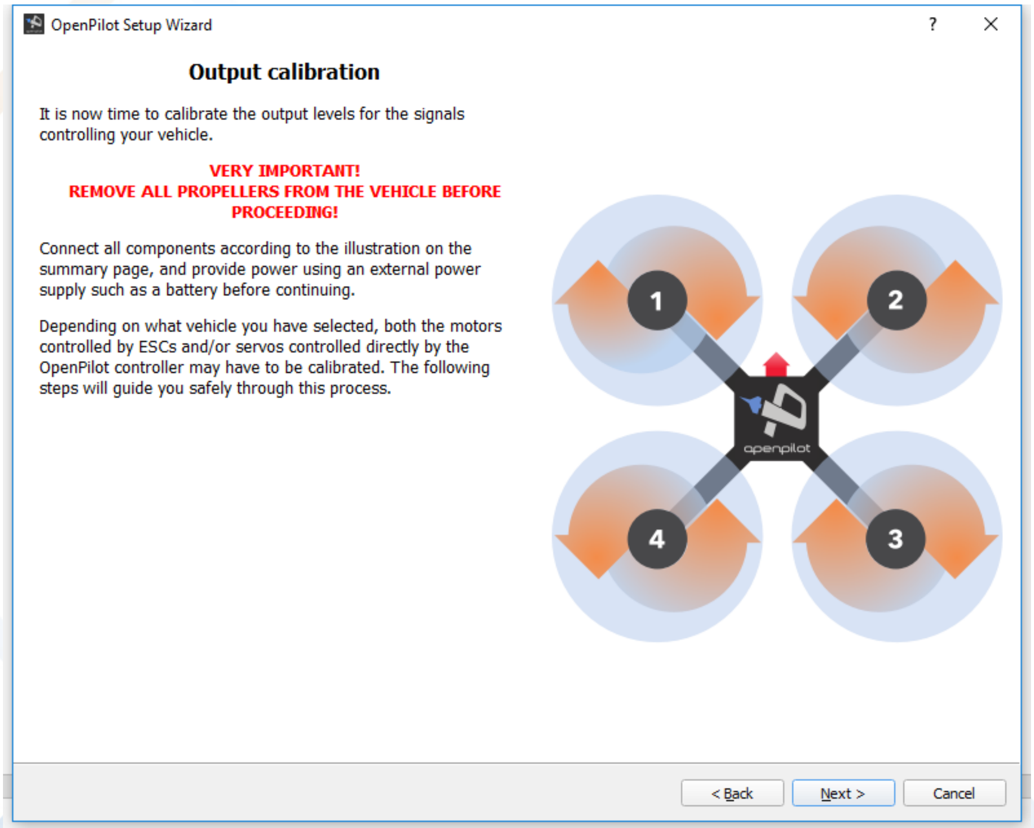

OpenPilot Setup Wizard
?
OpenPilot Configuration Summary
The first part of this wizard is now complete. All information required to create a basic OpenPilot controller configuration for a specific vehicle has been collected.
Below is a summary of the configuration and a button that links to a diagram illustrating how to connect required hardware and the OpenPilot Controller with the current configuration.
The following steps require that your OpenPilot controller is connected according to the diagram, remains connected to the computer by USB, and that you have a battery ready but do not connect it right now, you will be told when to in later steps of this wizard.
Controller type: OpenPilot CopterControl 3D
Vehicle type: Multirotor
Vehicle sub type: Quadcopter X
Input type: PWM (One cable per channel)
Speed Controller (ESC) type: Rapid ESC (490 Hz)
CONNECTION DIAGRAM
< Back
Next >
Cancel

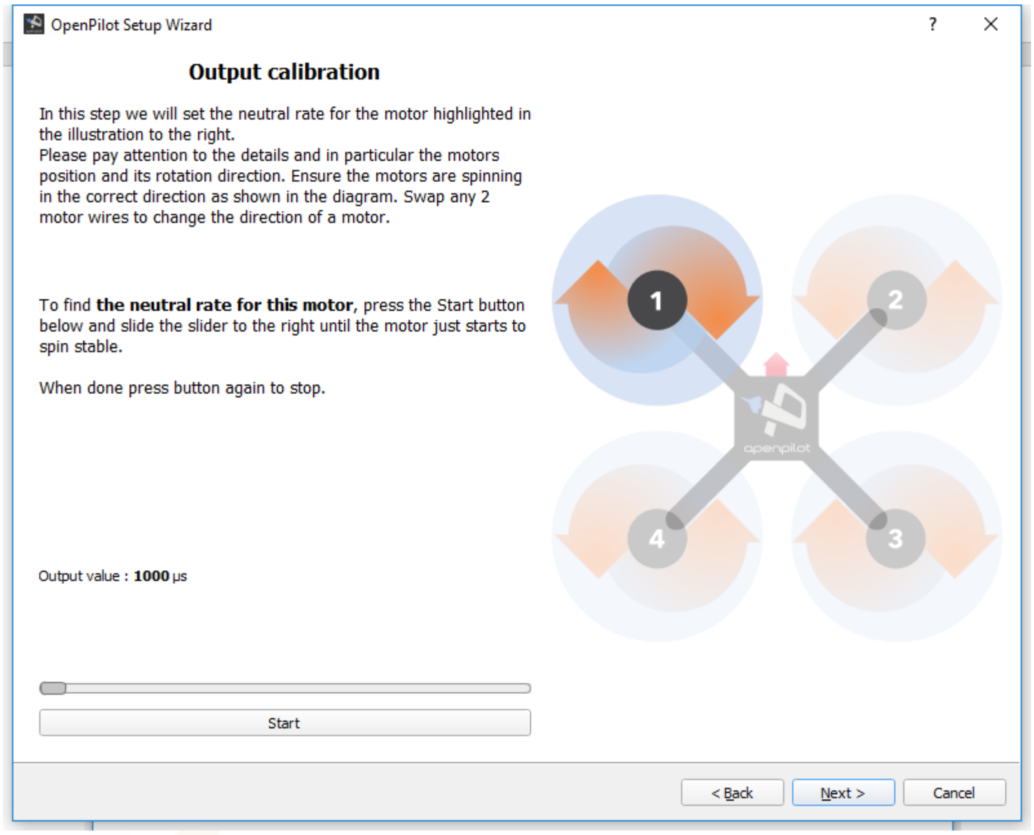

OpenPilot Setup Wizard
OpenPilot Sensor Calibration Procedure
The wizard needs to get information from the controller to determine in which position the vehicle is normally considered to be level. To be able to successfully perform these measurements, you need to place the vehicle on a surface that is as flat and level as possible. Examples of such surfaces could be a table top or the floor. Be careful to ensure that the vehicle really is level, since this step will affect the accelerometer and gyro bias in the controller software.
To perform the calibration, please push the Calculate button and wait for the process to finish.
Calculate
Done!
< Back
Next >
Cancel

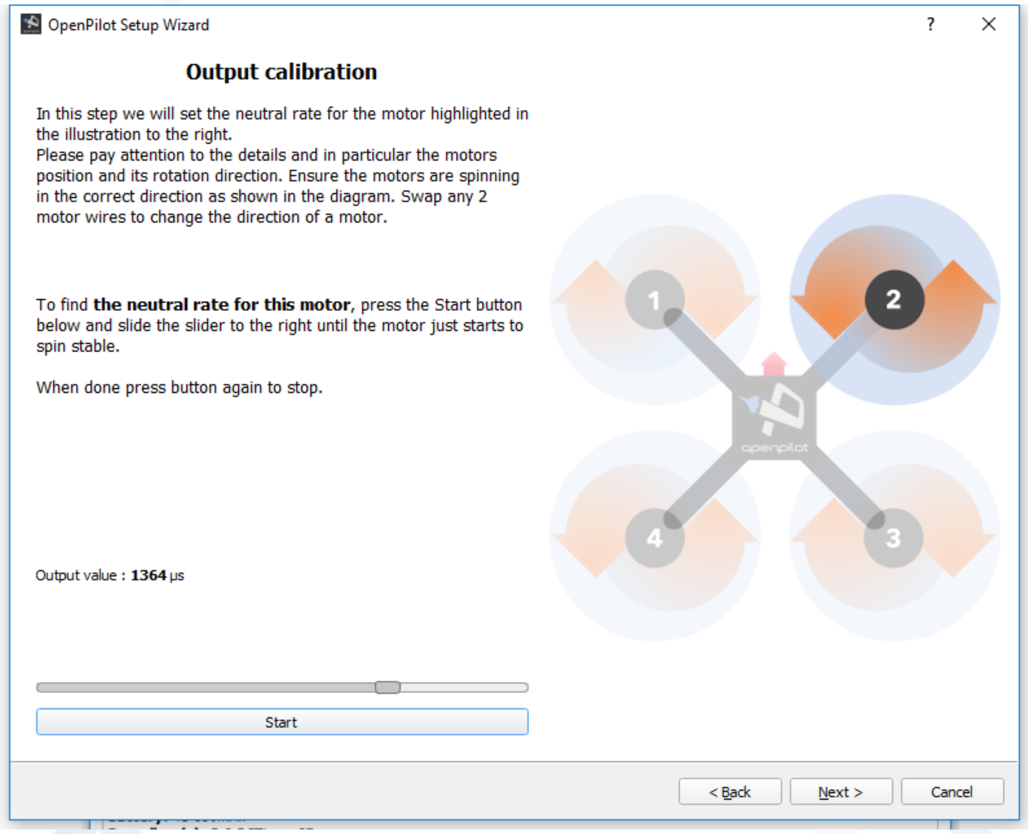

OpenPilot Setup Wizard
OpenPilot ESC Calibration Procedure
As you have selected to use a MultiRotor and Fast / Flashed ESCs, we need to calibrate the endpoints of these ESCs so they can see the full throttle range sent from the flight controller.
This part of the wizard will tell you to connect the battery to your aircraft, before doing so you absolutely must remove the propellers from all motors.
The steps to perform this calibration are as follows:
1. Confirm all safety questions,
2. Press the Start button when it becomes enabled, battery not connected,
3. Connect the battery to your airframe,
4. Wait for ESC calibration beep(s),
5. Press the Stop button,
6. Wait for ESC confirmation beep(s),
7. Disconnect battery.
I have removed ALL propellers from ALL motors of my vehicle.
The vehicle is NOT powered by any external power source but USB
I confirm i have read and understood the above instructions in full
BAT
ESC
BAT
ESC
Low/Off
900 µs
High
Start
Stop
< Back
Next >
Cancel

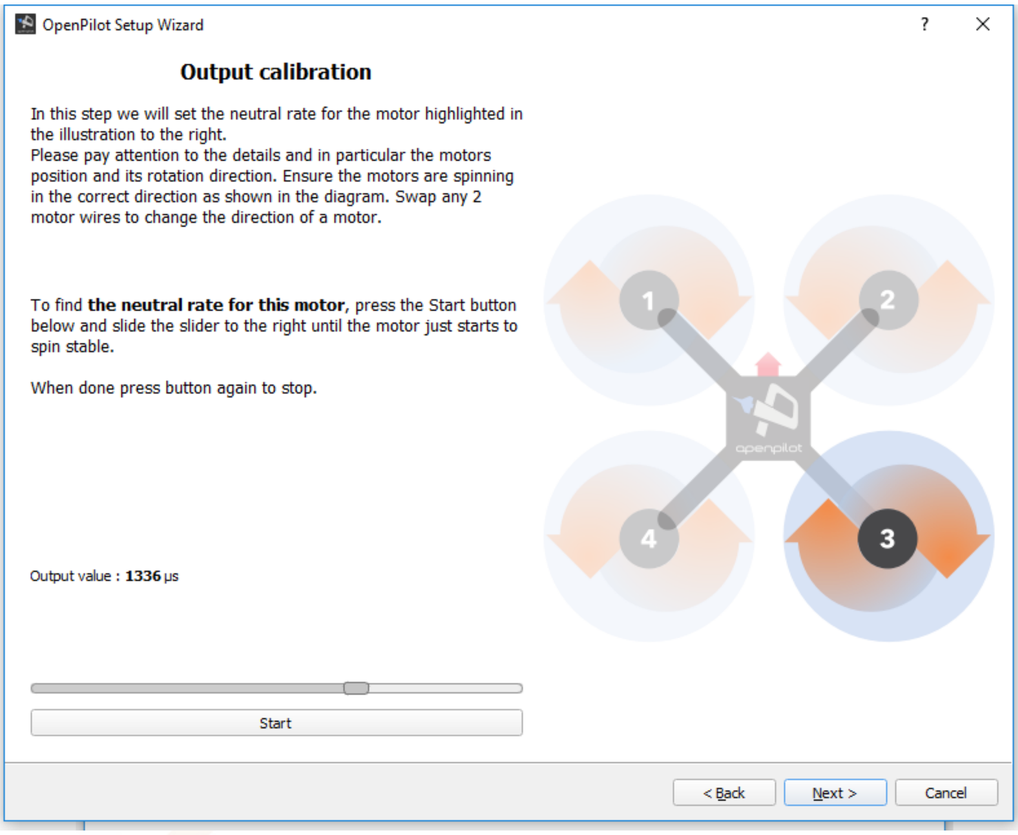


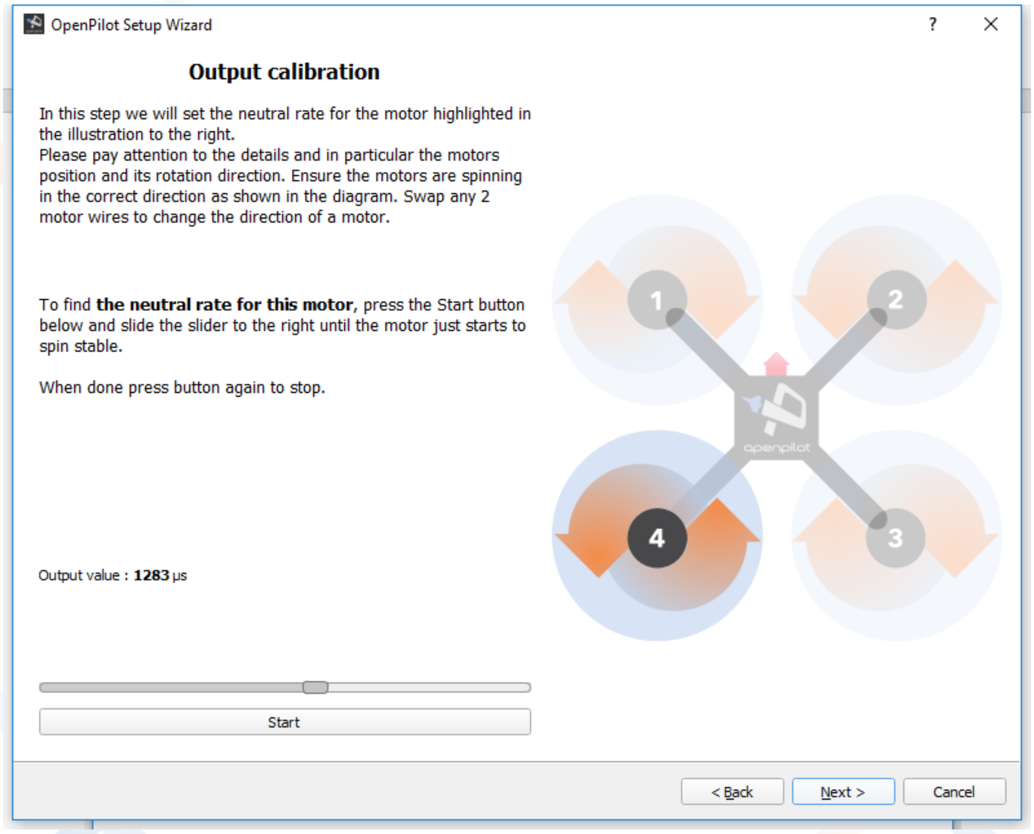
1.2. PRESS START, LATER PRESS STOP

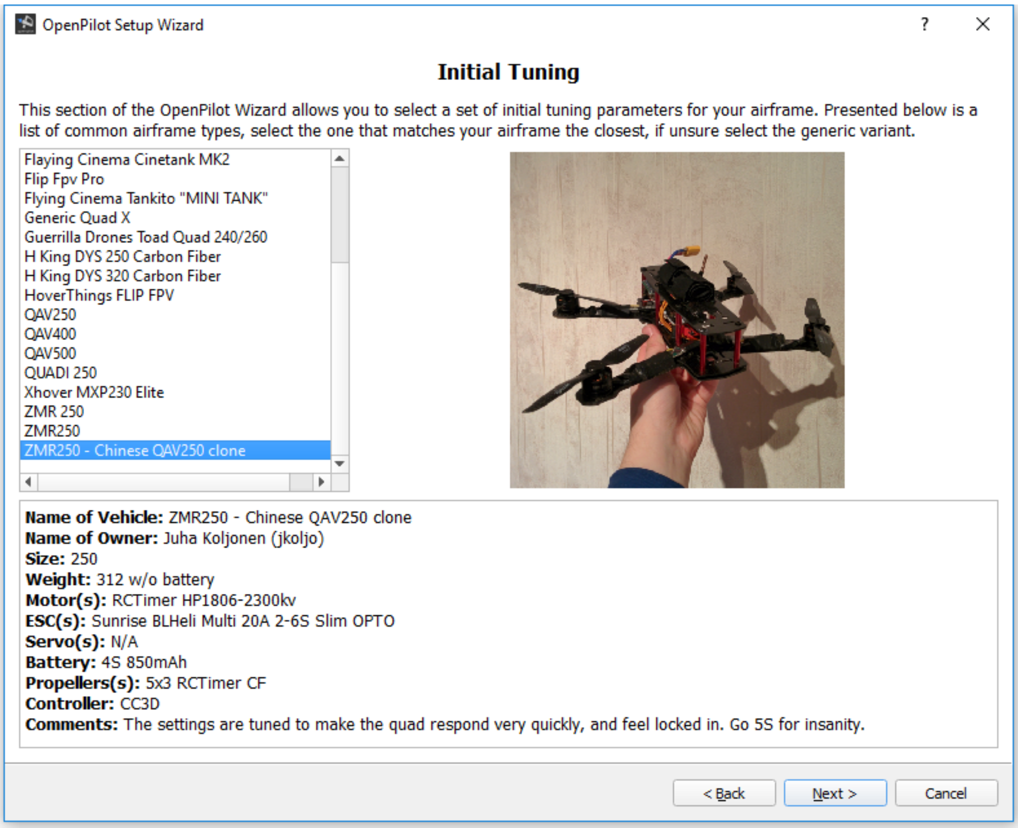



OpenPilot Setup Wizard
Output calibration
In this step we will set the neutral rate for the motor highlighted in the illustration to the right.
Please pay attention to the details and in particular the motors position and its rotation direction. Ensure the motors are spinning in the correct direction as shown in the diagram. Swap any 2 motor wires to change the direction of a motor.
To find the neutral rate for this motor, press the Start button below and slide the slider to the right until the motor just starts to spin stable.
When done press button again to stop.
Output value : 1000 µs
Start
1
2
3
4
openpilot
< Back
Next >
Cancel


OpenPilot Setup Wizard
Output calibration
In this step we will set the neutral rate for the motor highlighted in the illustration to the right.
Please pay attention to the details and in particular the motors position and its rotation direction. Ensure the motors are spinning in the correct direction as shown in the diagram. Swap any 2 motor wires to change the direction of a motor.
To find the neutral rate for this motor, press the Start button below and slide the slider to the right until the motor just starts to spin stable.
When done press button again to stop.
Output value : 1364 µs
Start
1
2
3
4
openpilot
< Back
Next >
Cancel


OpenPilot Setup Wizard
Output calibration
In this step we will set the neutral rate for the motor highlighted in the illustration to the right.
Please pay attention to the details and in particular the motors position and its rotation direction. Ensure the motors are spinning in the correct direction as shown in the diagram. Swap any 2 motor wires to change the direction of a motor.
To find the neutral rate for this motor, press the Start button below and slide the slider to the right until the motor just starts to spin stable.
When done press button again to stop.
Output value : 1336 µs
Start
< Back
Next >
Cancel
1
2
3
4


OpenPilot Setup Wizard
Output calibration
In this step we will set the neutral rate for the motor highlighted in the illustration to the right.
Please pay attention to the details and in particular the motors position and its rotation direction. Ensure the motors are spinning in the correct direction as shown in the diagram. Swap any 2 motor wires to change the direction of a motor.
To find the neutral rate for this motor, press the Start button below and slide the slider to the right until the motor just starts to spin stable.
When done press button again to stop.
Output value : 1283 µs
Start
< Back
Next >
Cancel
1
2
3
4



1.3. RECEIVER/TRANSMITER SETUP

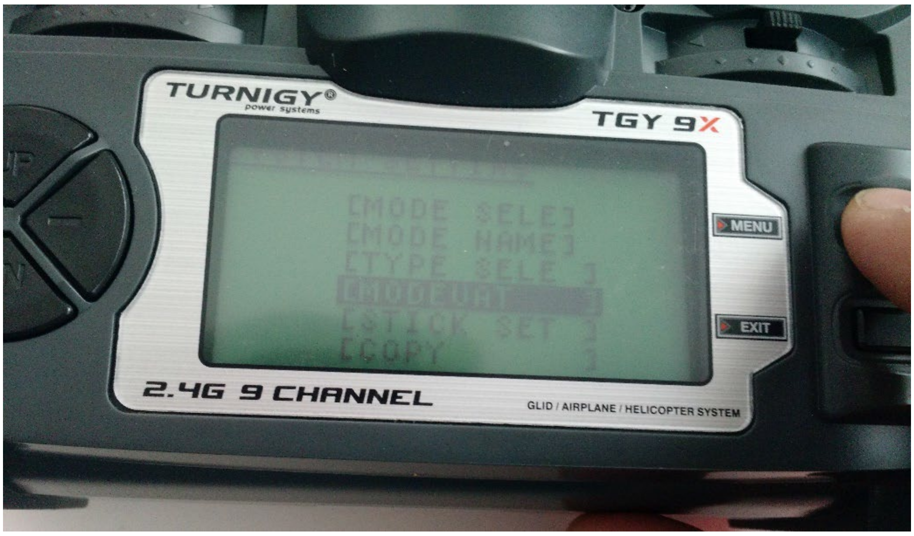
TRANSMITER SETUP


TURNIGY®
power systems
TGY 9X
[MODE SELE]
[MODE NAME]
[TYPE SELE]
[MODEVAT
[STICK SET
[COPY
MEN
EX
4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TGY 9X
TURNIGY
TURNIGY
TURNIGY
TURNIGY
TURNIGY
TURNIGY
TURNIGY
MENU
EXIT
GLID / AIRPLANE / HELICOPTER SYSTEM


TGY 9X
[MODE SELE]
[MODE NAME]
[TYPE SELE]
[MODEVAT]
[STICK SET]
[COPY]
MENU
EXIT
NNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
TGY 9X
UP
DN
HELI
ACRO
GLID
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
power systems
TGY 9X
[MODE SELE]
[MODE NAME]
[TYPE SELE]
[MODEUAT]
[STICK SET]
[COPY]
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM

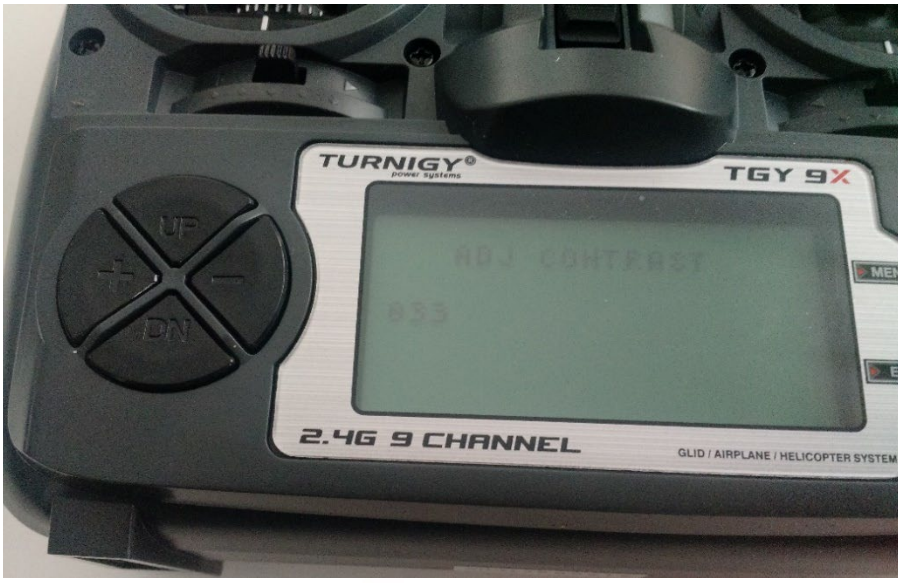

TGY 9X
PPM
PCM
MENU
EXIT
9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
power systems
TGY 9X
[MODE SELE]
[MODE NAME]
[TYPE SELE]
[MODEUAT]
[STICK SET]
[COPY]
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM

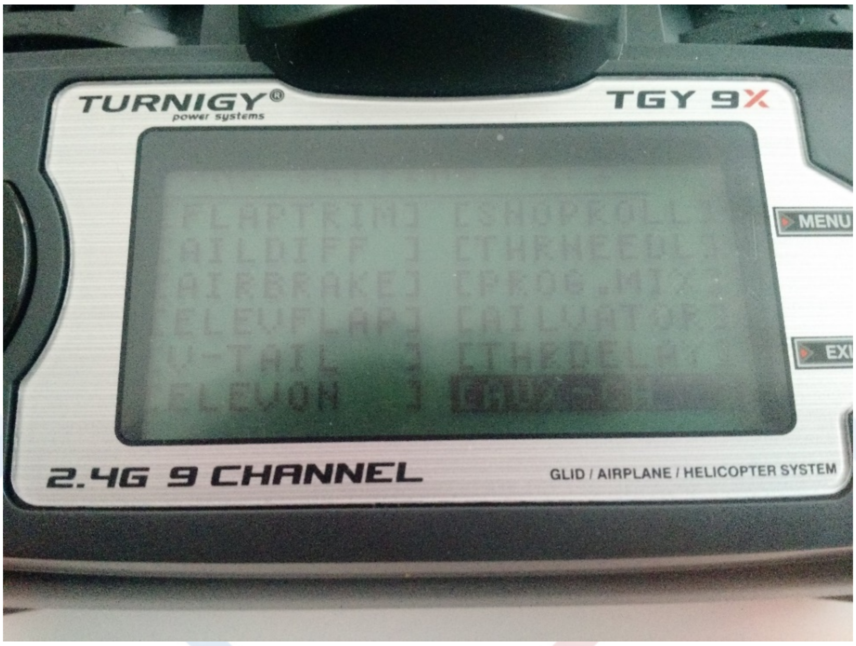

TURNIGY
power systems
TGY 9X
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
power systems
TGY 9X
THRO REVERSE?
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM

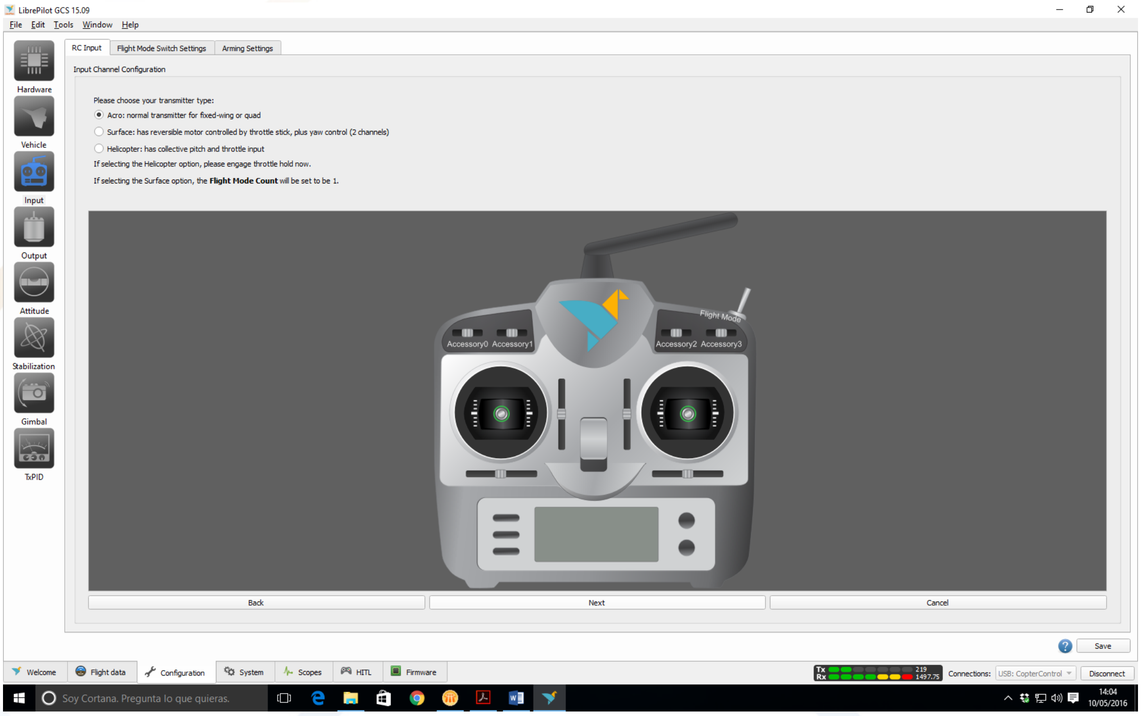

TURNIGY
TGY 9X
MENU
EXIT
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM

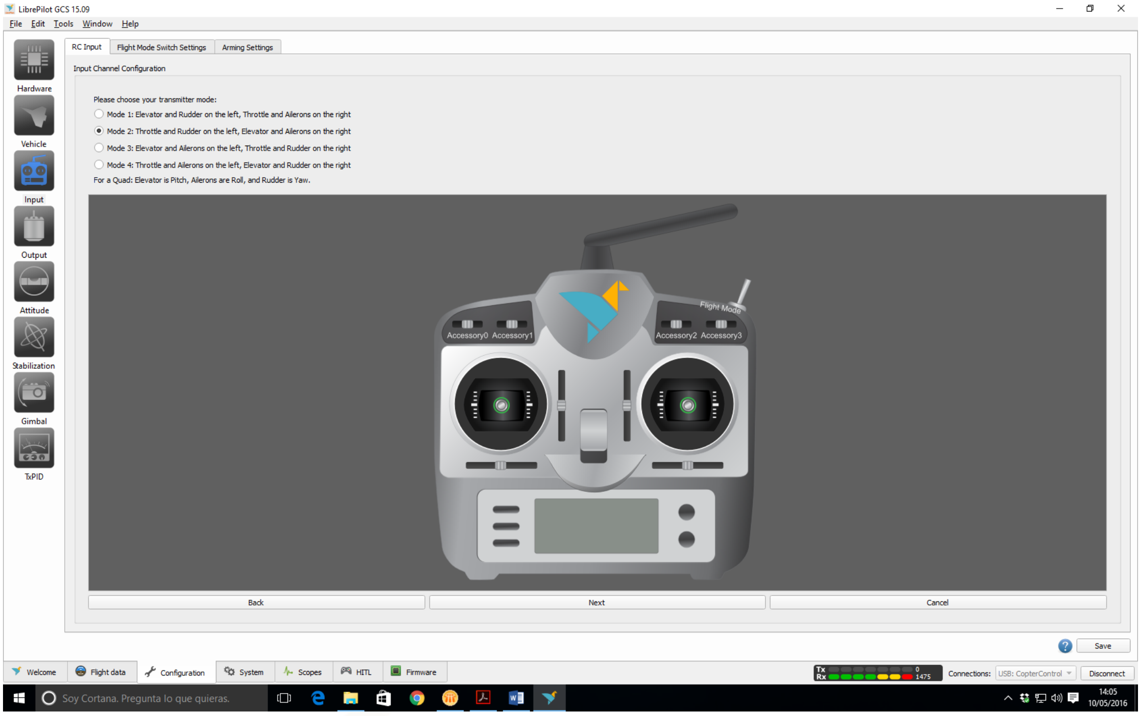

TURNIGY
TGY 9X
UP
DN
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
TGY 9X
UP
MENU
EXIT
FUNC SETTING
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM


TURNIGY
TGY 9X
MENU
2.4G 9 CHANNEL
GLID / AIRPLANE / HELICOPTER SYSTEM

1.4. LIBREPILOT





MOVE CONTROLLERS AS INDICATION IN SCREEN

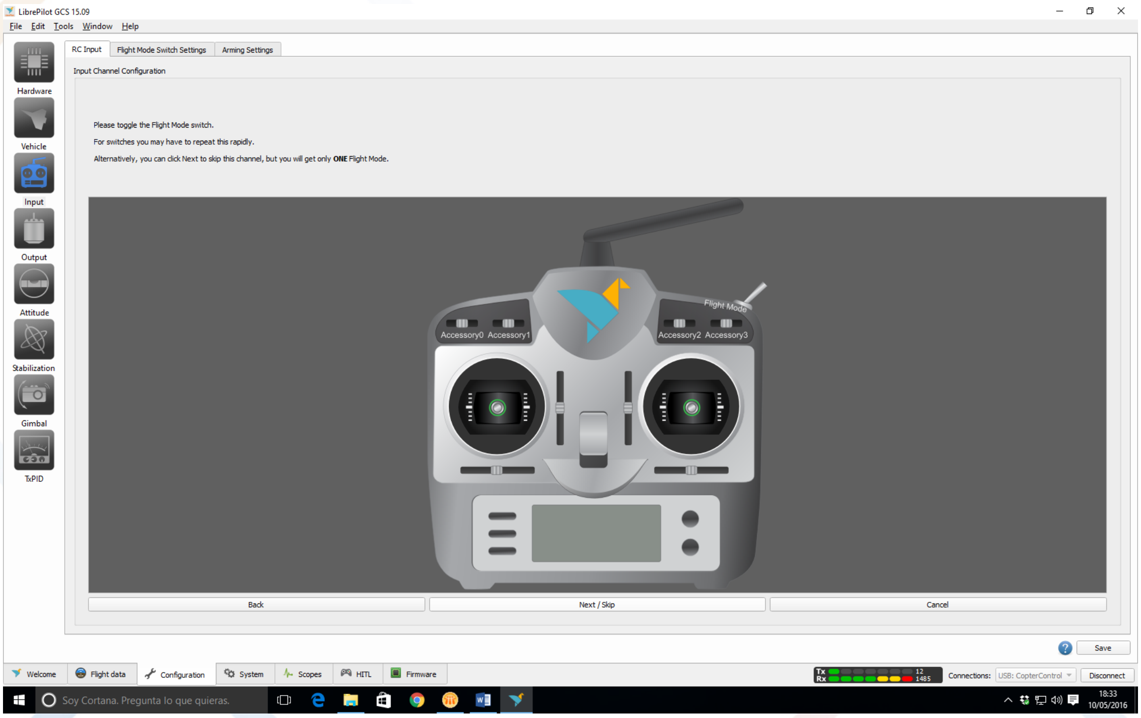
SKIP FLIGHT CONTROLLER SETUP

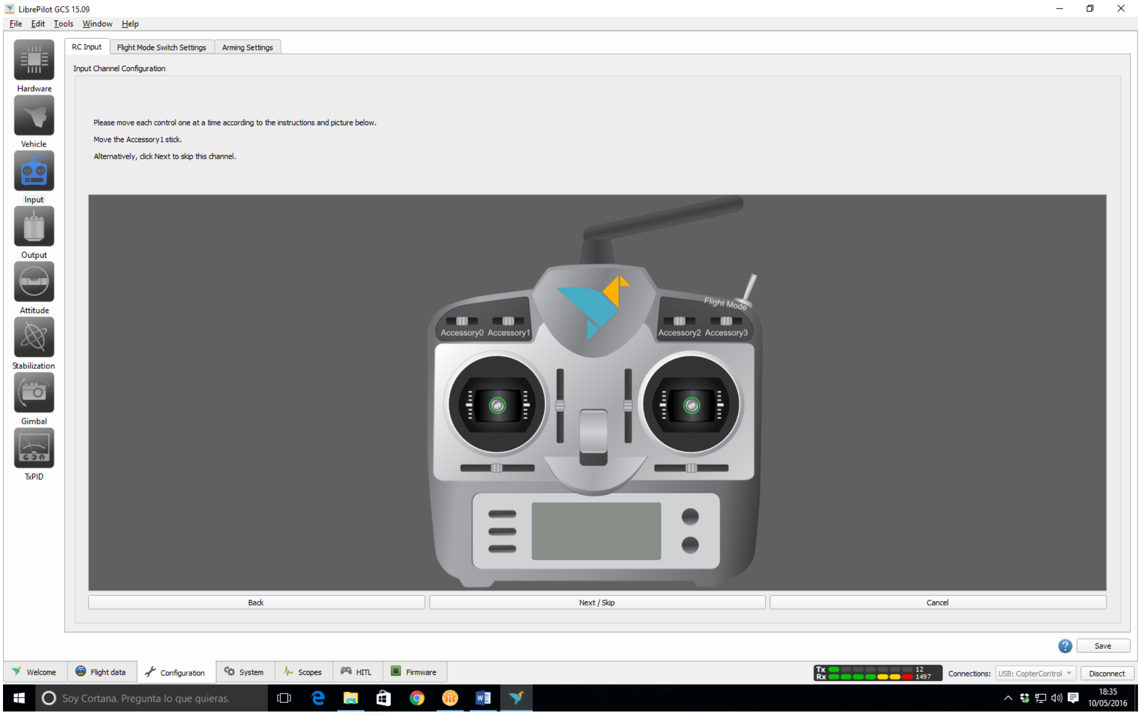
SKIP ACCESSORIES 01 SETUP

SKIP ACCESSORIES 02 SETUP

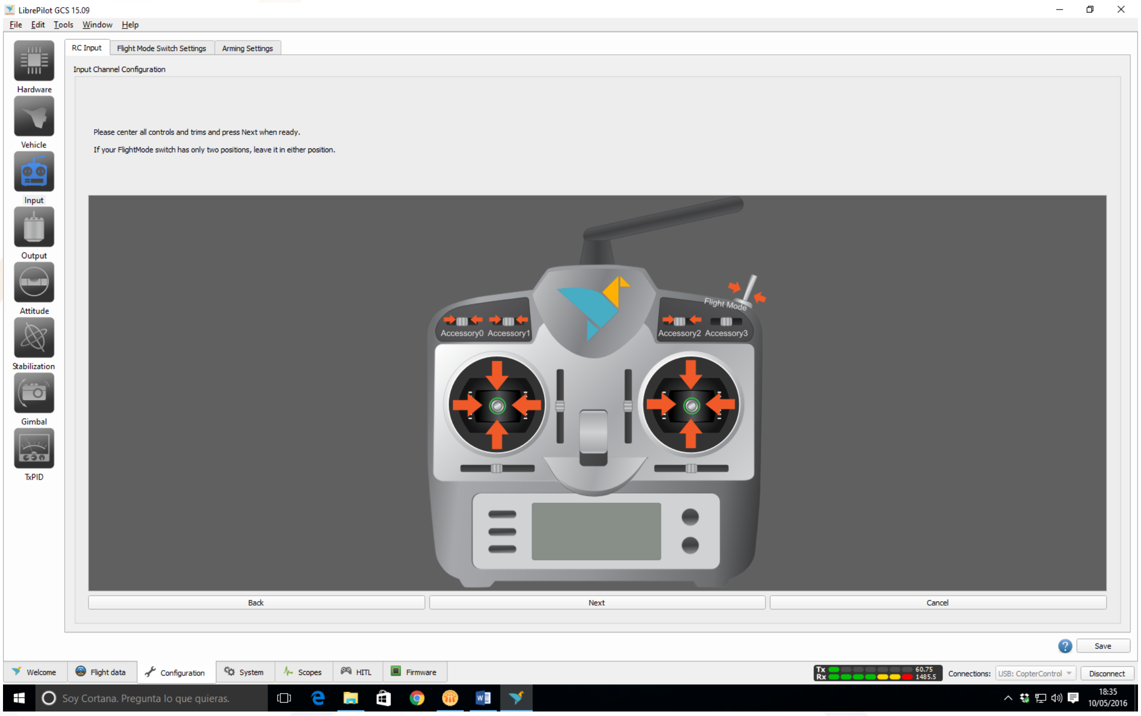
SKIP ACCESSORIES 03 SETUP

SKIP ACCESSORIES 04 SETUP

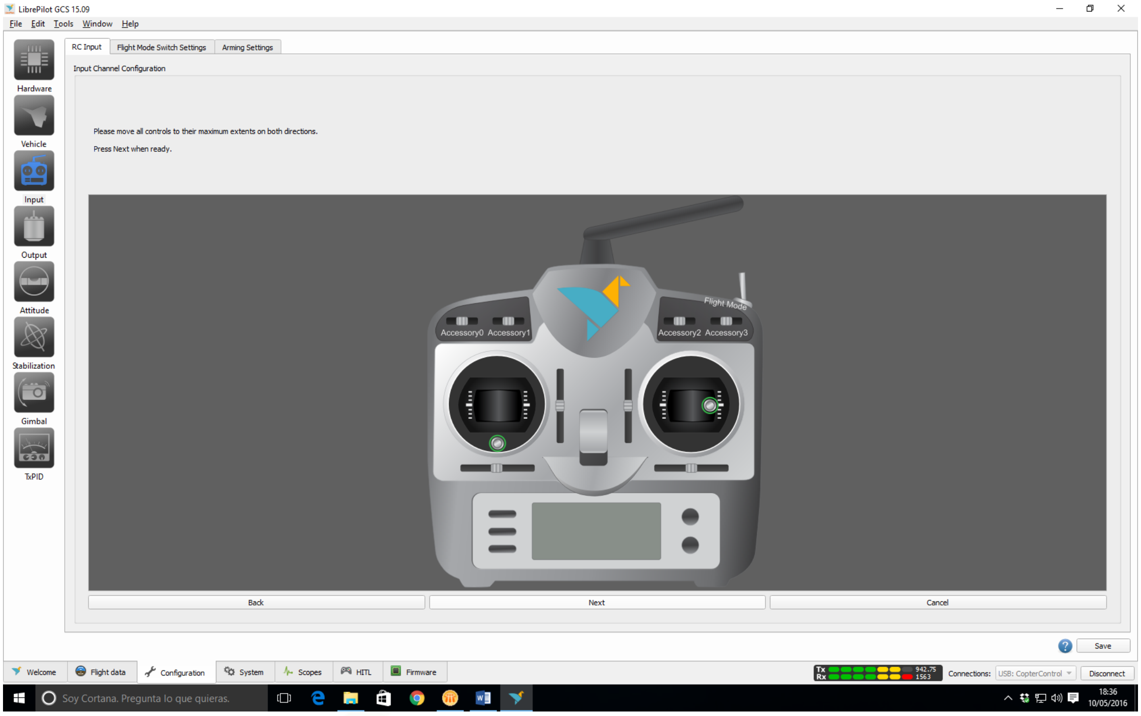




CENTER STICKERS



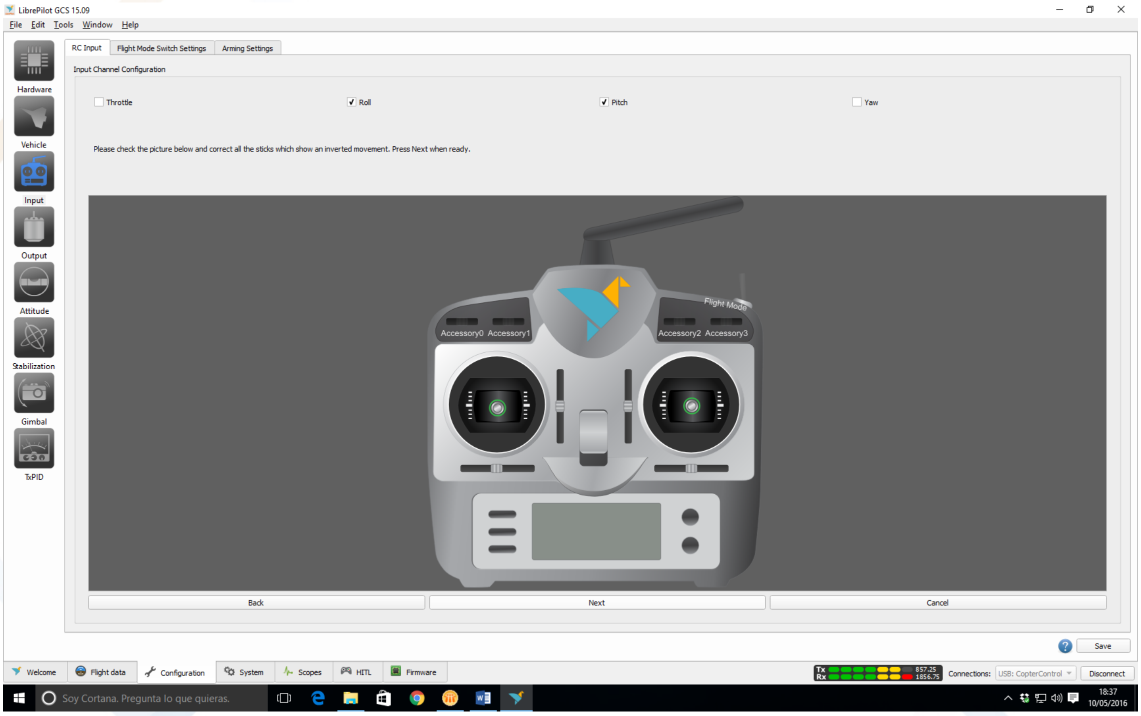
MOVE STICKERS TO BE SURE THAT MOVE IN CORRECT DIRECTION



IF ANY STICKER MOVE INVERTED, IN NEXT STEP YOU CAN REVERSE STICKERS MOVEMENT JUST CLICK IN CHECKING BOX

(THROTTLE (UP/DOWN), YAW (LEFT/RIGHT)→ LEFT STICKER)

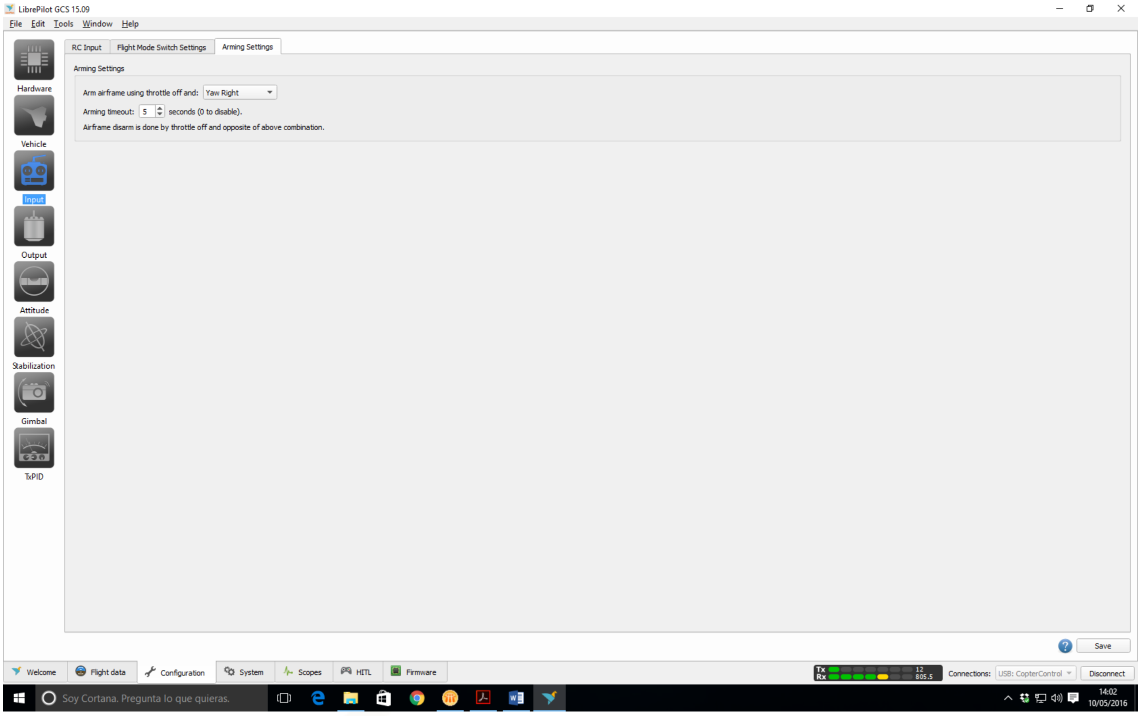
(PITCH (UP/DOWN), ROLL (LEFT/RIGHT) → RIGHT STICKER)



LAST STEP IS RELATED TO ARMING MOTORS

IN ARMING SETTING, ARM AIR FRAME USING THROTTLE OFF AND “YAW RIGHT”

TIMEOUT “5” SECONDS

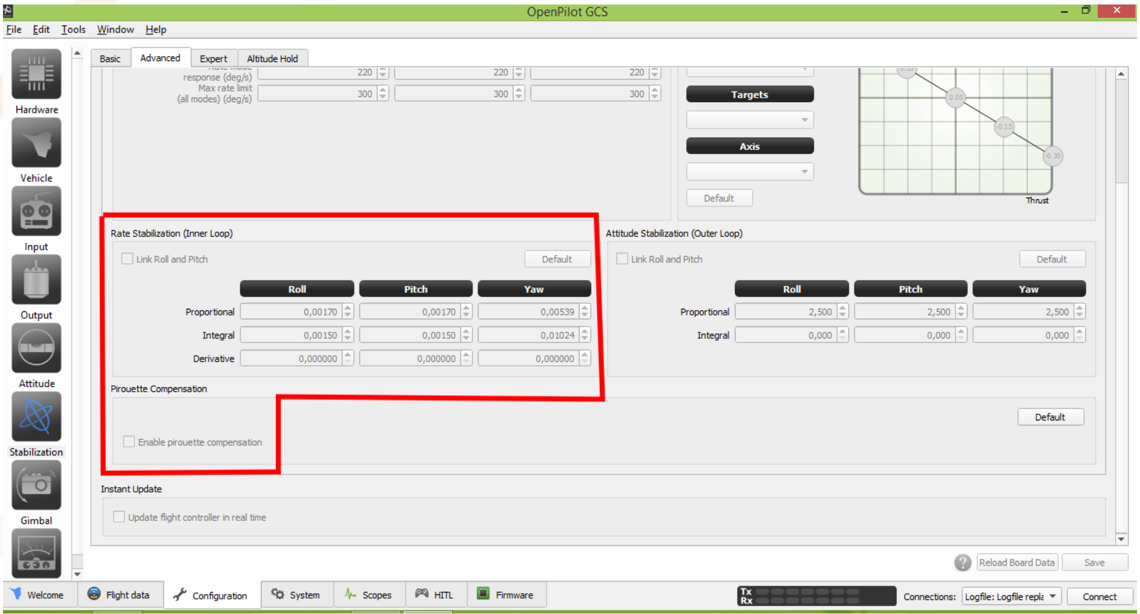


ARMING MOTORS (5 seconds)

DISARMING MOTORS (5 seconds)

WARNING: Some firmware versions are not compatible with the CC3D board, we recommend for now to don't upgrade the firmware and just skip that step in the Wizard.

Once you have finished the wizard you should add the following for the PID settings.

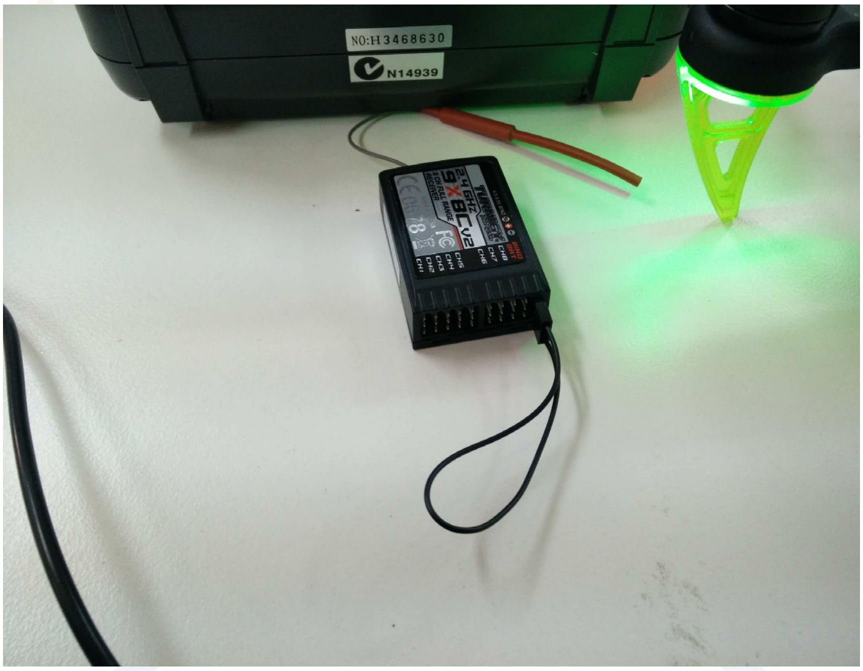


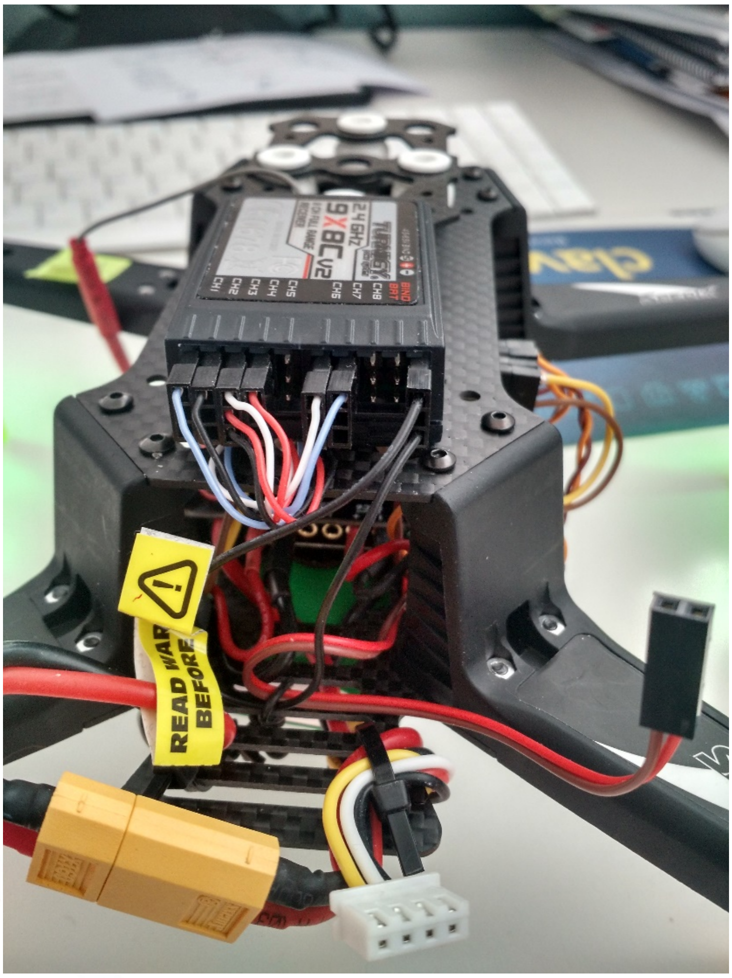
BINDING TRANSMISOR/RECEIVER (ONLY IF RED LIGHT IN NOT FIXED IN RECEIVER)

Binding is necessary to teach the receiver the code of the specific transmitter so that they can talk to each other.

NOTE: This would be not necessary if you are using the Transmitter-Receiver that are packed in the same box.

1. Connect the Turnigy 2.2 battery into the Transmitter (Don't turn on the transmitter).
2. Connect the binding wire(Jumper) in the bind port of the Receiver.



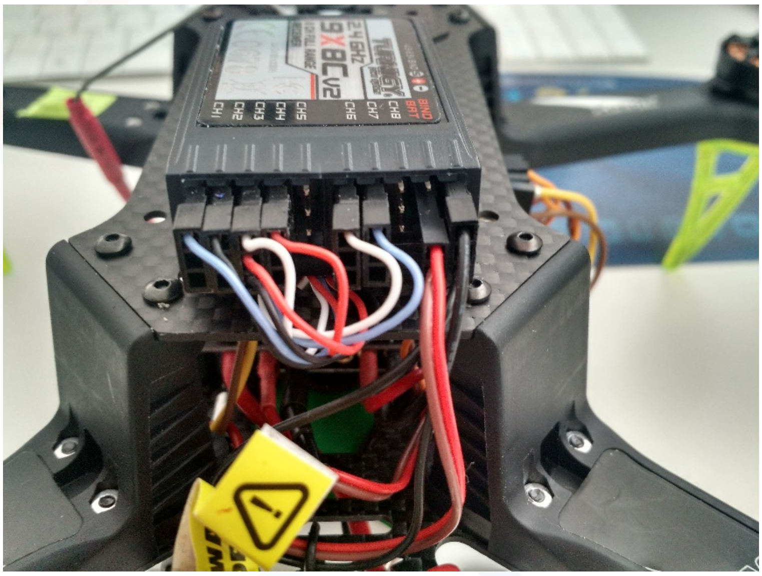


3. Using the Transmitter (power off) hold the button (Bind range test) and switch on the transmitter.

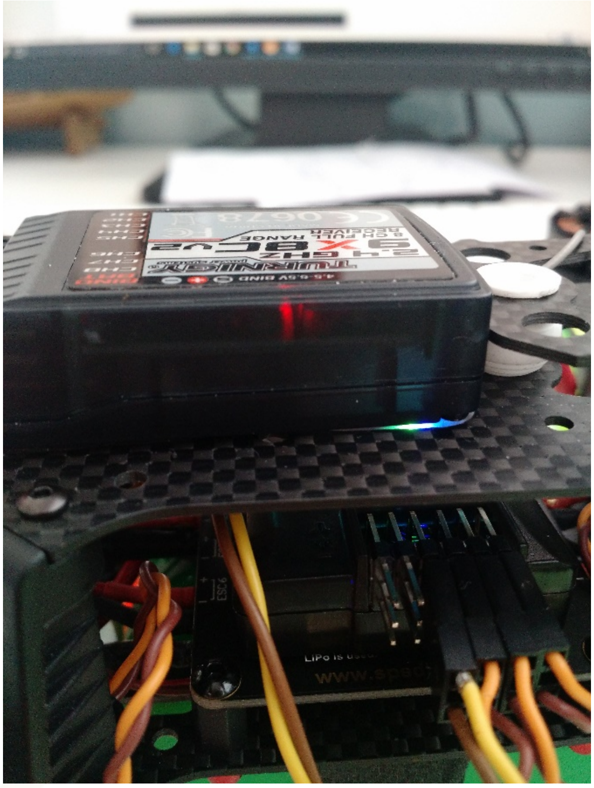
4. Keeping the Bind Range Test held go to the next step.

5. Power up the Receiver, using the 3 wire cable connector and plug it into the Receiver power port.

6. Turn off the transmitter, take off the power cable of the receiver and take off the bind cable of the receiver as well.



7. If this this step has been made properly whenever you turn on the transmitter the led in the receiver should go on (steady red light), or off if you turn off the transmitter.



2. REMOTE CONTROL.

New extra activity is use Arduino for remote control.

To create a connection between a drone and a phone we had to establish a wireless connection via Wi-Fi.

The first step was to turn an esp32 into access point mode. This mode is available from the public esp32 library.

Mavlink is a communication protocol with small vehicles such as drones, cars, etc. It is used in the Mission Planner software, a ground control station for drones. The large openness of the protocol and community support allow you to customize your Mavlink.

The basics of the Mavlink protocol and how to implement it for Arduino is described.

Introduction to the protocol

Using the protocol in practice is not particularly difficult and can be implemented on many platforms. Nevertheless, the basics are an advanced topic.

Much more about the Mavlink can be read here:

Before we start working with the code, we must declare our environment:

```
#include <Arduino.h>
#include <mavlink.h>

int sysid = 255;//GCS                   ///< ID 20 for this airplane.
1 PX, 255 ground station
int compid = 190;//Mission Planner                ///< The component
sending the message
int type = MAV_TYPE_QUADROTOR;   ///< This system is an airplane /
fixed wing

// Define the system type, in this case an airplane -> on-board
controller
uint8_t system_type = MAV_TYPE_GENERIC;
uint8_t autopilot_type = MAV_AUTOPILOT_GENERIC;

// Hardware definitions
uint8_t system_mode = MAV_MODE_TEST_ARMED; /// /* UNDEFINED mode. This
solely depends on the autopilot - use with caution, intended for
developers only. | */
uint32_t custom_mode = MAV_MODE_FLAG_SAFETY_ARMED; ///< Custom mode,
can be defined by user/adopter
uint8_t system_state = MAV_STATE_STANDBY; ///< System ready for flight
```

At the beginning, we import the Mavlink and Arduino library and then define the variables representing our device. The dependent will be whether the protocol will correctly interpret the data. The set of all possible configurations can be found in the common file.

Basics of communication

Mavlink support requires us to declare a message variable and a buffer. These variables used when sending and receiving commands:

```
// Initialize the required buffers
mavlink_message_t msg;
uint8_t buf[MAVLINK_MAX_PACKET_LEN];
```

The rules of communication

Devices that work with the Mavlink protocol do not send or receive any data themselves. Everything we want to do we have to signal with a command (pack) or request (request).

```
//Request a data from a device
mavlink_msg_request_data_stream_pack(2, 200, &msg, 1, 0,
MAVStreams[i], MAVRates[i], 1);

//Send a heartbeat packet
mavlink_msg_heartbeat_pack(255,0, &msg, type, autopilot_type,
system_mode, custom_mode, system_state);
```

Connection indication

The Mavlink for signalling the connection state uses the heartbeat object. In later coding it is rather unnecessary. The package itself can extract information such as the current mode or protocol version.

```
mavlink_message_t msg;
mavlink_status_t status;

mavlink_heartbeat_t hb;
mavlink_msg_heartbeat_decode(&msg,&hb);

        #ifdef DEBUG
          Serial.print(millis());
          Serial.print("\ncustom_mode:
");Serial.println(hb.custom_mode);
          Serial.print("Type: ");Serial.println(hb.type);
          Serial.print("autopilot: ");Serial.println(hb.autopilot);
          Serial.print("base_mode: ");Serial.println(hb.base_mode);
          Serial.print("system_status:
");Serial.println(hb.system_status);
          Serial.print("mavlink_version:
");Serial.println(hb.mavlink_version);
```

Arduino with Mavlink - Reading data

By using the Mavlink gadget we can read and interpret several information from the device. In following steps, we will describe how to do it.

The protocol itself only issues one information. This information is the Heartbeat package. To receive any other data, we must ask for it in advance. How to do it?

```
// Pack the message
mavlink_msg_heartbeat_pack(255,0, &msg, type, autopilot_type, system_mode, custom_mode, system_state);
uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
```

The first thing to do is declare some time interval. Thanks to this, Mavlink will send us a new portion of information from time to time. At the beginning, variables. Such declaration of variables in the code will wait with the receipt of data streams minute and the interval will be called every second. In a later call, the code looks something like this:

```
void Mav_Request_Data()
{
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  // To be setup according to the needed information to be requested from the Pixhawk
  const int  maxStreams = 1;
  const uint8_t MAVStreams[maxStreams] = {MAV_DATA_STREAM_ALL};
  const uint16_t MAVRates[maxStreams] = {0x02};

  for (int i=0; i < maxStreams; i++) {
    mavlink_msg_request_data_stream_pack(2, 200, &msg, 1, 0, MAVStreams[i], MAVRates[i], 1);
    uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
    SerialMAV.write(buf, len);
  }
}
```

In the code above, two lines are very important. In line 9 on the SerialMAV object (serial connection is created), we call the write () method. Thanks to this, we will save the data request to the buffer.

In turn, the function for data streams is called on line 15.

Data request

In order for Mavlink to send us information, we must first enter the required package into the buffer and specify the parameters for it. We call these two functions before the code above.

```
// Pack the message
mavlink_msg_heartbeat_pack(255,0, &msg, type, autopilot_type,
system_mode, custom_mode, system_state);
uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
```

Data streams

The packages that will flow to the receiving device can be defined via Mavlink streams.

```
void Mav_Request_Data()
{
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  // To be setup according to the needed information to be requested
from the Pixhawk
  const int  maxStreams = 1;
  const uint8_t MAVStreams[maxStreams] = {MAV_DATA_STREAM_ALL};
  const uint16_t MAVRates[maxStreams] = {0x02};

  for (int i=0; i < maxStreams; i++) {
    mavlink_msg_request_data_stream_pack(2, 200, &msg, 1, 0,
MAVStreams[i], MAVRates[i], 1);
    uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
    SerialMAV.write(buf, len);
  }
}
```

In the MAVStreams array, we need to choose the range of packages. These constants can be found in the common file. Below are some of the more important ones.

MAV_DATA_STREAM_ALL - All packages

MAV_DATA_STREAM_RAW_SENSORS - Raw data for: GPS, IMU

MAV_DATA_STREAM_EXTENDED_STATUS - GPS Status, Control Status, AUX Status

MAV_DATA_STREAM_RC_CHANNELS - RC (Radio control) channels and their variations like RAW or SCALED

MAV_DATA_STREAM_RAW_CONTROLLER - Altitude parameters, controller output

MAV_DATA_STREAM_POSITION - Local / global positions

In the MAVRates table, we define frequencies for streams. It is not entirely clear to me how this is determined.

Interpretation of data

When the information is already in the buffer, we can finally create a switch with fixed Mavlink to extract data from the device. The code below is an implementation for the APM 2.6 drone.

```
mavlink_message_t msg;
mavlink_status_t status;

while(SerialMAV.available()) {
    uint8_t c = SerialMAV.read();

    // Try to get a new message
    if(mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {

      case MAVLINK_MSG_ID_HEARTBEAT:  // #0: Heartbeat
          {
            // E.g. read GCS heartbeat and go into
            // comm lost mode if timer times out
            //Serial.println("MAVLINK_MSG_ID_HEARTBEAT");
            mavlink_heartbeat_t hb;
            mavlink_msg_heartbeat_decode(&msg, &hb);
            Serial.print("State: "); Serial.println(hb.base_mode ==
209 ? "Armed" : "Disarmed");
            Serial.print("Mode: ");
            switch(hb.custom_mode) {
              case 0:
                Serial.println("Stabilize");
              break;
              case 2:
                Serial.println("AltHold");
              break;
              case 3:
                Serial.println("Auto");
              break;
              case 5:
                Serial.println("Loiter");
              break;
              case 7:
                Serial.println("Circle");
              break;
              default:
                Serial.println("Mode not known");
              break;
            }
          }
      break;
      case MAVLINK_MSG_ID_SYS_STATUS:  // #1: SYS_STATUS
          {
            /* Message decoding: PRIMITIVE
             *    mavlink_msg_sys_status_decode(const
mavlink_message_t* msg, mavlink_sys_status_t* sys_status)
             */
            mavlink_sys_status_t sys_status;
```

```
             mavlink_msg_sys_status_decode(&msg, &sys_status);
             Serial.println("Battery (V): ");
             Serial.println(sys_status.voltage_battery);
           }
      break;
      case MAVLINK_MSG_ID_ATTITUDE:  // #30
           {
             /* Message decoding: PRIMITIVE
              *    mavlink_msg_attitude_decode(const mavlink_message_t*
msg, mavlink_attitude_t* attitude)
              */
             mavlink_attitude_t attitude;
             mavlink_msg_attitude_decode(&msg, &attitude);
             Serial.println("ROLL: ");
             Serial.println(attitude.roll);
           }
           break;
           //Not overriden channels
           case MAVLINK_MSG_ID_RC_CHANNELS_RAW: // #35
           {
            /*
            *  RC (Radio controll) channels are the inputs and outputs
for controlling all
            *  actions called from joystick / mission planner. E.g.
arm, throttle, pitch.
            */
             mavlink_rc_channels_raw_t chs;
             mavlink_msg_rc_channels_raw_decode(&msg, &chs);

             Serial.print("Roll: ");  Serial.print(chs.chan1_raw);
             Serial.println();
             Serial.print("Pitch: ");  Serial.print(chs.chan2_raw +
'\n');
             Serial.println();
             Serial.print("Throttle: ");  Serial.print(chs.chan3_raw +
'\n');
             Serial.println();
           }
           break;
    }
}
```

This is only part of the information we can get through the Mavlink protocol. Finally, the whole code should be placed in the loop () function in Arduino. All other constants needed for the implementation can of course be found in the common file.

Arduino with Mavlink - Send commands

The Mavlink protocol also allows us to send commands to devices. Thanks to this, we can control the rotation of the drone motors.

Introduction

Sending commands and in this case is based on cyclical saving of information packets to the buffer. In my code I wanted to create an implementation for controlling drone data such as: flight mode, arming and RC channels.

At the beginning, we must declare global variables that store the current state of the drone.

```
boolean current_arm = false;
String current_mode = STABILIZE;
int current_roll = 0;
int current_pitch = 0;
int current_throttle = 0; //Min value is 1150 to run motors
int current_yaw = 0;
```

In most online tutorials, the code for receiving and sending data is mixed together. We decided to make the code somewhat more flexible and transferred every action to the function with parameters.

```
void loop() {

  // Initialize the required buffers
  mavlink_rc_channels_override_t sp;
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  //We have to send the heartbeats to indicate side by side connection
  mav_heartbeat_pack();

  mav_set_mode(current_mode);

  mav_arm_pack(current_arm);

  // ROLL, PITCH, THROTTLE, YAW
  mav_override_rc(current_roll, current_pitch, current_throttle,
current_yaw);

  ...
}
```

Now we will go on to describe each of them.

Broadcasting a heartbeat

This is the simplest function and its calling contains only system parameters. Mavlink to send the heartbeat package will answer us the same.

This is what functions to call and what arguments can be found in the mavlink_msg_heartbeat.h file in the mavlink / common / folder. We do the same for other commands.

```
void mav_heartbeat_pack() {
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  // Pack the message
  mavlink_msg_heartbeat_pack(255,0, &msg, type, autopilot_type,
system_mode, custom_mode, system_state);
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  SerialMAV.write(buf, len);
}
```

Change of flight mode

The change of the flight mode is based on the mavlink_msg_set_mode_pack function call. This function for APM 2.6 worked when the target_system parameter is 1. We define the given flight mode with the custom_mode parameter in the last place.

```
void mav_set_mode(String value) {
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  value.trim();

  //SET_MODE
  //Works with 1 at 4'th parameter
  if (value == STABILIZE){
    mavlink_msg_set_mode_pack(0xFF, 0xBE, &msg, 1, 209, 0);
  }

  if (value == ALTHOLD){
    mavlink_msg_set_mode_pack(0xFF, 0xBE, &msg, 1, 209, 2);
  }

  if (value == LOITER){
    mavlink_msg_set_mode_pack(0xFF, 0xBE, &msg, 1, 209, 5);
  }

  if (value == AUTO){
    mavlink_msg_set_mode_pack(0xFF, 0xBE, &msg, 1, 209, 3);
  }

  if (value == CIRCLE){
    mavlink_msg_set_mode_pack(0xFF, 0xBE, &msg, 1, 209, 7);
  }

  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  SerialMAV.write(buf, len);
}
```

Arming the drone

The use of this action made a problem, because in the Mavlink to arm and disarm we use the so-called long command (command_long) MAV_CMD_COMPONENT_ARM_DISARM (400).

```
void mav_arm_pack(boolean state) {
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  //Arm the drone
  //400 stands for MAV_CMD_COMPONENT_ARM_DISARM
  // 1 an 8'th argument is for ARM (0 for DISARM)
  if(state) {
    //ARM
    mavlink_msg_command_long_pack(0xFF, 0xBE, &msg, 1, 1, 400,
1,1.0,0,0,0,0,0,0);
  }else {
    //DISARM
```

```
    mavlink_msg_command_long_pack(0xFF, 0xBE, &msg, 1, 1, 400,
1,0.0,0,0,0,0,0,0);
  }
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  SerialMAV.write(buf, len);
}
```

What is needed for this is the mavlink_msg_command_long_pack function, where as parameter 6 we define the use of just a long command. They are used as commands during mission planning, for example at Mission Planner. The next 8 parameters are the parameters of the long command.

Adjusting RC channels

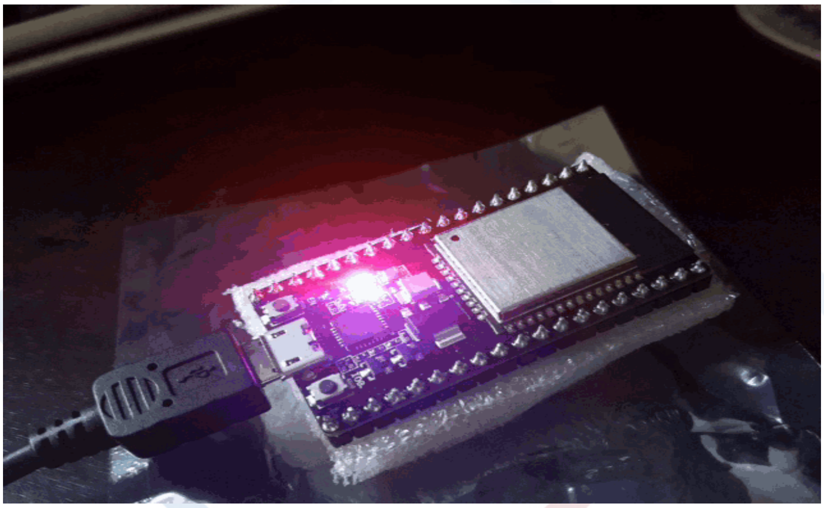
RC channels (Radio control) are responsible for signals from the apparatus: this joystick to control the device.

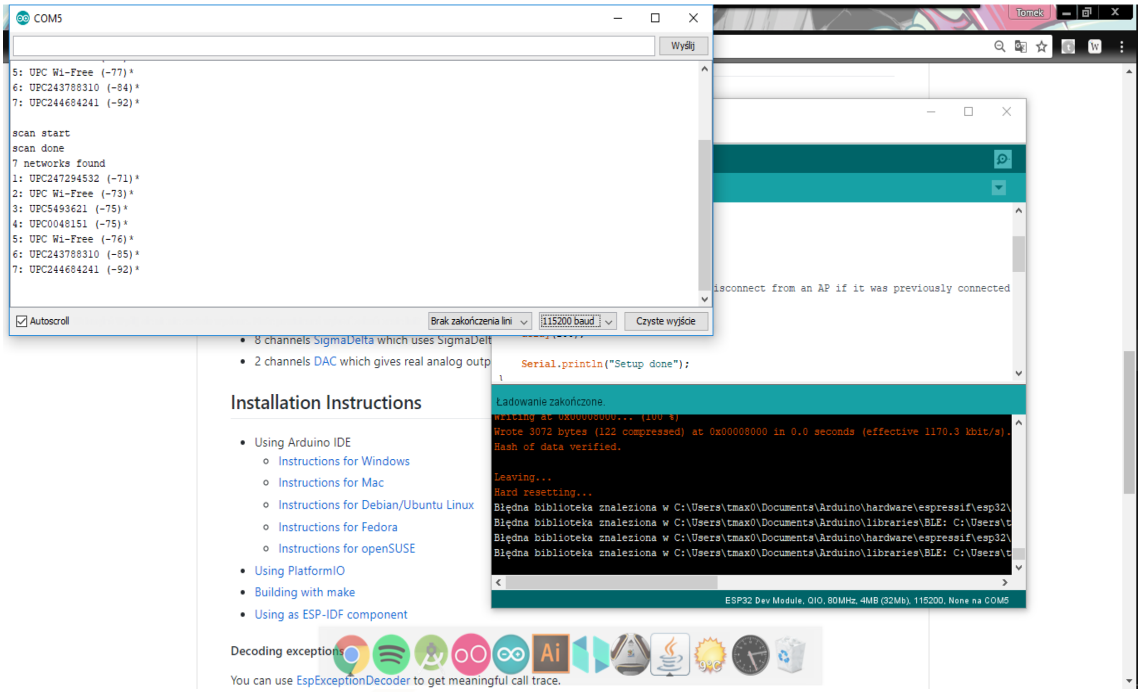
Thanks to them, we can introduce our own values, e.g. for throttle or yaw rods. Generally, it is possible to overwrite all elements of the apparatus, but we are limited to 8 channels.

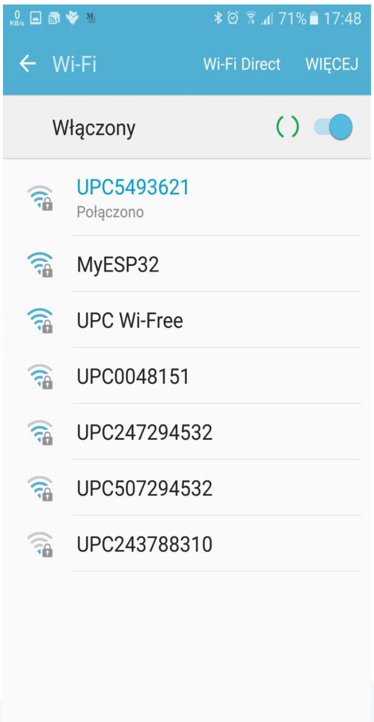
```
void mav_override_rc(int roll, int pitch, int throttle, int yaw) {
  mavlink_message_t msg;
  uint8_t buf[MAVLINK_MAX_PACKET_LEN];

  mavlink_msg_rc_channels_override_pack(0xFF, 0xBE, &msg, 1, 1, roll,
pitch, throttle, yaw, 0, 0, 0, 0);
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
  SerialMAV.write(buf, len);
}
```

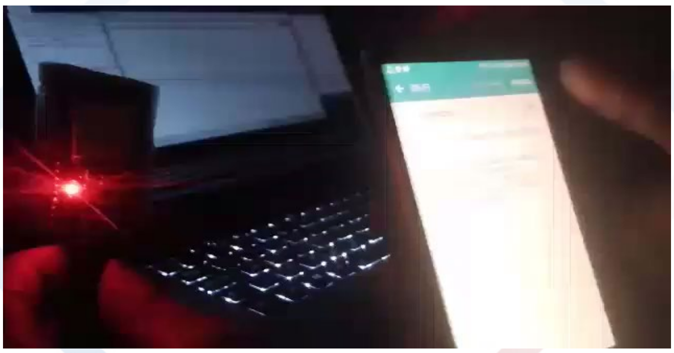
In the case of APM 2.6, we had to check the channels in Mission Planner. You can do it with the help of even any pad in the emulation mode of the apparatus in the program.

Indication of a wireless connection

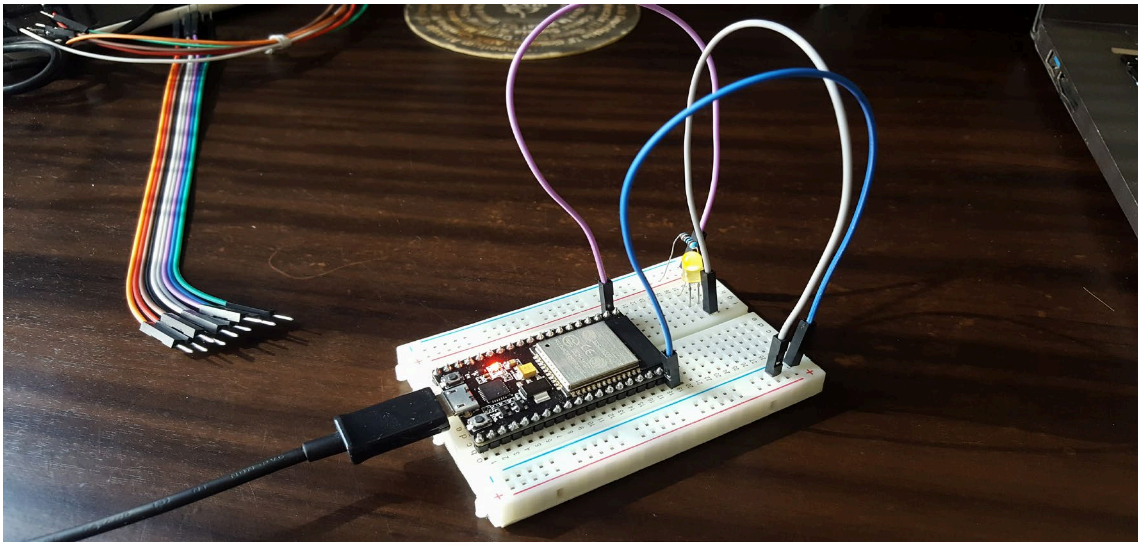



COM5
Wyślij
5: UPC Wi-Free (-77)*
6: UPC243788310 (-84)*
7: UPC244684241 (-92)*
scan start
scan done
7 networks found
1: UPC247294532 (-71)*
2: UPC Wi-Free (-73)*
3: UPC5493621 (-75)*
4: UPC0048151 (-75)*
5: UPC Wi-Free (-76)*
6: UPC243788310 (-85)*
7: UPC244684241 (-92)*
Autoscroll
Brak zakończenia linii
115200 baud
Czyste wyjście
isconnect from an AP if it was previously connected
Serial.println("Setup done");
Ładowanie zakończone.
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (effective 1170.3 kbit/s).
Hash of data verified.
Leaving...
Hard resetting...
Błędna biblioteka znaleziona w C:\Users\tmax0\Documents\Arduino\hardware\espressif\esp32\
Błędna biblioteka znaleziona w C:\Users\tmax0\Documents\Arduino\libraries\BLE: C:\Users\t
Błędna biblioteka znaleziona w C:\Users\tmax0\Documents\Arduino\hardware\espressif\esp32\
Błędna biblioteka znaleziona w C:\Users\tmax0\Documents\Arduino\libraries\BLE: C:\Users\t
ESP32 Dev Module, QIO, 80MHz, 4MB (32Mb), 115200, None na COM5
8 channels SigmaDelta which uses SigmaDelt
2 channels DAC which gives real analog outp
Installation Instructions
Using Arduino IDE
Instructions for Windows
Instructions for Mac
Instructions for Debian/Ubuntu Linux
Instructions for Fedora
Instructions for openSUSE
Using PlatformIO
Building with make
Using as ESP-IDF component
Decoding exceptions
You can use EspExceptionDecoder to get meaningful call trace.


71% 17:48
Wi-Fi
Wi-Fi Direct
WIĘCEJ
Włączony
UPC5493621
Połączono
MyESP32
UPC Wi-Free
UPC0048151
UPC247294532
UPC507294532
UPC243788310



Basis for the soft AP server



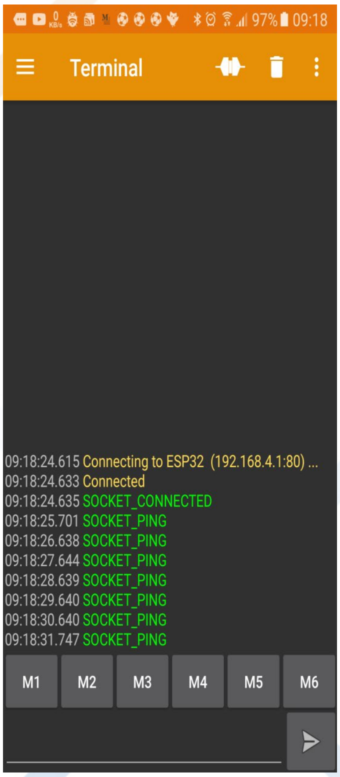
Implicit Arduino uses **softwareSerial** library to establish an UART connection.

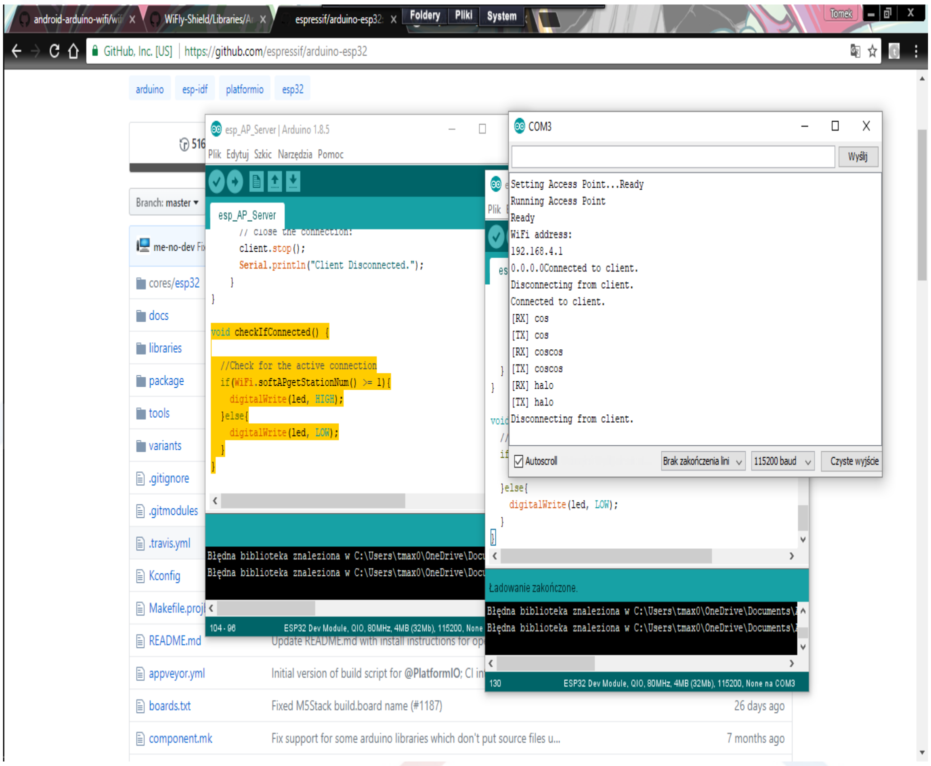
Esp constructors decided to implement faster and more flexible **hardwareSerial.**

Serial data transfer: Every action that is passed through serial ports is a simple text command.

As a first step we had to check our connection and return results.

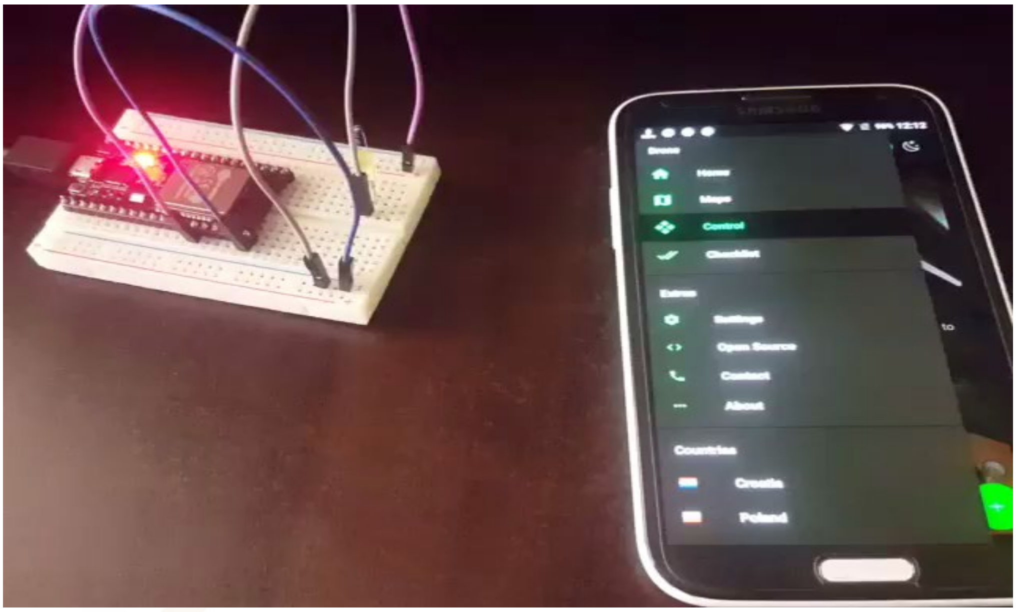
A good way to do that was to write a simple ping interval.





Implementation in a drone app: We had to rewrite all the interfaces to the app's side code.

In the first approach application has automatically found esp server and connected to it.

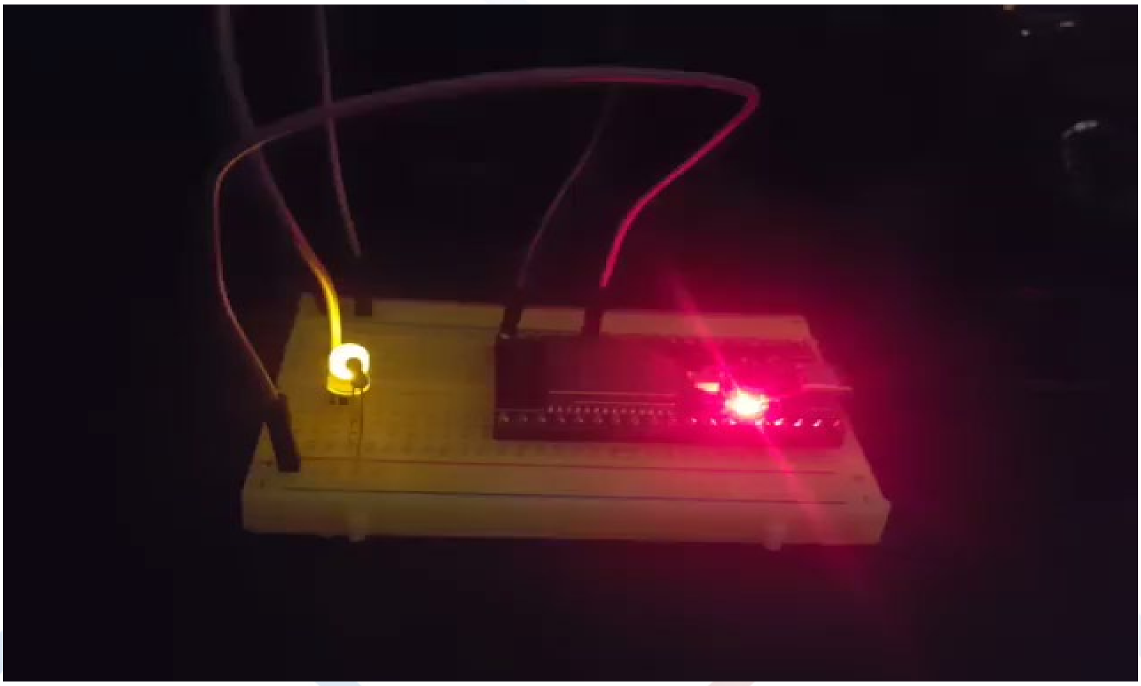


Testing a PWM ports:

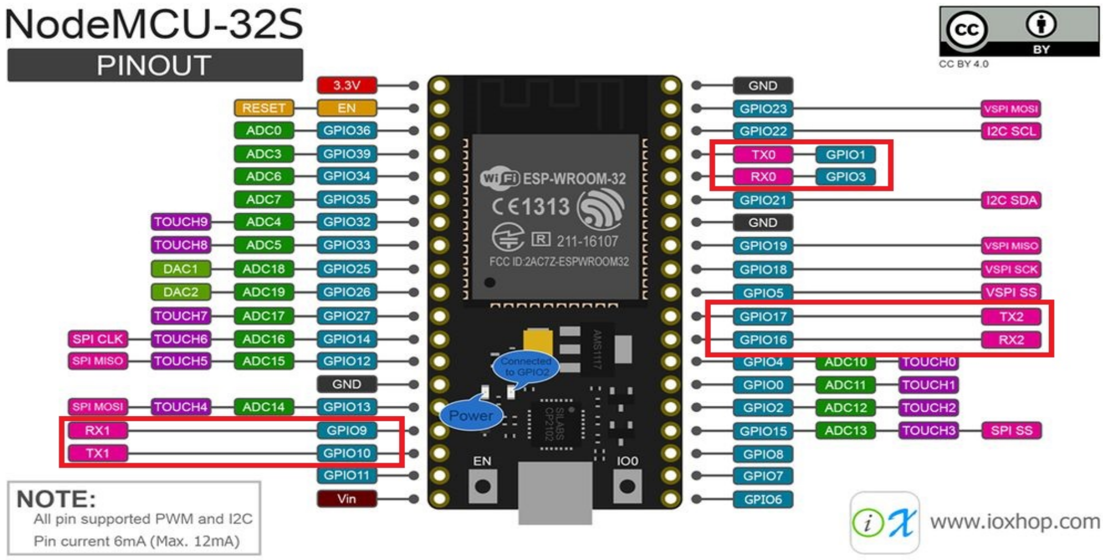
PWM (Pulse Width Modulation) allows to move out modulated signal of value between 0-255.

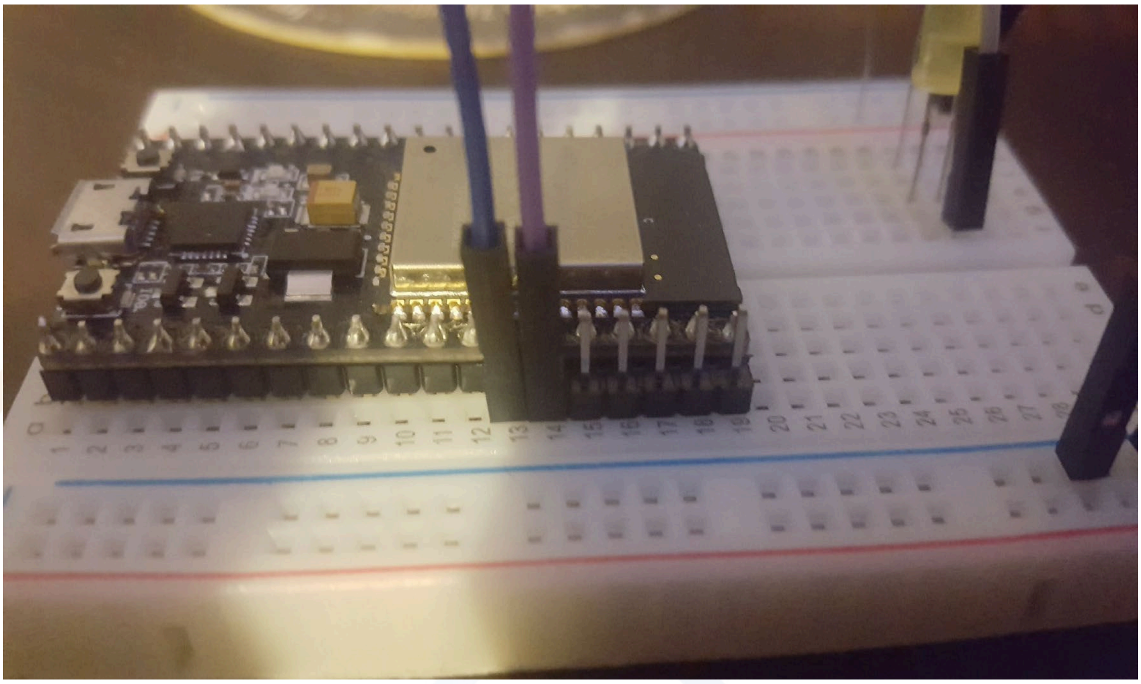
Good way to learn how the PWM works, was to make a common LED example.

PWM modulation is widely used in motor regulation.



Locate RX and TX pins:





Different GPIO's

GPIO1, GPIO3 support serial connection but block other serial ports like usb so it's quite problematic using them.

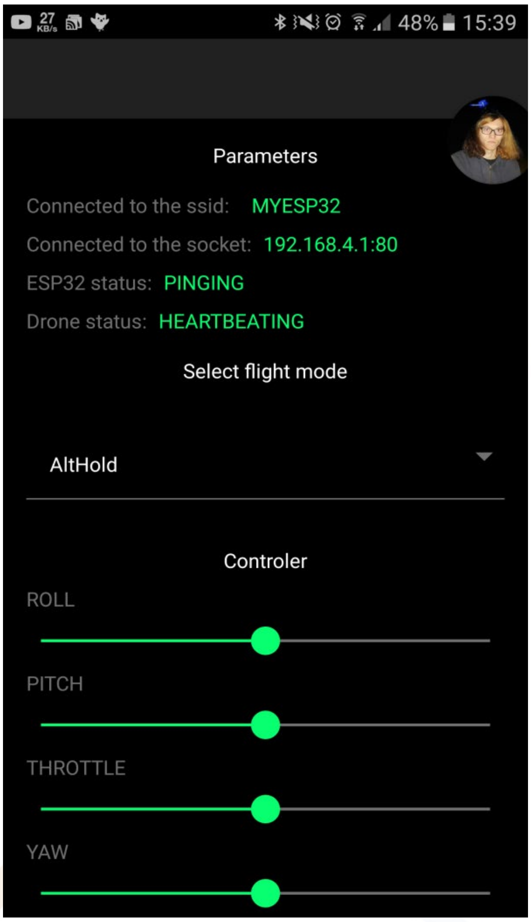
In a next step we decided to change ports to the GPIO16, GPIO17.

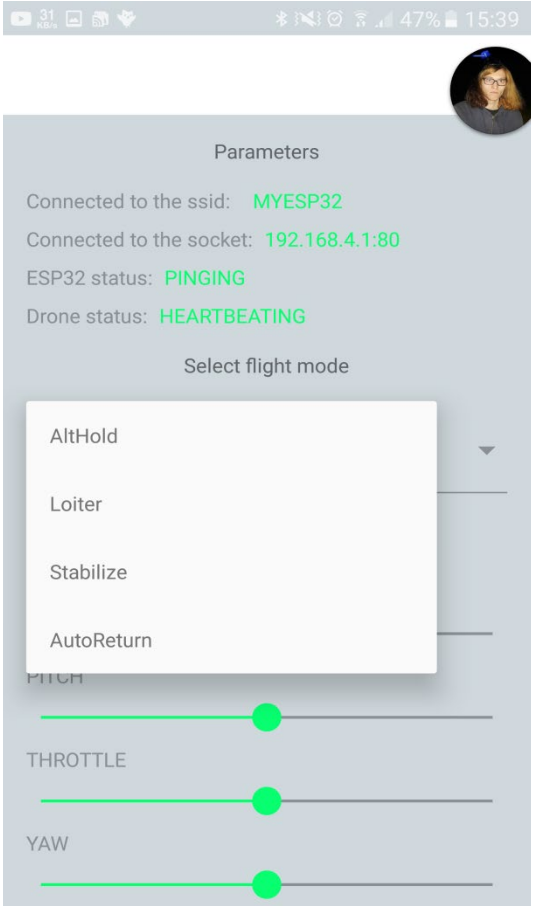
Esp32 serial tricks:

HardwareSerial supports baud rate from 9600 to 115200, where 115200 is the fault value.

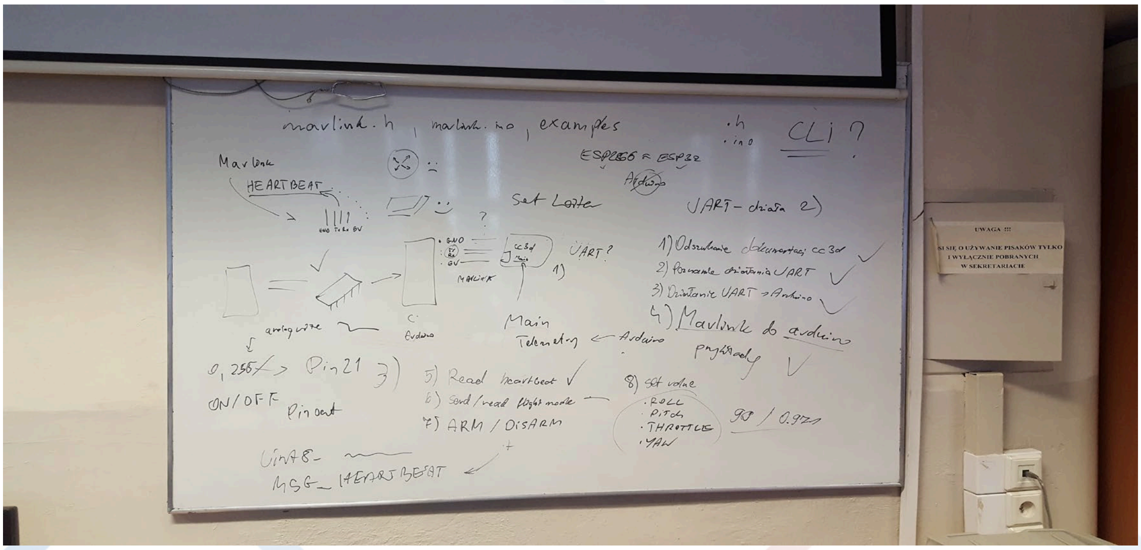
It is possible to run synchronized 57600 **hardwareSerial** for APM telemetry and a Serial 115200 for a app commands.

Creating an GUI in app:





Discussing a proper development:



It is necessary to review Mavlink Documentation and Arduino code.

These sources are crucial in a code development because Mavlink is well documented but for experts.

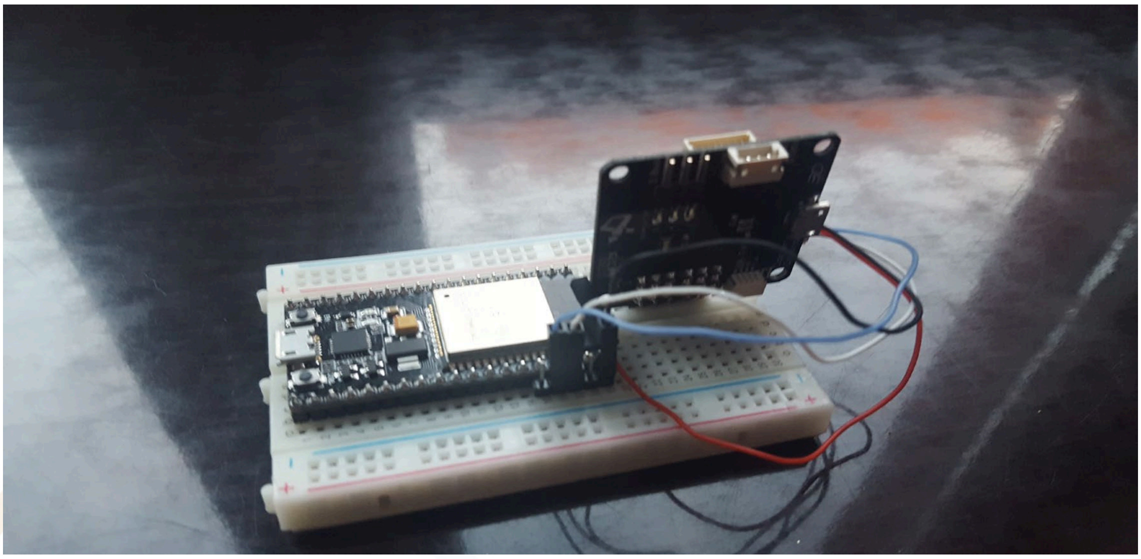
Changing app architecture

Changing the code layout on the M (model) V (view) P (presenter) approach.

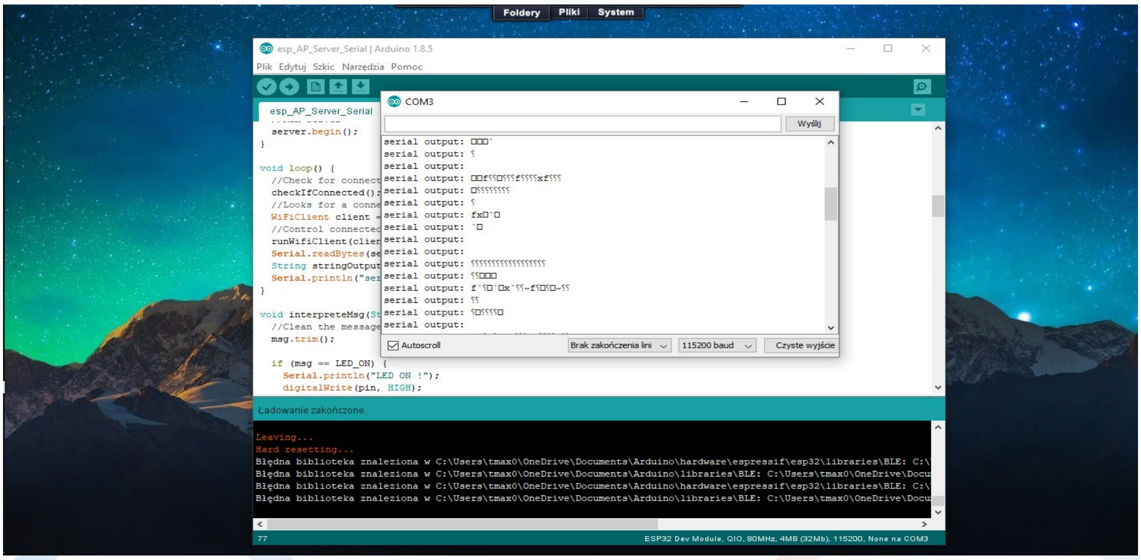
The big amount of code and high complication forced me to separated code sections.

Thus now app is more flexible and easier to testing.

Connection



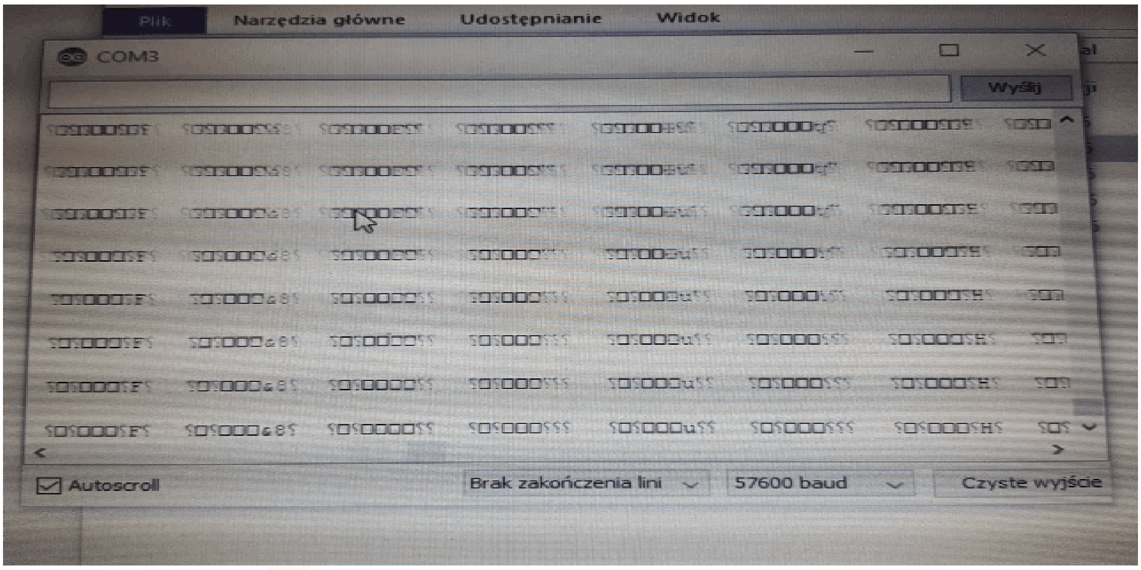
Initiating serial outputs

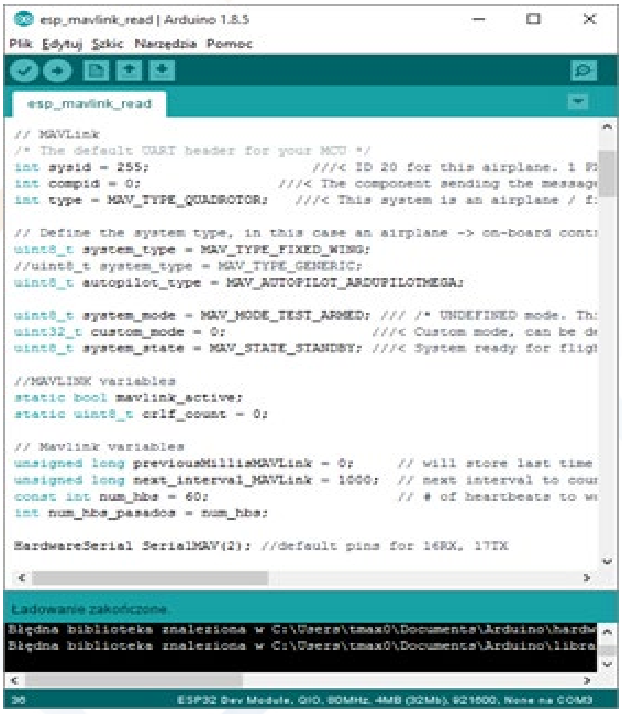


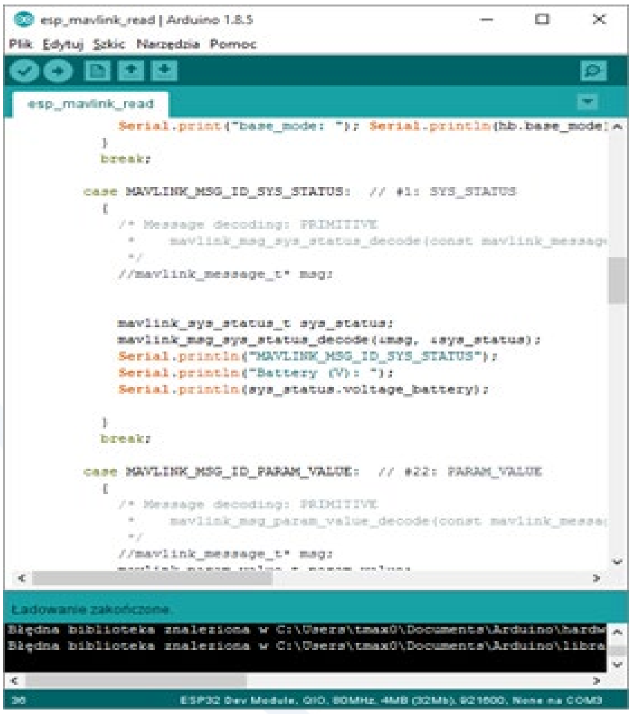
Problems with CC3D:

cc3d does not use mavlink interface and requires software serial

Raw bitframes

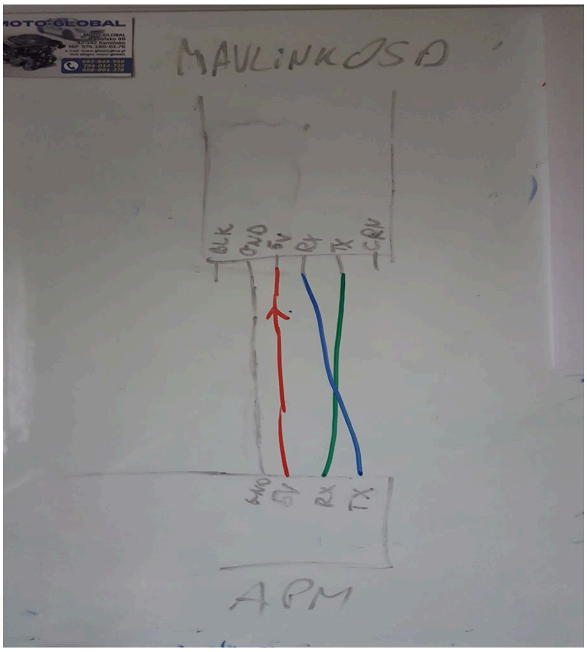






Exchange from CC3D to APM

Changing from basic (CC3D) control to advanced control (APM). DroneTeam used CC3D for basic drone and APM for advanced drone. How to change is explained in following steps:

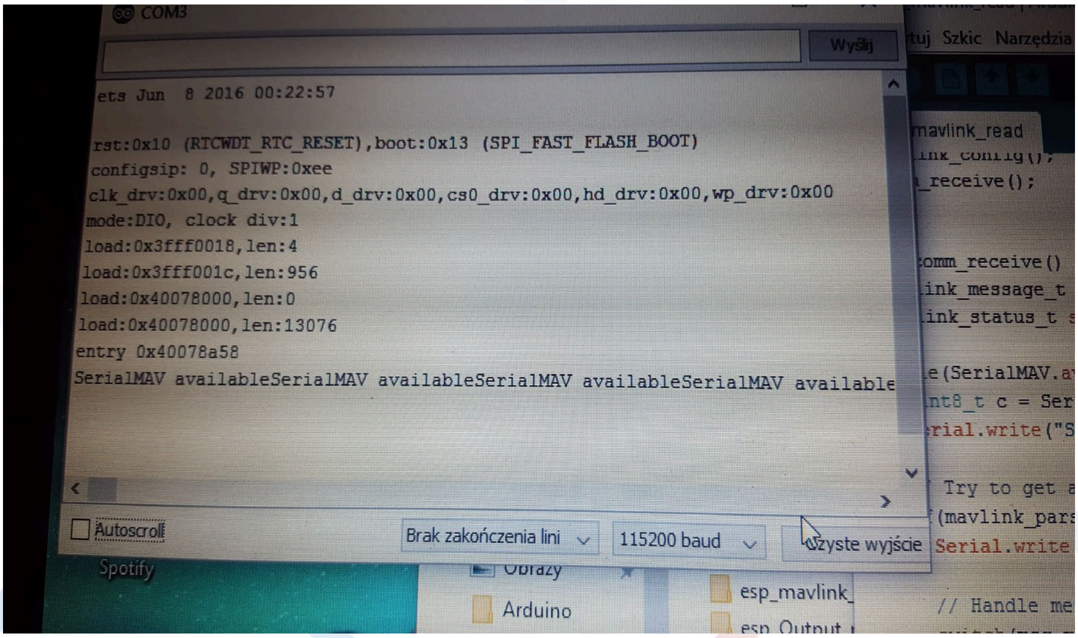


Connexions MavLINK *vs* APM

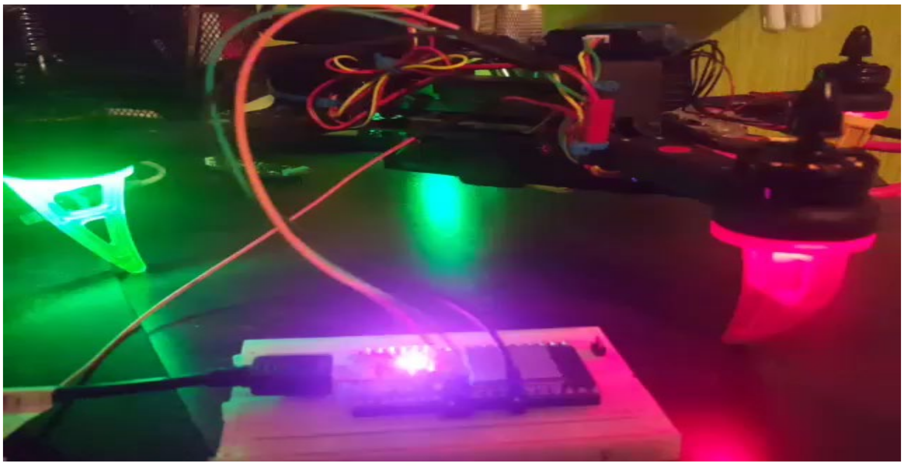
Mavlink and APM

With APM the things had gone match better. APM supports **hardwareSerial** and Mavlink commands.

We were able to use already written mavlink code and check bits' transfer



Indicating serial read:

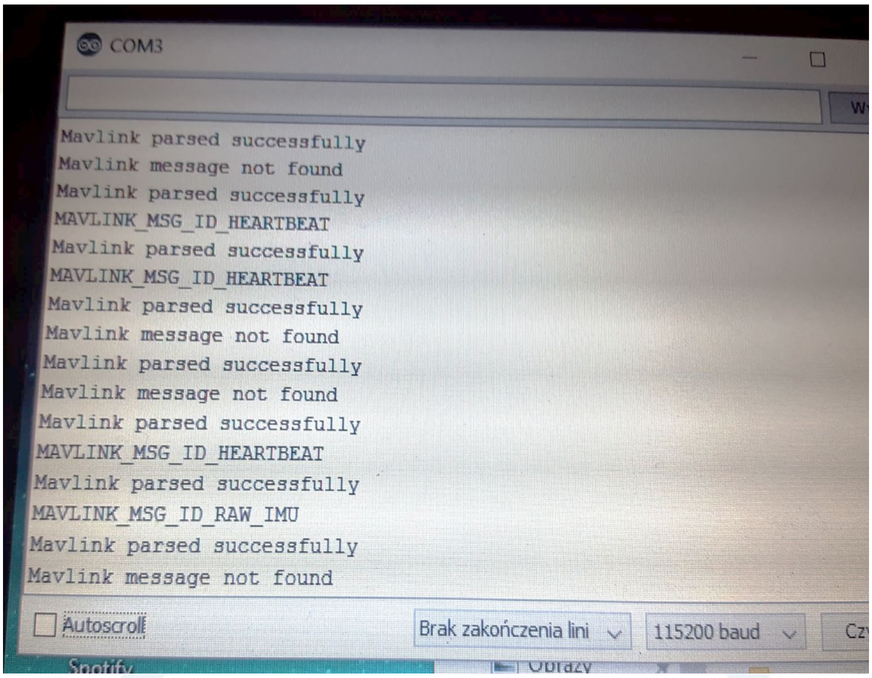


Mavlink explanation:

Drones are using heartbeat to confirm a connection.

Receiving heartbeat packets is first and most important thing.

After that we can decode other drone's parameters.



Sending mavlink requests

After we have learned how to receive mavlink's data we are able to send our commands.

As always the initial thing is to arm the drone.

We have discovered that:

- Base mode= 81 (drone disarmed)
- Base mode=209 (drone armed)

Drone disarmed

Drone armed

Recognition RC channels

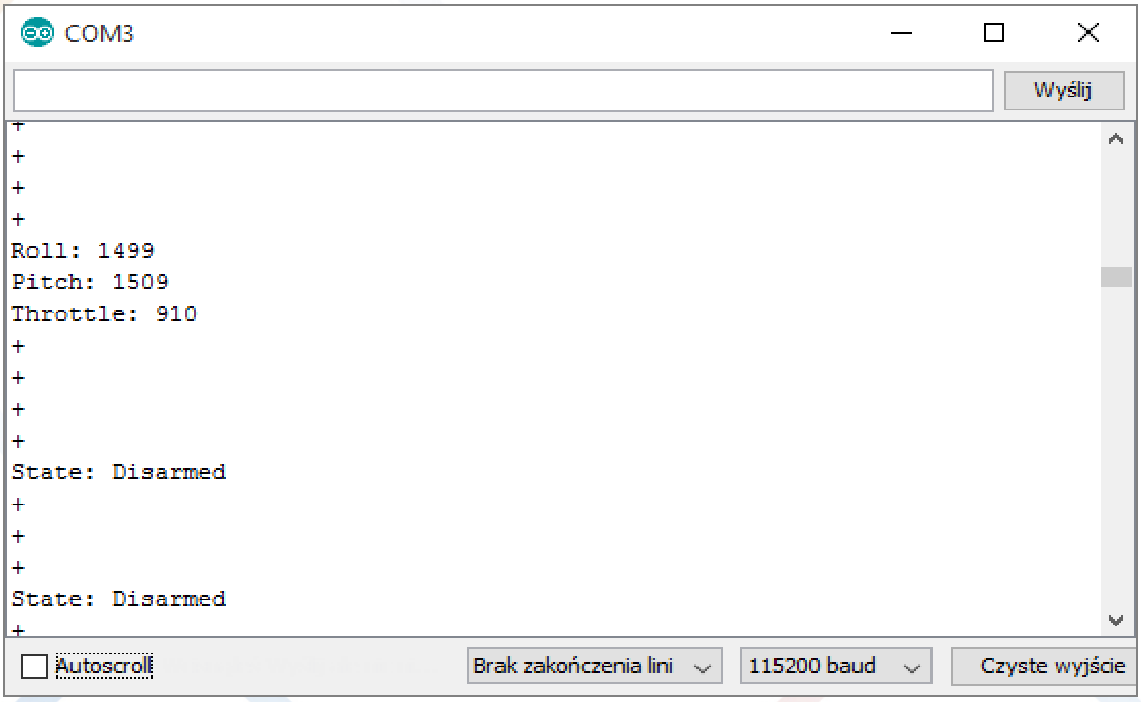
RC (Radio Controller- every equivalent of telemetry).

RC channels correspond to actions that are called on the telemetry e.g. flight mode, pitch, rtl.

We had to check witch channels were for pitch, roll, throttle, yaw.

Channels

- 1- roll
- 2- pitch
- 3- throttle
- 4- yaw
- From CH5- to CH8: free



What command is for motors?

Command DO_SET_SERVO is only used for extra servos like triggers arms.

For setting speed of a motor we have to use

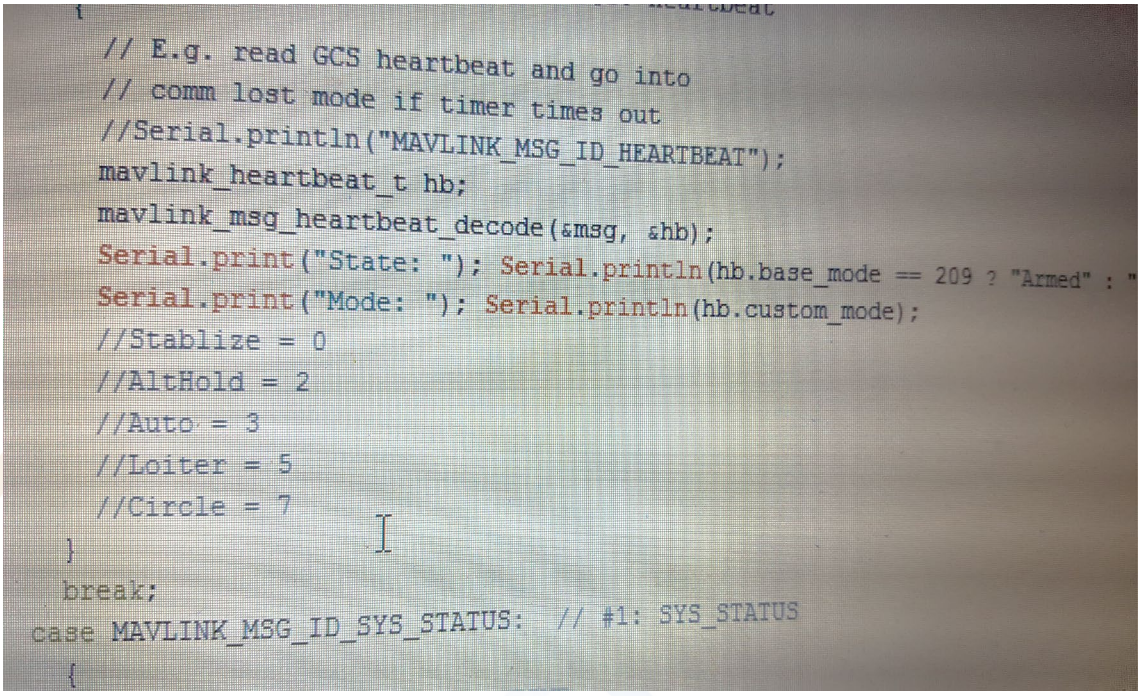
Mavlink_msg_rc_channels_override.

Reading flight modes

In order to check and change current flight mode in Arduino I read custom_mode parameter.

Custom_mode returns number representation of current mode.

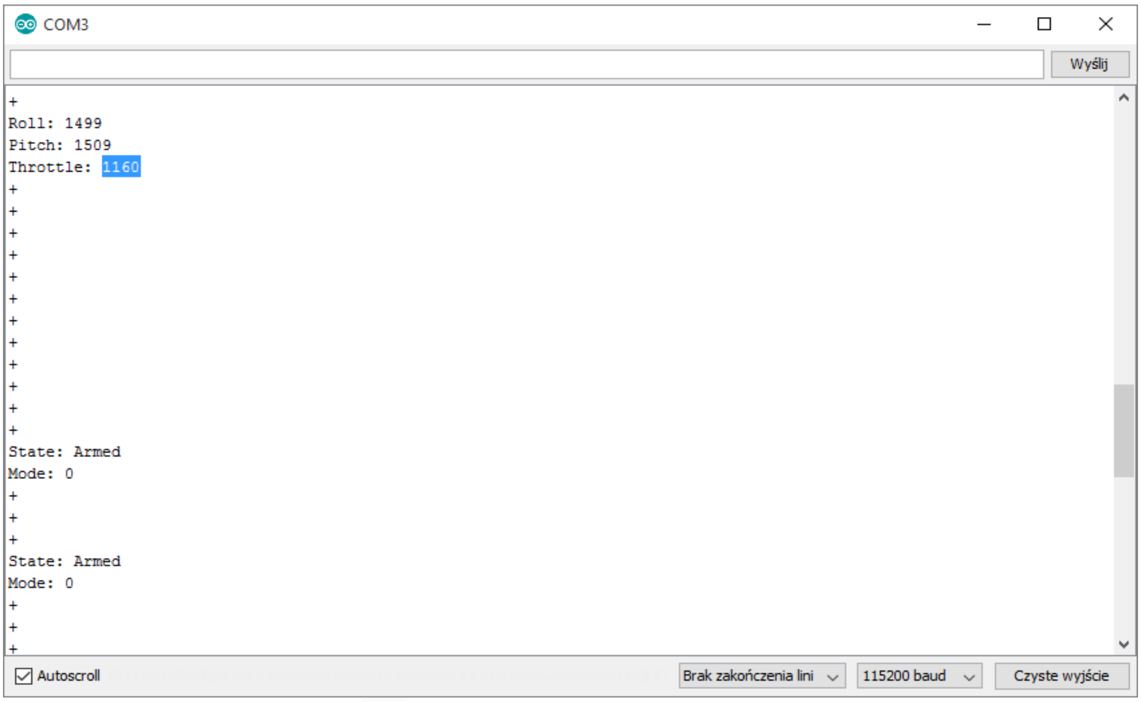
A value that stands for a number can be checked in file common.xml



The biggest breakpoint

Our drone starts running above 1150 value over throttle.

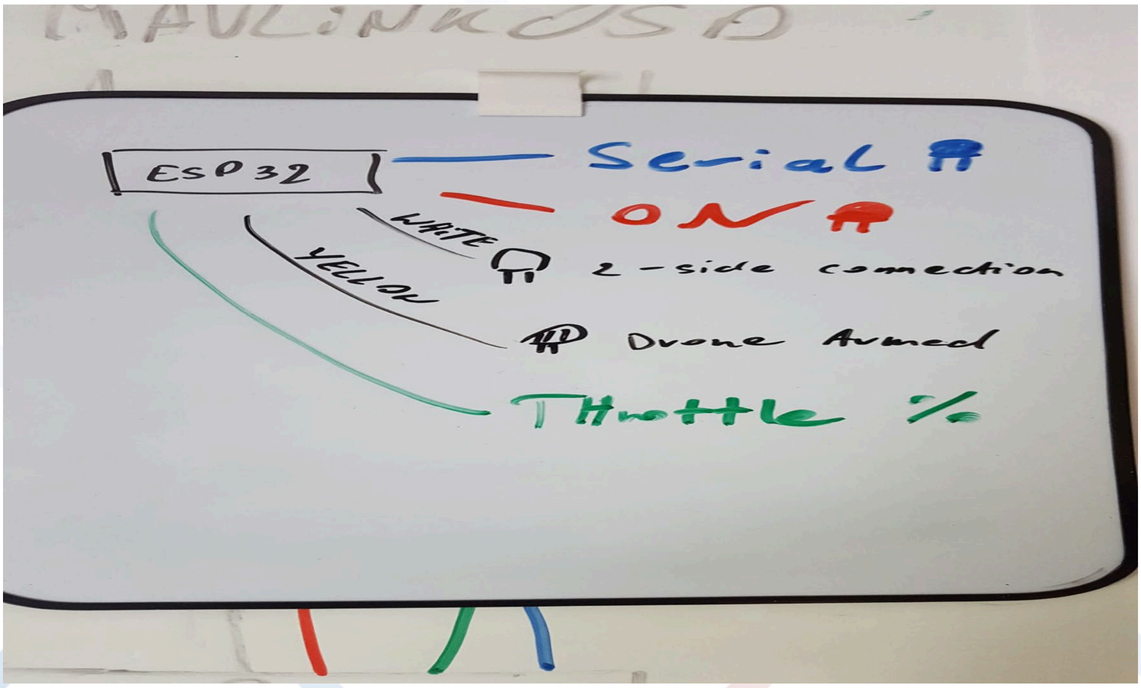
Before that the drone had to be armed.

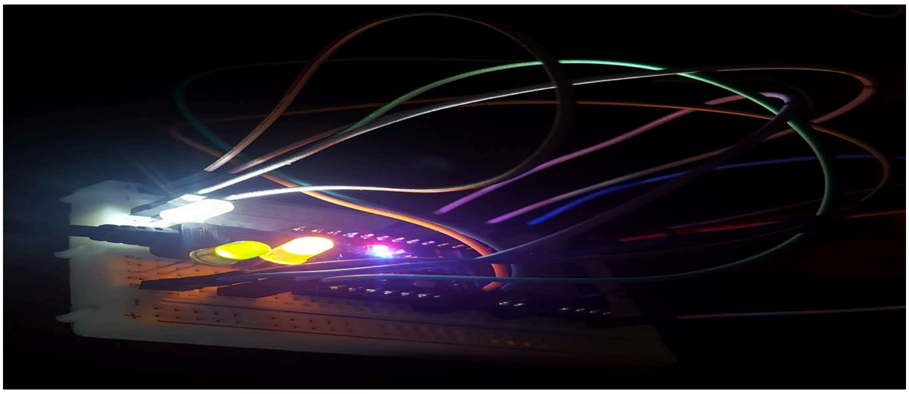


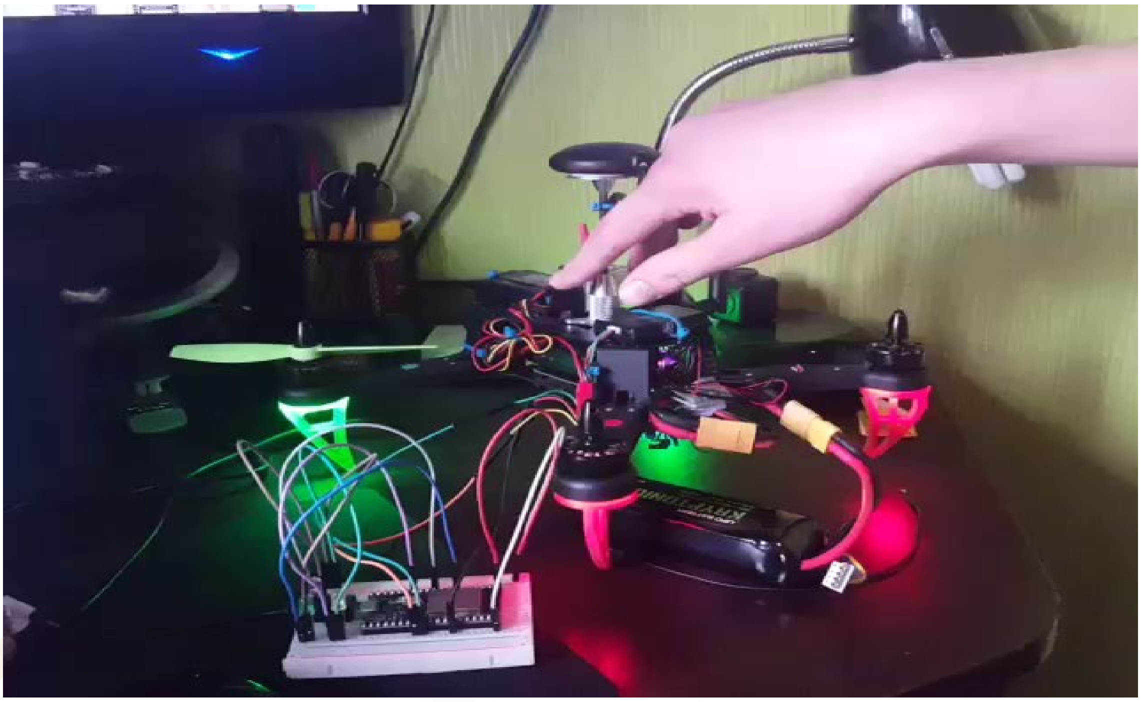
Designing indication system

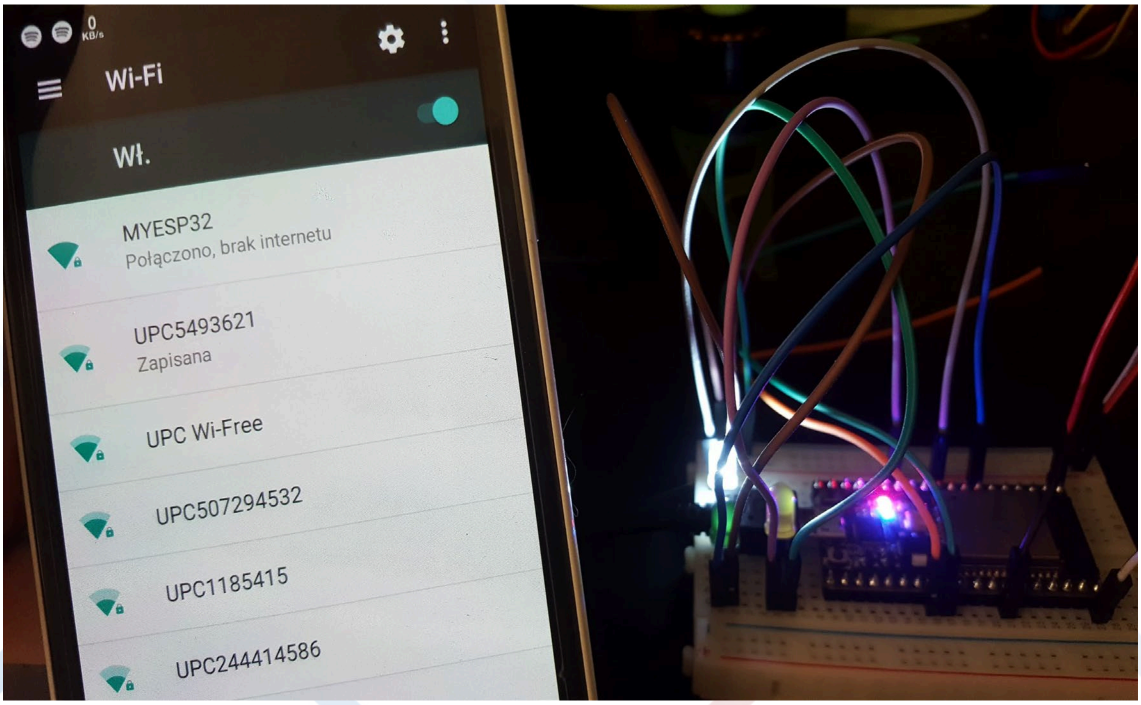
For pointing present state (arm, stabilize, throttle) we use builtin and outers leds.

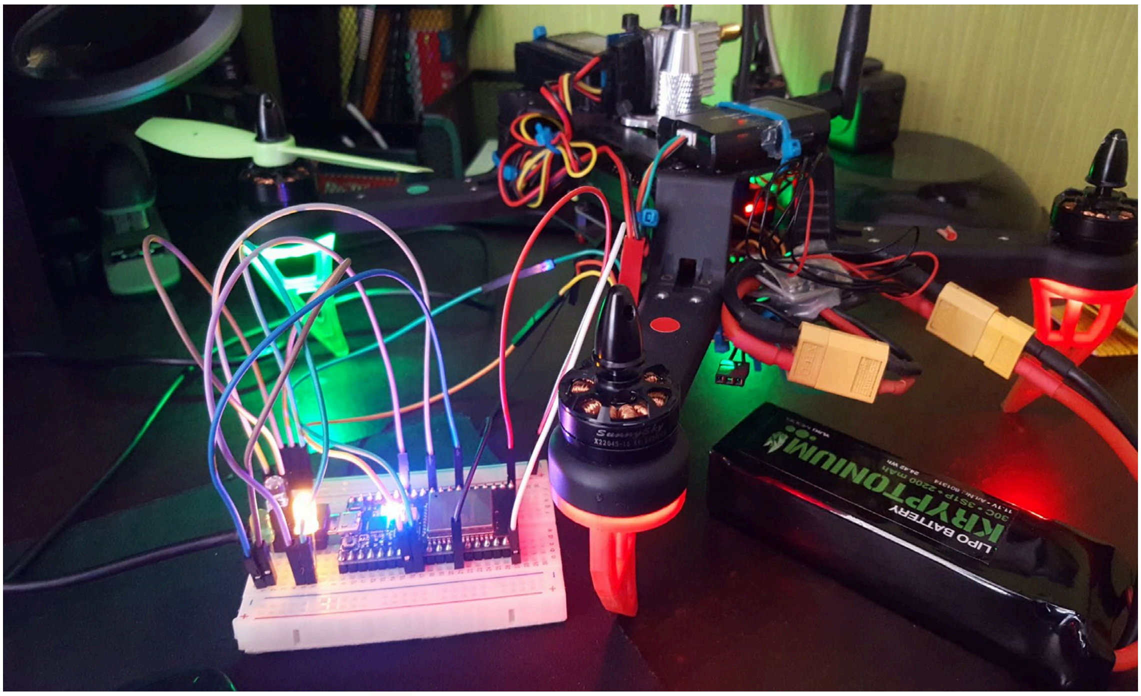
- Blue- serial data read
- Red- state of esp32
- White- connection with a phone
- Yellow- arming state
- Green- throttle in percent



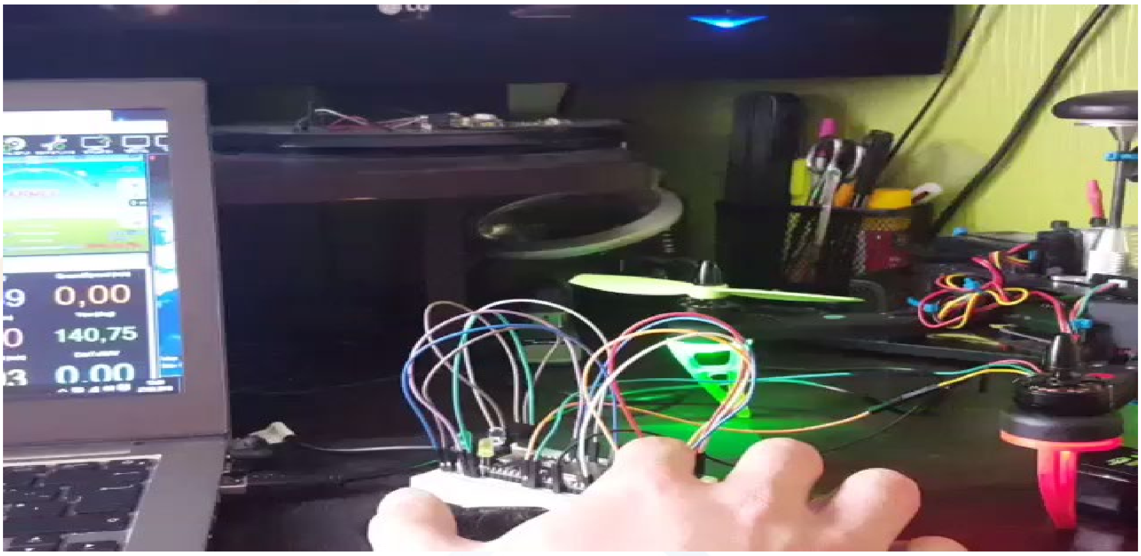




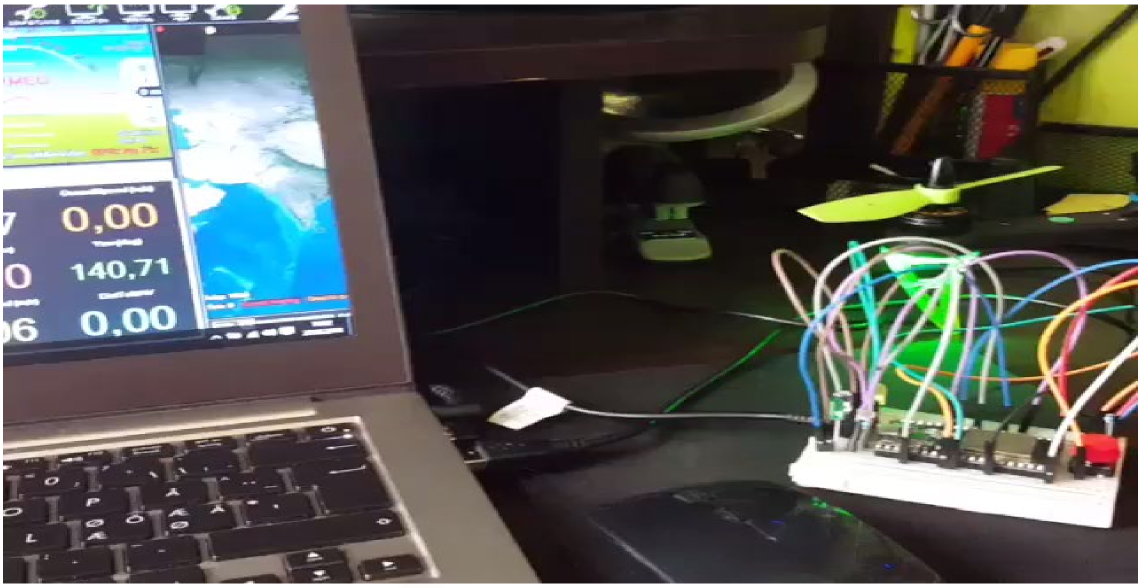

Wi-Fi
Wł.
MYESP32
Połączono, brak internetu
UPC5493621
Zapisana
UPC Wi-Free
UPC507294532
UPC1185415
UPC244414586



Arming with a button



Final esp32 control



Code summary

After working in communication control, the following table summarised the development did in this Project:

Arduino	**Android**
• Created files: 15 • Libraries: 34 • Lines of code: 645	• Created files: 43 • Libraries: 26 • Lines of code: 19 087 (with libraries)

Source code for remote control of drones.

Link to the repository with the application code:

https://github.com/tmaxxdd/DronE

Link to the repository with code for tile electronics ESP32:

https://github.com/tmaxxdd/arduino-with-mavlink