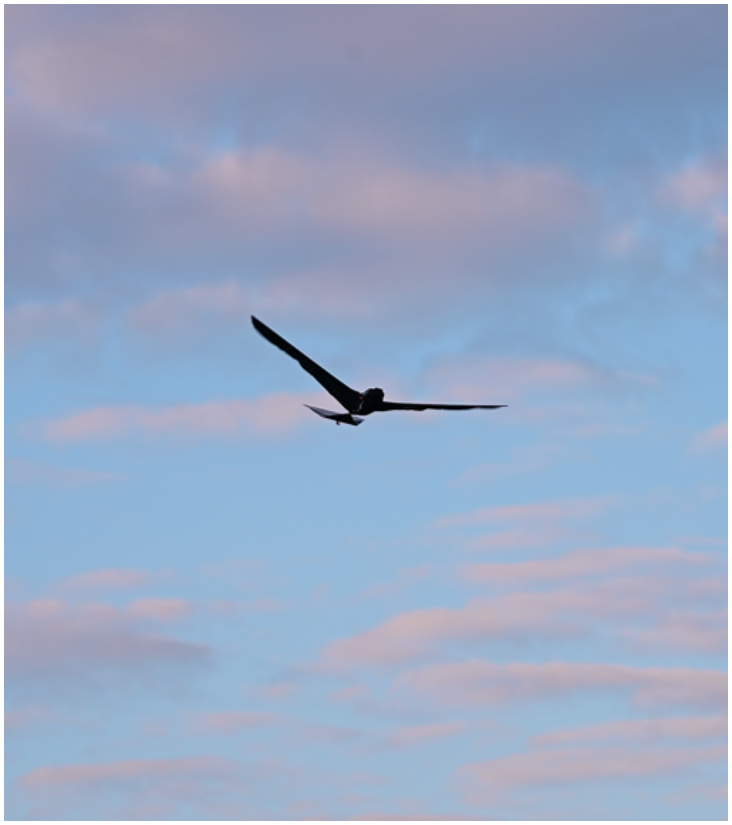
Autopilot Architecture for Micro Flapping-Wing Robots

Laboratory of Intelligent Systems - Biorobotics Laboratory



Stephen Monnet (*327447*)
Supervised by : Raphaël Zufferey
Professor : Dario Floreano

January 6, 2023

Contents

EPFL

Laboratory of Intelligent Systems

SEMESTER PROJECT

Title: Autopilot architecture for micro flapping-wing robots

Student(s): Stephen Monnet (Microengineering)

Professor: Dario Floreano

Assistant 1: Raphaël Zufferey

Project description:

The objective of this project is to study and develop an autopilot architecture for lightweight flapping-wing robots.. As part of the LIS and BIOROB, you will be part of state-of-the-art robotics research labs, working on novel multimodal locomotion systems, giving you exposure to real robots, prototyping and with hands-on experiments.

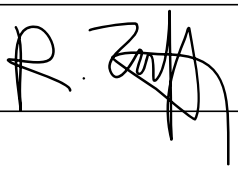
Aerial-aquatic robots need to fly and swim both in lab and outdoor conditions in order to fullfill autonomous sensing missions and animal locomotion studies. For that, a lightweight, low power consumption, fully contained flight/swim controller is required. This project will have 3 stages: 1) Develop the software framework which will run the autopilot 2) Develop and test an oscillation-compensating controller 3) Interface with Bluetooth low energy, long-range radio, sensors. The student is also expected to build a user interface to quickly change settings remotely (i.e. from a smartphone or laptop).

This should be an exciting project for a creative student with an excellent control background and strong interest in prototyping, electronics and willing to tackle issues in many different fields that come with aerial-aquatic research!

Remarks:

You should present a research plan (Gantt chart) to your first assistant before the end of the second week of the project. An intermediate presentation of your project, containing 8 minutes of presentation and 7 minutes of discussion, will be held on November 10, 2022. The goal of this presentation is to briefly summarize the work done so far and discuss a precise plan for the remaining time of the project. Your final report should start by the original project description (this page) followed by a one page summary of your work. This summary (single sided A4), should contain the date, laboratory name, project title and type (semester project or master project) followed by the description of the project and 1 or 2 representative figures. In the report, importance will be given to the description of the experiments and to the obtained results. A preliminary version of your report should be given to your first assistant at the latest 10 days before the final hand-in deadline. A PDF (dated & signed) of your final version should be

Sent to your first assistant and to the administrative assistant of the lab before noon January 6, 2023. Failing this deadline will reflect in the final grade. A 20 minute project defense, including 5 minutes for discussion, will take place January 12, 2023. You will be graded based on your results, report, final defense, and working style. All documents, including the report (source and pdf), summary page and presentations along with the source of your programs should be handed-in as a single compressed file on the day of the final defense at the latest.

Responsible professor:	Responsible assistant:
Signature:  Dario Floreano	Signature:  Raphael Zufferey

Lausanne, 20 September 2022

1 Abstract

When it comes to build an Unmanned Aerial Vehicle (UAV) from scratches, the flight controller must be included in the early stages of design. It is supposed to act as the brain of the robot, handling communications, external devices and controlling the robot according to the radio communications input. While a lot of autopilot are already available on the market, they are generally not suitable for flapping-wings UAVs, multicopter being the most popular type of drones. People who developed flapping-wing drones in the past have generally used custom autopilot, specifically designed for their prototype and therefore not very versatile [1] [2].
This project aims at providing a general autopilot architecture, including *Bluetooth Low Energy* (BLE) and radio communications, useable for flapping-wings UAVs, but also easily adaptable for multicopters and fixed-wings. The final objective being to test this architecture with outdoor flight experiments.
For this purpose, the first feature added to the drone is the BLE communications with a computer, through a Graphical User Interface (GUI). This latter can be used with any type of drone, it just requires to use skeleton-code (which are provided) in the onboard controller. These skeleton-code also include the long range radio communications reading process. The GUI allows users to set and adjust parameters remotely, which might be very convenient during the prototyping phase.

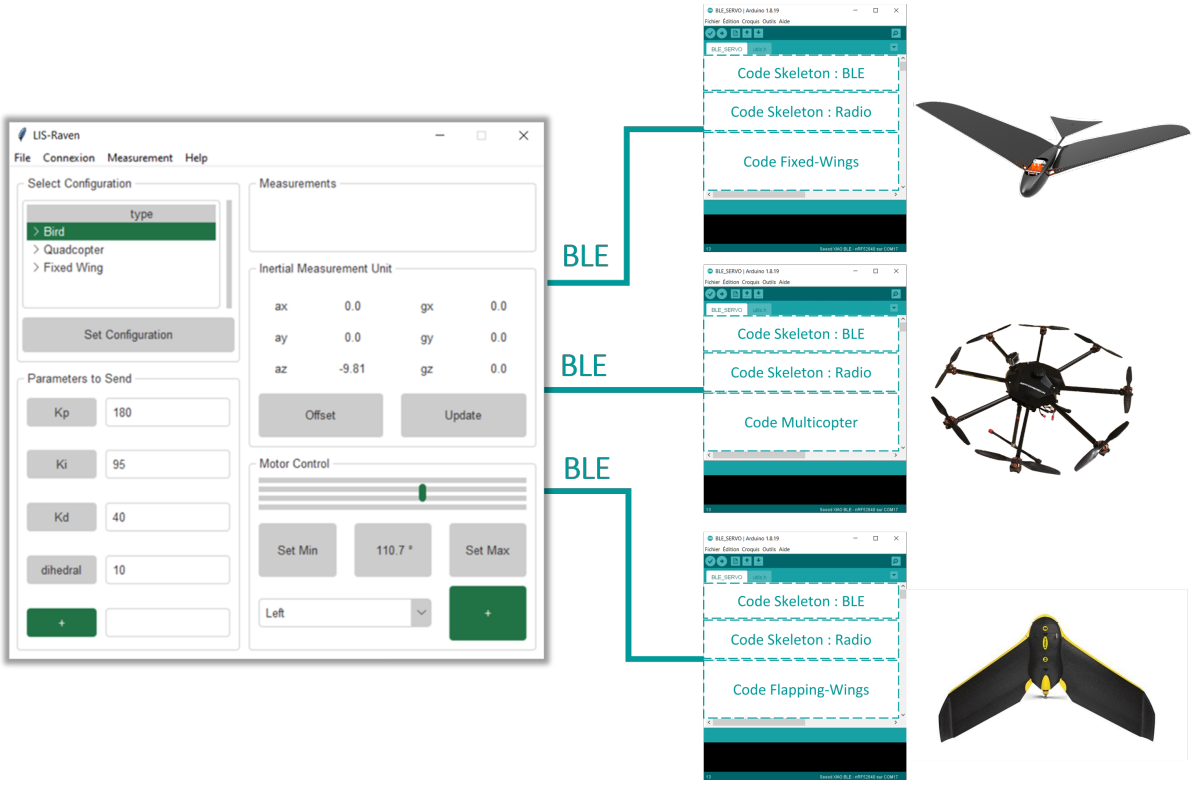


Figure 1: The GUI developed for this project, on the left, communicates with any UAV using the BLE skeleton codes. Here for example, with our flapping-wings, an *Aerodimension*'s *Zeus Octo 8* and a *SenseFly*'s *Ebee*.

For the control of the drone, the current version is sending periodic signals to the wings servo motors. The several parameters of these signals, and the tail's position, are mapped to the remote controller which has been used for the two flight tests. These latter have highlighted the very stable behavior of the drone, even for very low dihedral angles. Nevertheless, it also appeared that the motor were not powerful enough with the current electronic setup to keep the drone in the air.
Further tests must be conducted after modifying the electronic system to increase the mechanical power provided by the wings motors.

2 Introduction

2.1 Context

This project has been conducted jointly at the Laboratory of Intelligent System (LIS) and Biorobotics Laboratory (BioRob), which are part of the Ecole Polytechnique Fédérale De Lausanne (EPFL), during fall semester 2022. It has been supervised by Dr. Raphaël Zufferey and Professor Dario Floreano and is valued at 10 ECTS credits in the Robotics Master.

2.2 Project Overview

Figure 2 shows a 3D rendering of the drone at the beginning of the project. It is a flapping wing, multimodal, drone mainly made of carbon fibers with a wingspan of around 75 cm and a length of 43 cm. The wings are controlled with two Servo motors (2 DOFs, one for each wing) and made of a flexible structure which supports a sail. Its tail is controlled via a Servo motor (1 DOF) and a rod.

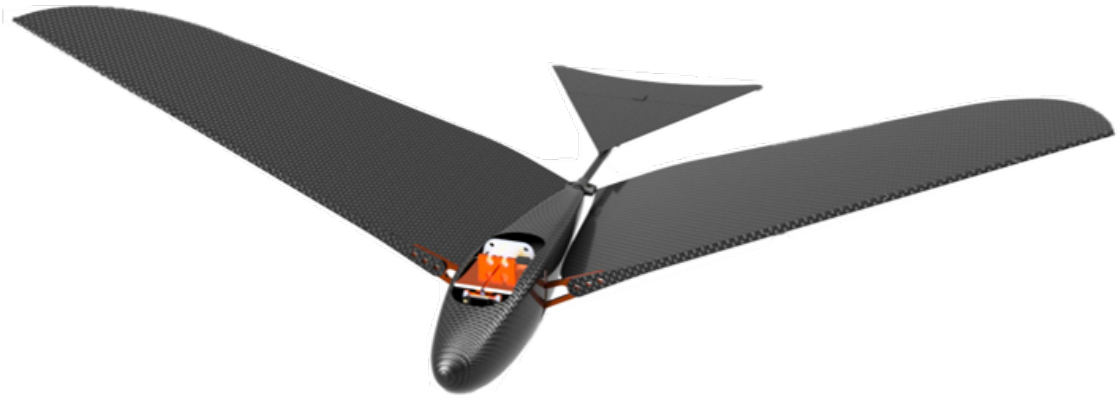


Figure 2: 3D-rendering of the flapping-wing UAV.

An nRF52840-based microcontroller (MCU, see figure 3) is available onboard, with a built-in Inertial Measurement Unit (IMU) and a *Bluetooth Low Energy* (BLE) chip [3]. It can be programmed using the Arduino IDLE (C/C++) which will make it very convenient for the prototyping phase.

Figure 3: nRF52840-based micro-controller used in the drone.

The final (long-term) objective of the drone is to be able to both fly in the air and swim underwater. A typical use-case could be to fly to a specific zone far from the coast to take water measurements and bring it back autonomously.

2.3 Objectives

The main objective of this project is to develop different solutions to control and access to some parameters of the drone. The following list of specifications was given at the beginning of the project :

(i) Develop the software framework that will run the autopilot.

(ii) Develop and test an oscillations-compensating controller.

(iii) Interface the drone with BLE, long-range radio and sensors.

(iv) Build a user interface to quickly change settings remotely from a computer or a smartphone.

To summarize, the final objective is to control the drone using a Remote Controller (RC) that sends commands to an onboard receiver. The commands must then be read by a microcontroller and converted into Pulse Width Modulation (PWM) commands to the Servo while considering data coming from the IMU. Lastly, a graphical user interface (GUI) should be developed to allow the user to communicate with the drone using BLE.

2.4 State of the art

A lot of flight controller are already available on the market, and may include additional features such as camera, *Bluetooth* or *WiFi*. This section aims to provide an overview of the current state of the art in this field, first focusing on the autopilots available on the market, and then on the method used by other flapping wings UAVs.

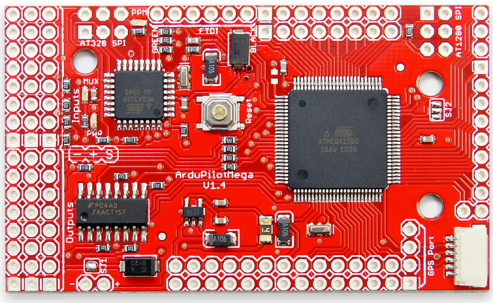
One of the most important company is *DJI*. Their multicopter flight controllers generally provide 8 Electronic Speed Controller (ESC) pins and 8 inputs for the radio receiver (see figure 4) as well as camera connectors and onboard IMU and GPS chip. The user simply needs to connect and set correctly the flight controller using an application to make his drone fly. Nevertheless, it is only able to control multicopter with Brushless DC (BDC) motors and it is not very compact (57.9mm x 39mm x 17mm, 132gr) [4]. Servo motors input may be available on such flight controller, such as the *Pixhawk*'s 4, but only to control an onboard camera [5].
Pixhawk provides also flight controllers (open source) for racing drones. The *mRo Pixracer* is designed for racing quadcopters and fixed wings, and includes a *WiFi* module for both flashing new firmware and system setup [6]. It is lightweight (10.54gr) and indeed more compact than industrial flight controllers (36mm x 36mm). Nevertheless, it still can only handle BDC motors.

Figure 4: N3 flight controller from *DJI*.

Other compact flight controller for multicopter exist such as *Flyduino*'s *KISS* serie [7] or *SpeedyBee*'s F7 AIO [8], but none of them provides a way to control both servos and brushless DC motors.
For fixed wings UAVs, very compact flight controller also exist such as the *Matek*'s F411-WSE with 2 BDC output and 4 servo outputs for a 28mm x 14 mm chip [9]. Again, the servo output are not designed to provide high-current and therefore not useable to actuate wings such as in our case. Furthermore, it does not provide any wireless communication except the radio frequency module.

For flapping-wing UAVs, no "plug-and-play" commercial flight controllers are available since it is much more difficult to control and less common. One of the most famous bird-scaled flapping-wing UAV is the *Robird* and uses an ArduPilot board to control its BDC motors that generate the flapping motion and its servo motor used for the tail [1].

(a) Robird drone



(b) ArduPilot board

Figure 5: Robird drone with its board running with a Pixhawk autopilot system.

Another flapping-wings drone in the same scale is *Festo*'s *SmartBird* which uses a custom flight controller on an LM3S811-based MCU [2]. These two examples are representative of most of the flapping-wings UAVs until today. The controller is generally programmed from scratches, with only radio frequency to communicate.

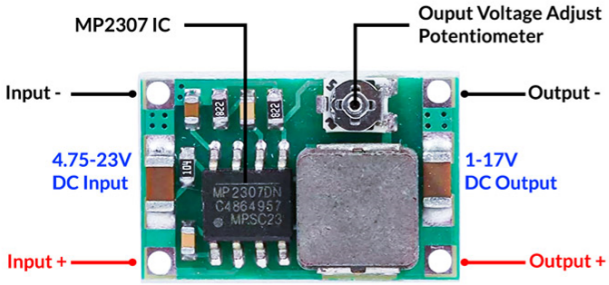
3 Electronic Setup

The electronic setup is designed to power three servo motors, the radio-frequency receiver, and the microcontroller from an onboard battery. This chapter presents the different components of the setup and provides the complete wiring scheme.

3.1 Components

The whole setup is powered by a 2-cells Lithium battery with a capacity of 500 mAh (see figure 6a).

(a) 2S Battery with 500 mAh capacity.



(b) Adjustable buck converter based on an MP2307 integrated circuit.

Figure 6: Power supply module of the drone.

It is directly connected to an adjustable voltage regulator (see figure 6b) which can lower voltages of 4.75-23 V into 1-17 V [10]. In our case, the output of the voltage regulator is set at 5 V since it is in the working voltage range of all the onboard components. According to the datasheet, it has an efficiency of around 90% for an input voltage of 7.4 V.
For the actuation of the drone, three motors are used. The two motors actuating the wings are the same (see figure 7b) while a smaller one (see figure 7a) is used to actuate the tail. The delivered torque of the wing's motors can reach 10 $kg \cdot cm$ at 5 V while the tail's motor can reach 2 $kg \cdot cm$ at 5 V [11] [12].

(a) BMS-A10V

(b) BMS-A921+

Figure 7: Servo motor for the tail actuation (7a) and wings actuation (7b)

The receiver (see figure 8) is powered by the voltage regulator and connected to the microcontroller (four channels) to transmit the data coming from the remote controller (see section 4 for details). It consumes

approximately 100 mA at 5 V, that is a power consumption of 0.5 W [13].

(a) R9 Slim + OTA receiver (b) Receiver connections

Figure 8: Long range receiver R9 Slim + OTA from FrSky. It provides a serial port (SBUS or inverted SBUS on CON1) as well as PWM signals (CON2) to read its data. The receiver is designed with dual antennas to improve the signal strength.

Finally, the microcontroller is a nRF85240 to which is added a BLE module as well as an IMU. The whole chip (XIAO nRF85240 BLE Sense) is provided by *Seeed Studio* and provides several analog and digital pins as well as interface for standard communication protocols (see figure 9). Its consumption is negligible (around 0.6 mW with *Bluetooth* communications enabled [14]) compared to the motors.

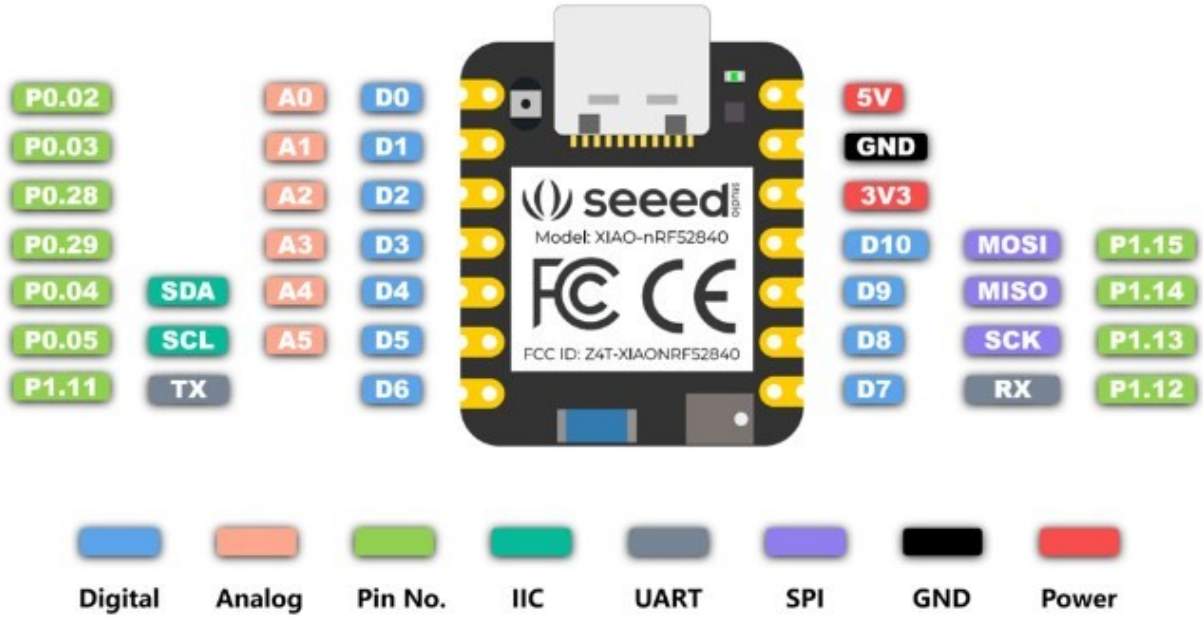


Figure 9: Pins of the nRF52840-based MCU used in our setup.

3.2 Wiring Scheme

Figure 10 shows the complete wiring scheme of the drone used during the flight test (see section 8).

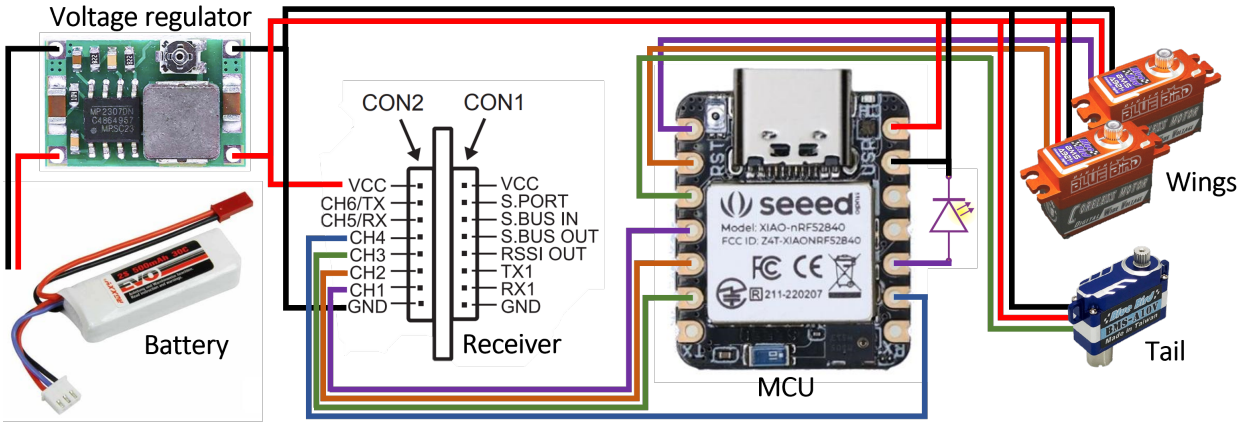


Figure 10: The voltage regulator transforms the 7.4 V into 5 V and provides it to the receiver, the MCU and the motors. Four of the receiver's output channels are directly connected to the MCU. The three signal pins (servo motors) are also directly connected to this latter. An LED is available on pin D9 and can be programmed to provide some information to the user. Note that pins D6 (Tx) and D7 (Rx) are left unused to allow serial communication with external devices.

3.3 Potential Improvements

One potential improvement would be to increase the motors supply voltage, which is currently at 5 V. This would allow the servo motors (especially those used for the wings) to deliver more mechanical power.
Other electronics improvement are possible about the receiver reading process, but are exposed in section 4.4 since they are more related to the radio-frequency communications setup.

4 Radio Communications

The objective of implementing radio communications between the drone and a remote controller is to allow the user to control the drone during a flight. This chapter first presents our setup, explains the radio-frequency communication settings and the binding procedure, then shows how the receiver transmits data to the MCU, and finally propose some improvements.

4.1 Setup

The setup consists of the following elements :

- Remote controller : Taranis QX7 ACCESS
- Long-range emitter : FrSky RM2019 ACCESS 868MHz
- Long-range receiver : FrSky R9 Slim + OTA ACCESS 868MHz

The emitter is directly plugged into the remote controller as an external communication module while the receiver is in the drone. A long-range setup was required to enable better communications when the drone is underwater. Indeed, longer wavelengths ($f = 868\ MHz$ in our case) will have a better penetration in the water.

Figure 11: Long-range receiver (left) communicating with the long-range receiver (right) connected to the remote controller (middle).

4.2 Receiver & Transmitter Communications

The emitter and receiver communicate using the so-called ACCESS protocol, which is developed by FrSky [15]. To setup the communications, one must first verify that all the firmware are up-to-date and compatible with hardware. Then, the binding procedure recommended by *FrSky* must be followed to link the receiver with the emitter. Finally, the available channels must be parameterized in the remote controller.

4.2.1 Firmwares

For this project, the following firmwares[1] have been used :

- Taranis QX7 ACCESS : *FW-X7_X7S_ACCESS-210208/firmware_x7access_en_210208.bin*
- FrSky RM2019 ACCESS : *FW-R9M2019_ACCESS_V1.3.0/R9M2019_LBT.frk*
- FrSky R9M Slim + OTA ACCESS : *FW-R9Slim+OTA-ACCESS_v1.3.2/R9Slim+_OTA_LBT.frsk*

[1] All the firmware used for this project can be found on the *FrSky* website : https://www.frsky-rc.com/download/

4.2.2 Binding Procedure

First, some informations about the receiver must be provided to the remote controller. To do so, in the Taranis QX7 go to the second page (**SETUP**) and set the variable **Internal RF** → **Mode** to **OFF**. Then, in the **External RF** menu, set the variables as follow (see figure 12):

- **Mode : R9M ACCESS**
- **Ch.Range : CH1-16**
- **RxNum : 00**
- **Failsafe : Hold**

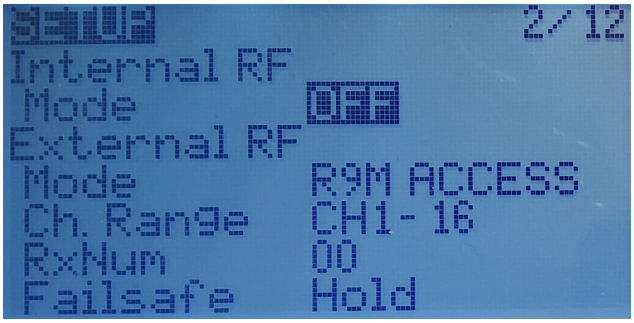


Figure 12: Settings of the internal RF emitter before binding.

After this step, one should be able to bind the receiver with the emitter using the procedure recommended by FrSky [13]. In our case, that is :

(i) Put the transmitter module into [Reg] status. To do so, in the **External RF** menu, click on **Module** → **[Reg]**. The window shown in figure 13 will appear.

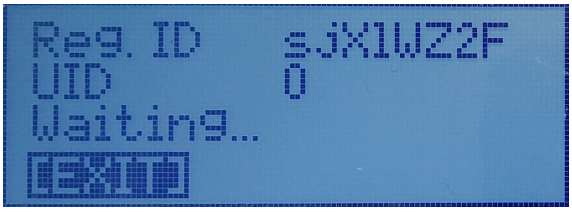


Figure 13: Window shown by the remote controller during binding process.

(ii) Turn on the receiver while holding its F/S button (see figure 14). The red and green LEDs on the receiver will be on, indicating into the [Reg] status.

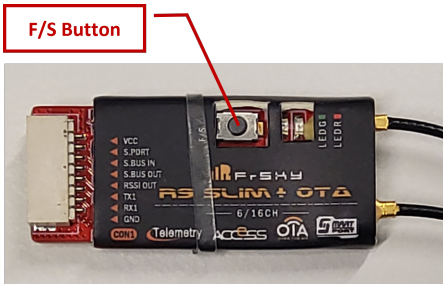


Figure 14: R9 Slim + OTA receiver and its Failsafe (F/S) button.

(iii) Select [ENTER] on the transmitter, the two LEDs will flash and the transmitter displays [Registration OK].

(iv) Turn off the receiver.

(v) At the bottom of the **External RF** menu, click on [Bnd] for one of the receiver (**Receiver1-3**). Up to three receivers can be bind with the same emitter, it doesn't matter which one you choose.

(vi) Turn on the receiver, its green LED will be on. On the remote controller, select the RX and the transmitter will display [Bind successful]. At the end of this process, you should get a result similar to figure 15

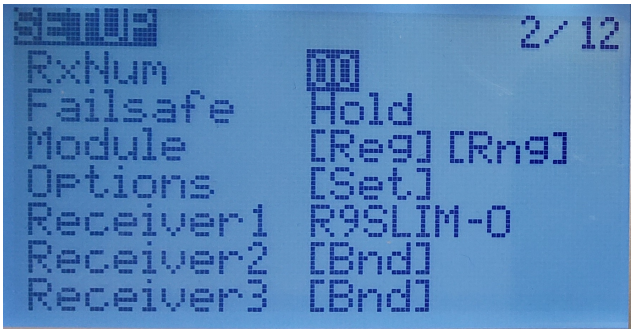


Figure 15: Screen of the remote controller after binding process. In this case, we have chosen **Receiver1**.

After this procedure, the remote controller and the receiver will automatically detect each other when they are both turned on.

4.2.3 Channels Setting

This step consists of linking each channel of the RF communication to one of the potentiometer, joystick or button of the RC. To do so, in the Taranis QX7 go to the fifth page (**INPUTS**). It shows a list of 32 possible channels to which you can link any input of the RC.

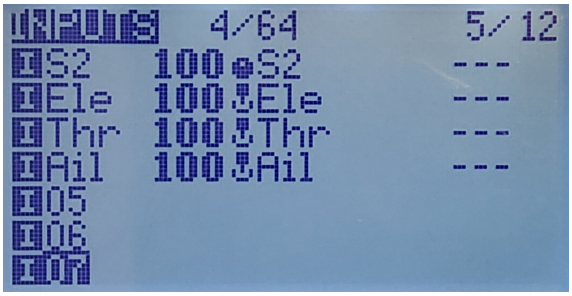


Figure 16: List of available channels to link with any input of the RC. In this case, only the fourth first channels have been linked to some inputs.

To edit an already existing link (such as those displayed on figure 16), click and maintain on the desired one and select **Edit**. To create a new link, click and maintain on the desired channel. The screen will display, among other things, the current input and a graph $y = f(x)$ showing which value y is returned depending on the position x of the input (see figure 17). This function can be linear, in which case it is parameterized by **Weight** (slope) and **Offset**. One can add a non-linear term using the **Expo** variable.
The most important variable is **Source** which selects which input of the RC is used for this channel.

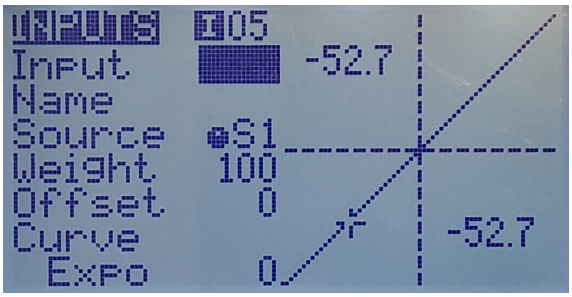
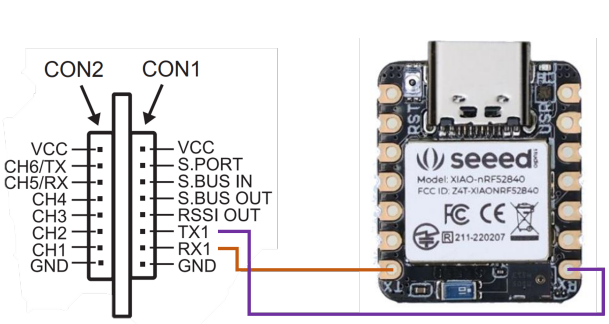


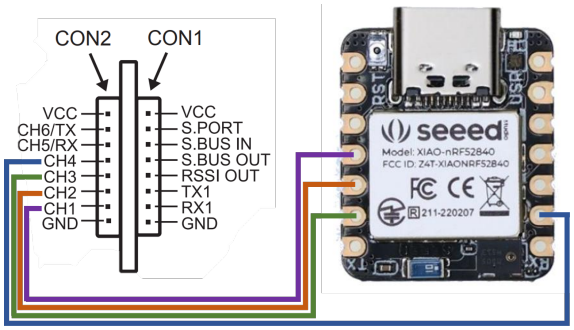
Figure 17: This view allows to setup a channel and link it with one of the RC input. On the right, a graph $y = f(x)$ shows the returned value depending on the position of the input.

4.3 Communications with the MCU

The R9 Slim receiver offers two main output signal types. The first one is a PWM signal, for each of the channels, that can directly be plugged to a motor controller. The other possibility is to use the so-called SBUS protocol (serial), developed by FrSky. While using PWM will limit the number of channels to 6 (this limit comes from the receiver), the SBUS protocol allows to read up to 16 channels.



(a) Wiring using inverted SBUS (CON1).



(b) Wiring using PWM (CON2).

Figure 18: Communication between the MCU and the receiver. Two setup are possible, using serial or PWM communications. While the serial allows to read more channels faster and with less connections, it never worked. Therefore the PWM are used (CON2).

Unfortunately, reading values with our microcontroller using the SBUS protocol never worked. The main reason is the lack of well-established libraries for nRF85240-based microcontroller [16]. Since developing a custom library for our setup may take too long, we decided to use the PWM signals.
The main drawback of this solution is that we need one pin of the microcontroller for each channel.
The reading of the PWM signals is done with the *pulseIn* function from Arduino. When called, this function measures the interval between the rising and falling edge of a signal (in μs) [17] . In our case, the pulse width may vary between 10 and 20 ms. Therefore, this function will return values comprised between approximately 10'000 and 20'000 μs.
In practice, this measurement method limits the sampling time. With four channels, it gives a maximum sampling frequency of $10^3/(4 \cdot 20) = 12.5Hz$.

4.4 Potential Improvements

The main potential improvement is about the reading of the recevier's channels. Currently, four channels from the receiver are directly connected to the MCU, while 6 are available. This will limit the number of inputs that can be sent from the remote controller to the UAV. Furthermore, reading PWM is slow since it requires to wait for a whole pulse. Therefore, adding channels will also significantly extend the sampling time.

- **Multiplexer to read receiver signal :** To be able to read more channels, one could use a multiplexer such as the CD4051BE which provides 8 output using 4 inputs (3-bits address and signal) [18]. This setup has been successfully tested but, while it increases the number of available channels, it also slows down the reading process.
- **Low-pass filter to read receiver signal :** Another possibility to read values from the receiver would be to use a low-pass filter along with an amplification circuit. Since the information lies in the duty cycle of the PWM signals, a low pass filter can be used to transform this information into a DC voltage. To increase the sensitivity of the circuit, a differential amplifier circuit could also be added at the output of the filter.
- **SBUS library for nRF52840-based MCU :** Finally, the best but also most time-expensive solution would be to design an SBUS library for our MCU, based on existing library such as *bolderflight* [19].

5 Bluetooth Communications

The objective of establishing a *Bluetooth* communication between the drone and the user is to provide an easy way to adjust some parameters of the drone and monitor its states before a flight. This chapter first presents our *Bluetooth* setup, then explains how the BLE protocol works and is used, and finally propose some possible improvements.

5.1 Setup

On the MCU, the *ArduinoBLE* library (C/C++) is used while the *Bleak* library (Python) is used within the GUI on the computer. These libraries enable the use of the BLE protocol, especially interesting for embedded systems [20] [21]. This communication type is possible thanks to the onboard BLE chip of the *Seeeduino* board. This is the main reason for choosing this board, which is one of the smallest one providing this feature.

5.2 Overview of the BLE Protocol

The *Bluetooth Low Energy* (BLE) protocol is based on the standard *Bluetooth* protocol [22]. The main difference is that BLE devices are able to go into sleep mode between communications [23]. While the speed of information transmission (1 Mbit/s) is as fast as for the *Bluetooth* protocol, it uses ten times less power (10 mW).
In practical terms, messages between peripherals are handled with so-called BLE *services* and *characteristics*. When a communication is established between two peripherals, a *service* is associated to it. The main information contained within it is the *Universally Unique Identifier* (UUID) which is a 128-bits address given to each of the peripheral such that they can identify each other [24].

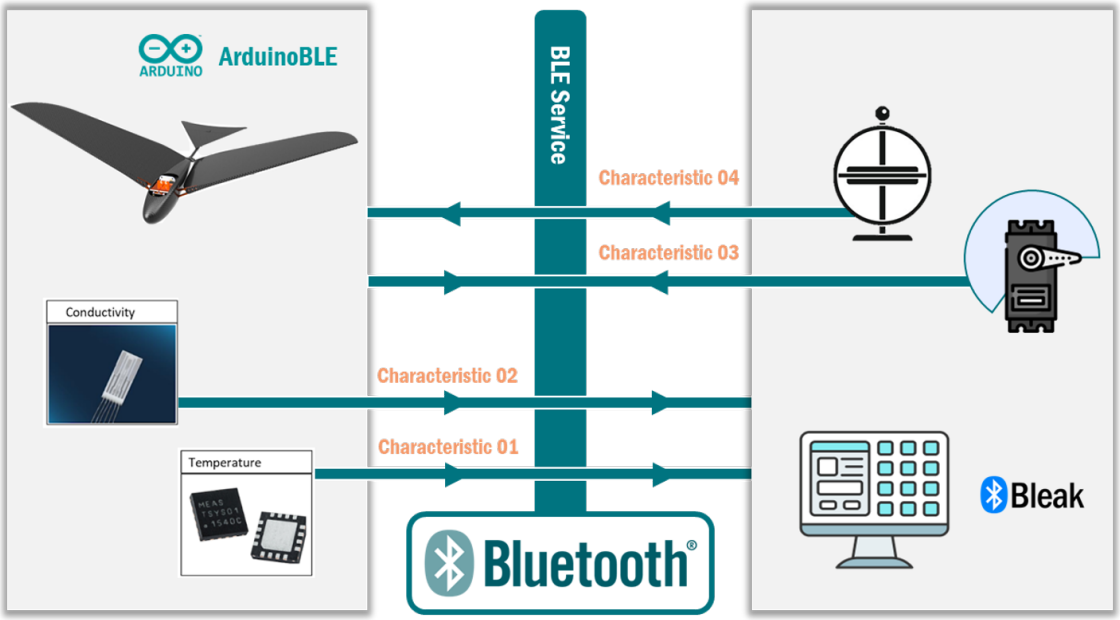


Figure 19: A BLE service is used to communicate between peripherals. In our setup, the drone may have conductivity or temperature sensors that communicate with a computer. Conversely, a user might want to modify the offset of the IMU or send a motor position command using the GUI. All these messages will be send through a BLE *service*.

Then, some *characteristics* are associated to this *service*. For each *characteristic*, the type of data must be specified, as well as the *service* (UUID address) to which it belongs. For each peripheral, one can also specify if it is going to write or read (or both) on the *characteristic*.

5.3 Example of a BLE Communication Sequence

To illustrate how to communicate between a computer and an nRF52840-based board, we consider the following example. The objective is to read IMU values of an nRF52840-based board (*ArduinoBLE* library in Arduino) from a computer (*Bleak* library in Python) using BLE.

5.3.1 Arduino Code on the MCU

First, a BLE *service* and *characteristic* are declared using randomly generated UUIDs[2].

```
String service_UUID = "13012F00-F8C3-4F4A-A8F4-15CD926DA146";
String char_UUID = "6b64b0c4-b675-4725-a02a-d7f6b02ae9db";
BLEService seeedService(service_UUID);
BLEIntCharacteristic IMUCharacteristic(char_UUID, BLERead);
```

Note that the *characteristic* declaration specifies that an external device can only read it. Then we check if the initialization of the BLE device is done correctly.

```
if (!BLE.begin()) {
    Serial.println("STARTING BLE FAILED!");
    while (1);
}
```

The *service* can then be transmitted to the BLE device and the *characteristic* can be linked to its *service*. Note that the so-called local name (*"Seeed XIAO nRF52840 BLE"*) of the BLE device is the name that will appear to the other BLE peripherals.

```
// set advertised local name and service UUID:
BLE.setLocalName("Seeed XIAO nRF52840 BLE");
BLE.setAdvertisedService(seeedService);

// add the characteristic to the service
seeedService.addCharacteristic(IMUCharacteristic);

// add service
BLE.addService(seeedService);
```

Now that the *service* is set, the device can start advertising.

```
BLE.advertise();
```

Then, jumping to the main loop of the program, the device can listen for a new device to connect :

```
BLEDevice central = BLE.central();
```

And if a device is connected, write the IMU values on the *service*. Here the x-accleration value `aX` is a float variable generally comprised between -1.5 and 1.5. Nevertheless, since its *characteristic* is designed to transmit integer, `aX` must be converted to an integer value. To remove the negative part, 10 is added to `aX` which is then multiplied by 1000 to have a resolution of 1/1000. These values are arbitrary and depends on the data that must be sent.

```
if (central) {
    // Print address of the connected device
    Serial.print("Connected to central: ");
    Serial.println(central.address());

    // while the central is still connected to peripheral:
```

[2]Random UUID address can be obtained from this website : https://www.uuidgenerator.net/

```
    while (central.connected()) {
        long currentMillis = millis();
        // if Ts milliseconds have passed, write IMU values
        if (currentMillis - previousMillis >= Ts) {
        previousMillis = currentMillis;
        // Write the x-axis acceleration on the characteristic
        IMUCharacteristic.writeValue(int((aX + 10) * 1000));
    }
```

5.3.2 Python Code on the Computer

The first thing to do for the computer is to scan for surrounding BLE devices.

```
detectedDevices = await BleakScanner.discover()
```

This variable contains a list of structures which corresponds to the list of detected devices. These structures contain, among other things, a field *name* which corresponds to the local name of the device. In our case, we specified this name in the Arduino code as *"Seeed XIAO nRF52840 BLE"*.
Therefore, one can simply browse through the list of structures to find the desired device, and then connect to this latter. Once the communication is established, the data can be read.

```
for d in detectedDevices:
    if d.name == "Seeed XIAO nRF52840 BLE":
        try:
            async with BleakClient(d.address) as client:
                connectedDevice = d.name
                while(1):
                    aX = await self.client.read_gatt_char(char_UUID)
                    aX = int.from_bytes(aX) / 1000. - 10.
        except:
            print("Failed to connect")
```

The `BleakClient` structure provides only the `read_gatt_char(uuid)` function to read integer values from a *characteristic* in byte array format. Since `aX` was sent as `int((aX + 10) * 1000)`, it must be converted back to float value in the Python code.

5.4 Potential Improvements

The BLE communication is working as expected but its use could be simplified by creating an abstraction layer. This could take the form of two main classes (in Python and C++), to interface with *Bleak* and *ArduinoBLE*, that automatically handles data type.

6 Graphical User Interface

The main objective of the Graphical User Interface (GUI) is to provide an easy access to the current states and software parameters of the drone. For example, it might be useful to quickly change some of the controller parameters between two flight tests during the prototyping phase. It may also be interesting to be able to check if all the motors are working correctly before launching the drone. The first part of this chapter presents the general architecture of the GUI. Then, it explains the general operation of the code and how the Bluetooth communications are integrated in the GUI.

6.1 General Architecture

The Python's libraries *Bleak* and *Tkinter* are used respectively to communicate using the BLE protocol and to create the GUI [25]. Its main window contains five different work-spaces also called "frames".

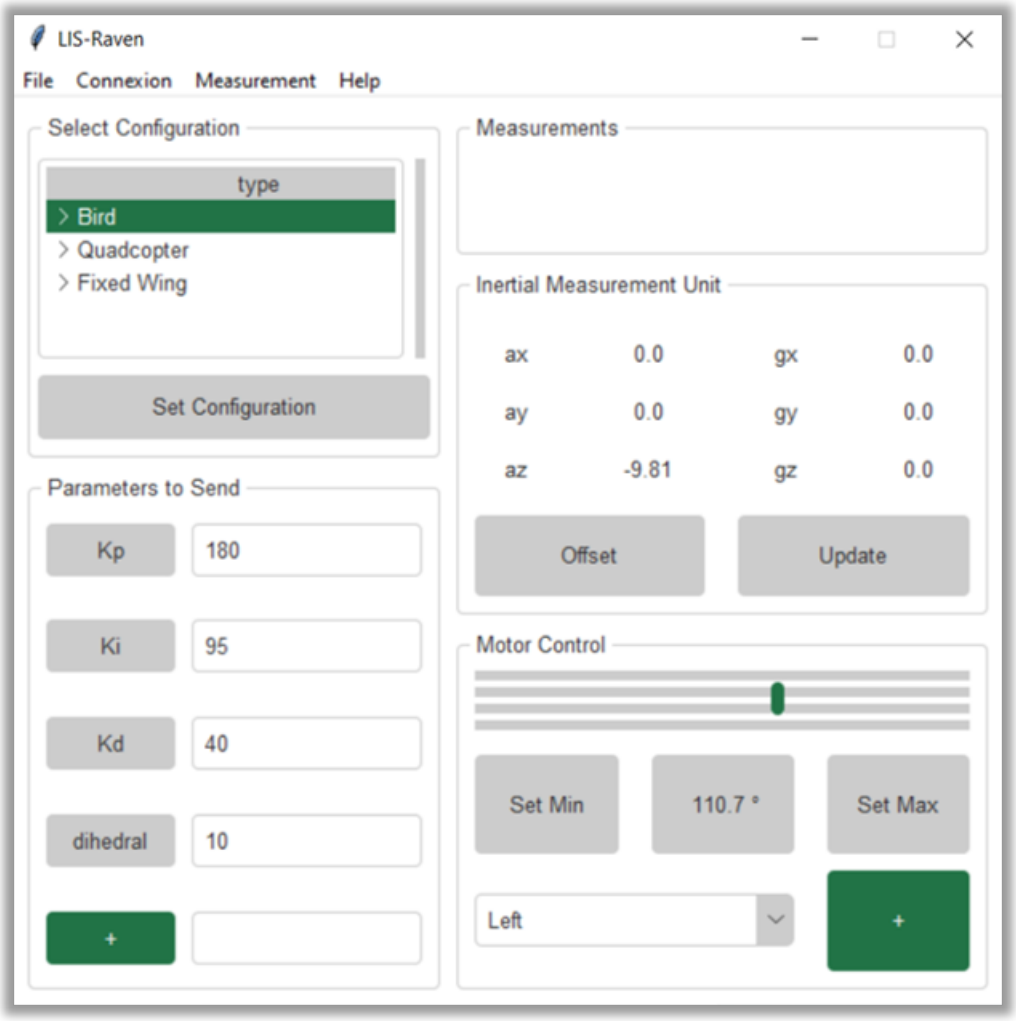
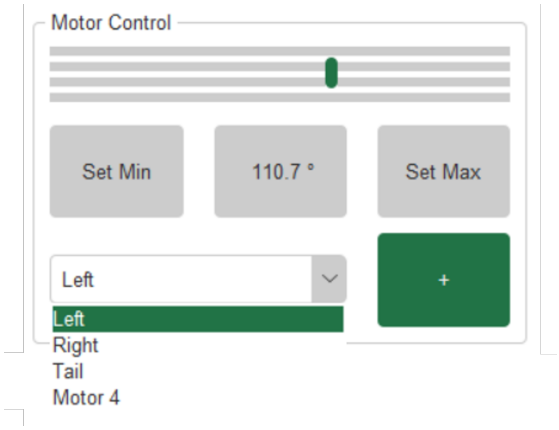


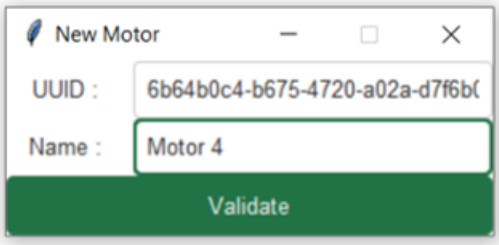
Figure 20: GUI developed in Python to transfer data between a computer and a UAV using BLE.

- **Motor control :** This part provides a way to control and set the operating range of the UAV's motors. Its first widget is a scale which, in this configuration, can select values from 0 to 180 degrees, corresponding to the angular range of a regular servo motor. Right under it, three buttons provide a way to (from the left to the right) set the minimum angle of a motor, move the motor to a specified

angle, set the maximum angle of a motor. Finally, a combo-box widget shows the current available motors and a green push-button labeled "+" provide a way to add motors to this list. When clicking on this latter, a popup window ask the UUID to use for the communication and the motor's name (see figure 21b).



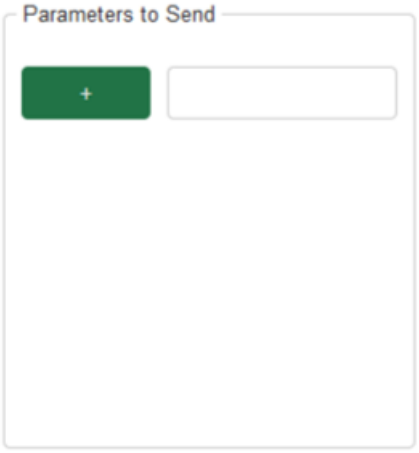
(a) The *"Motor Control"* frame provides a way to move and set the operating range of the UAV's motors.



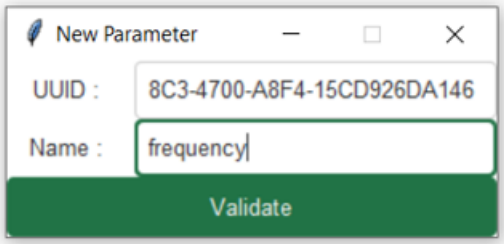
(b) Popup window to set the UUID and name of a new motor to control.

Figure 21: *"Motor Control"* frame and its pop-up window.

- **Parameters to Send :** The objective of this part is to send parameters to the drone. At the beginning, it only shows a green push-button labeled "+" and an empty label (see figure 22a).



(a) *"Parameters to Send"* frame at the opening of the GUI. The green button can be used to add a parameter. Up to five parameters can be sent with this configuration.



(b) Popup window to set the UUID and name of a new parameter to send to the UAV.

Figure 22: *"Parameters to Send"* frame and its pop-up window.

Pushing on the "+" button means that the user wants to add a parameter to send. To do so, one has to provide the characteristic's UUID which will be used to communicate and a parameter name in a pop-up window (see figure 22b).

- **Select Configuration :** A configuration is defined as a collection of motors to control and parameters to send and their respective UUIDs. For example, the flapping-wing UAV presented in this project may have a specific configuration called "Bird" that contains three motors ("right motor", "left motor" and "tail motor") and some parameters to send ("freq", "dihedral") to the computer. Therefore, it might be more convenient to simply load this configuration instead of having to recreate it every time you use this UAV.
  The *"Select Configuration"* frame provides exactly this feature by showing the list of available configurations and a button labeled "Set Configuration" to select the desired one.

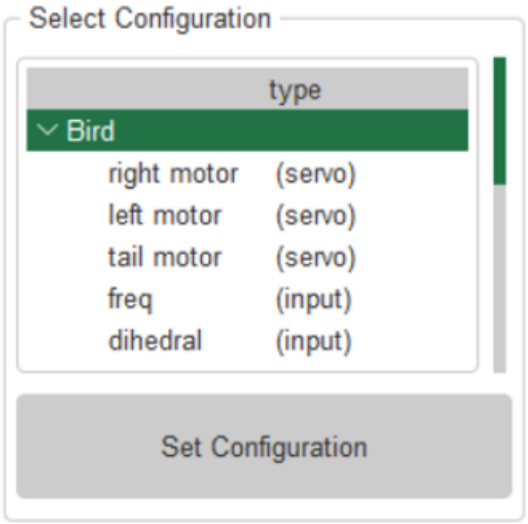


Figure 23: The *"Select Configuration"* frame provides a list of available configurations and a push-button to select the desired one.

- **Inertial Measurement Unit :** In this part, the IMU values (accelerometer and gyroscope) are displayed and two push-buttons are available to (left) manually set the onboard reference position (zero of the IMU) and (right) update the values display (since they are not displayed in real time).

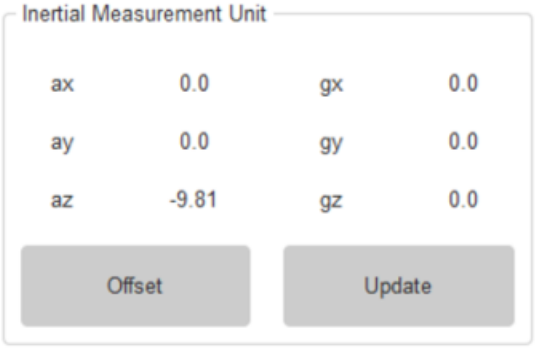


Figure 24: The *"IMU"* frame provides the accelerations a_i and angular velocities g_i on axis i.

- **Measurements :** the objective of this part is to provide some information about the data measured by the UAV's onboard sensors. In this version, nothing is implemented for this purpose but the *"Measurements"* frame is available to implement the desired widgets.

In the menu bar, the following menus are available :

- **File :**
  - **New Config** : Save the current configuration and reset all the motors and parameters.
  - **Save Config** : Save the current configuration.
  - **Save Config As..** : Save the current configuration with a specified name.
  - **Load Config..** : Load a configuration with a specified name.
  - **Quit** : Close the GUI.
- **Connection :**
  - **Disconnect** : Disconnect the BLE peripheral.
  - **Connect..** : This submenu allows to access the Bluetooth connection popup window (see figure 25).

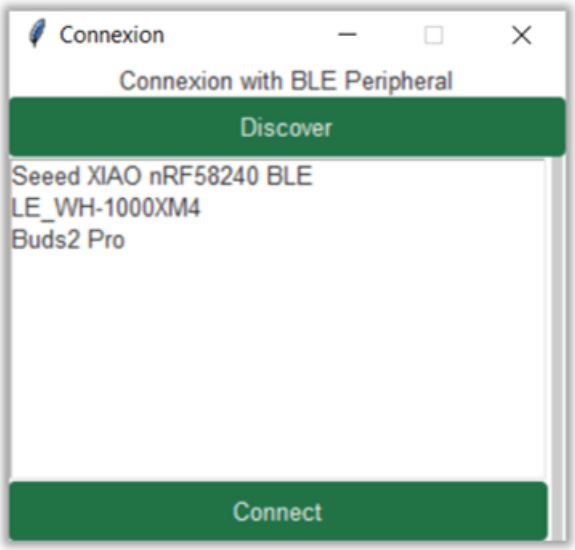


Figure 25: Connection pop-up window after scanning. Three peripherals are detected among which one can recognize the MCU used in our flapping-wing UAV.

By clicking on "Discover", the computer will scan its environment and displays the reachable *Bluetooth* peripherals in the list box. Then, the user can select the desired peripheral and click on "Connect" to establish a connection with the computer. Note that the "Connect" button is modified to "Disconnect" when a connection is established, which users can use to kill the current *Bluetooth* communication (same effect as the "Disconnect" button).

- **Measurements :**
  - **Save as ..** : Save the current configuration with a specified name. Not yet implemented.
  - **Save** : Save the current measurement. Not yet implemented.
- **Help :** Provides a general reminder on how to use the different frames of the GUI.

6.2 Code Architecture

The GUI is handled by a class called `BLE_GUI`. It contains all the code to initialize the different windows as well as the callback function linked to the widgets. The first thing done by its constructor is to initializes the main and pop-up windows. To do so, the procedure is the same for all the frames:

1. Create a `tk.Frame` object, whose parent is the main window.
2. Create the widgets to place in the frame.
3. Place the widgets in the frame using the `grid()` method.
4. Place the frame in the main window using the `grid()` method.

All the GUI's frames presented in 6.1 have their own initialization function in the `BLE_GUI` class. Thus, the constructor simply has to call every initialization function one after the other to create the main window.

```
self._init_menubar()
self._init_connectPopup()
self._init_helpPopup()
self._init_addParamPopup()
self._init_addMotorPopup()
self._init_frmTable()
self._init_frmConfig()
self._init_frmControl()
```

Then, the constructor creates a `BLE` object (based on *Bleak*) which handles all the BLE communications and external peripherals (see section 6.3). This object is simply running another thread in parallel with the *tkinter* main loop.

```
self.ThreadBLE = BLE(self.motorList) # Create BLE object
self.ThreadBLE.start() # Start BLE thread
self.show() # Show main window
```

Finally, the constructor launch the BLE thread and shows the main window.

6.3 BLE Integration in the GUI

As presented in the previous section the BLE communications are handled by a `BLE` object, which is an attribute of the `BLE_GUI` class. Therefore, the GUI's widgets have access to the `BLE` attributes but the opposite is false.
In the `BLE` class, two loops are working in parallel using the *asyncio* library, which allows asynchronous programming [26]. The first one handles the discovering process of new BLE perihperal while the second one establish the connection and handles the communication process. Each of these loops are working in the same way, using flag variables. The idea is to do nothing until the value of a flag variable (generally controlled by a push-button) change its value. When it happens, the code execute the desired action, reset the flag variable and goes back to its "do nothing" state.
The discovering loop is a good example, with the variable `ask2Discover` used as a flag. Its value is directly linked to the "Discover" button of the connection pop-up (see figure 25). When the button is pushed, it sets `ask2Discover = True`, which initiates a BLE scan.

```
    async def discoverLoop(self):
        while True:
            # Wait for discover request
            while not(self.ask2Discover):
                await asyncio.sleep(0.5)

            # Scan for BLE peripherals
```

```
8               self.detectedDevices = await BleakScanner.discover()
9
10              # Reset the flag
11              self.ask2Discover = False
```

6.4 Potential Improvements

While the main features of the GUI are working well, some useful ones are still to be completed :

- **Measurements Frame :** Nothing has be done for now about a way to transfer measurements from the drone to a computer. This is mainly because this part of the project has not been merged yet with the "measuring station" planned within the drone. The *Measurements* frame is intended for this purpose as well as the *Measurement* submenu in the menu-bar. For example, a future project could intend to provide a way to transfer onboard sensors measurements on the computer directly via the GUI.
- **Real-time display of the IMU values :** This improvement would probably make the GUI more user-friendly but also probably requires some important modifications of the architecture. For this reason, it has not been implemented in this version.

7 Control

This section presents the very first control method tested with this flapping-wing drone. It starts by presenting the different wings and tail control approaches, then presents the RC mapping, and finally proposes some improvements.

7.1 Wings Control

For the wings control, three different approaches have been implemented and are exposed in this section. The main idea is always the same; Imposing a cyclic signal $\theta(t)$ to the wings with adjustable amplitude and frequency.

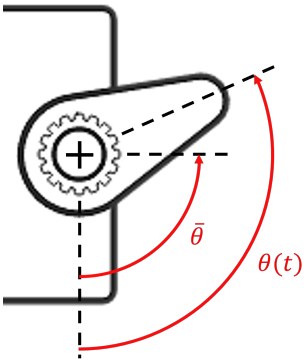


Figure 26: Wing control is done by imposing a signal $\theta(t)$ with mean value $\bar{\theta}$. For this project, $\bar{\theta} = \pi/2$ corresponds to a dihedral angle of $0\,rad$.

7.1.1 Sinus Mode

In Sinus mode, we impose for each Servo motor a sinusoidal oscillation of the form :

$$\theta(t) = \bar{\theta}(t) + A(t) \cdot \sin\left(2\pi \cdot f(t) \cdot t + \phi(t)\right) \tag{1}$$

Where $\theta[rad]$ is the angular position of the motor, $A[rad]$ is the amplitude, $t[s]$ is the time, $f[Hz]$ the frequency, $\phi[rad]$ the phase offset and $\bar{\theta}[rad]$ the mean angular position.

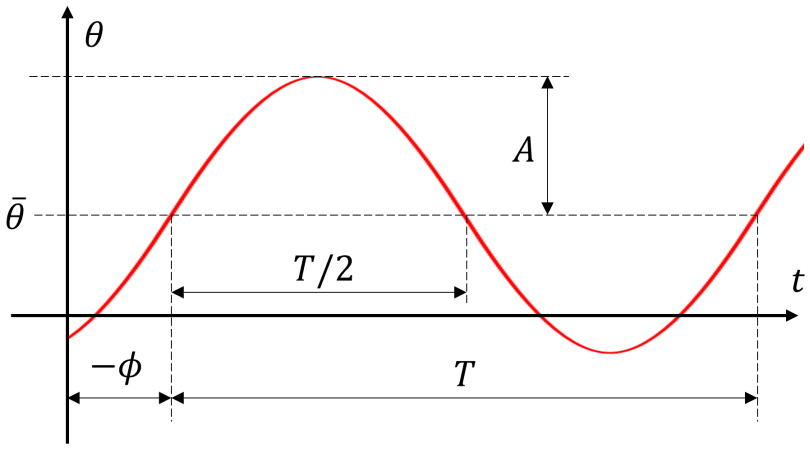


Figure 27: Sinusoidal trajectory imposed as angular position of the wings motors. The phase offset ϕ is only used to guarantee continuity of the signal when the frequency quickly changes.

Denote by index R and L the values corresponding to each of the motors (Left and Right). We can make the following observations :

- If $|A_R - A_L| > 0\,rad$, a moment in the roll axis will appear.
- Increasing $\bar{\theta}_R = \bar{\theta}_L = \bar{\theta}$ will result in a higher dihedral angle, hence providing more stability.
- Increasing $f_R = f_L = f$ will result in more lift and thrust.

Since these values are relevant to control the trajectory of the drone, they have been mapped from the remote controller (see section 7.3). Nevertheless, the oscillations cannot be infinitely large for mechanical reasons (wings have a mechanical range of around 80° in practice) and because of the Servo motors frequency response (see figure 28).

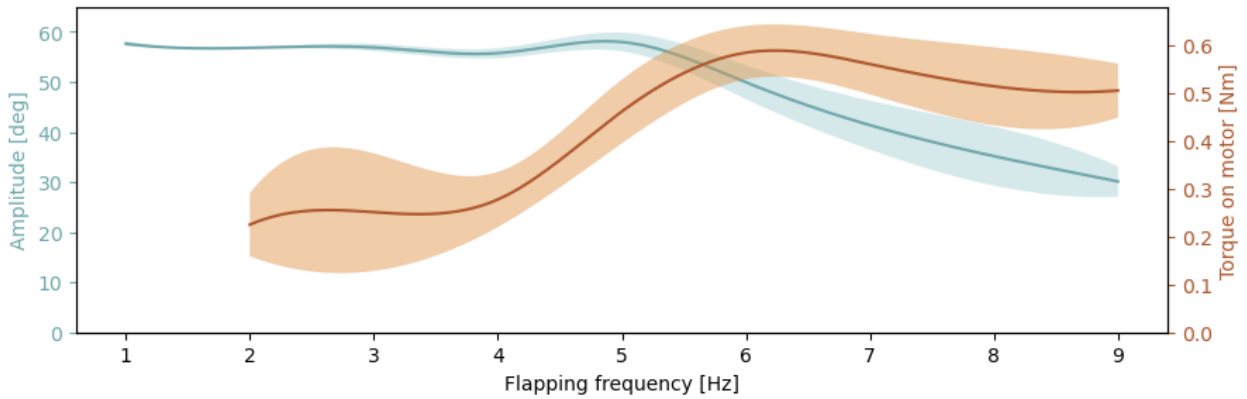


Figure 28: Response of the Servo motors used for the wings actuation. The motor was asked to oscillate at frequencies from 1 to 9 Hz with a peak-to-peak amplitude of 60 degrees. One can see a decrease of the amplitude from 5 Hz as well as a saturation of the delivered torque.

To handle these issues, the amplitude of each sinusoidal signal is limited depending on its mean value $\bar{\theta}$ and frequency f, i.e. :

$$A(t) = A(t, f, \bar{\theta})$$

Based on figure 28, the frequency-related limitation starts at $5Hz$ and can be designed for example to decrease linearly the maximal amplitude from $A(5Hz) = 30°$ to $A(10Hz) = 20°$.
Finally, since the frequency $f(t)$ will be linked to one of the remote controller's channels, a sudden change of frequency is possible and may result in undesired movements of the wings. To avoid potential jumps, we use the phase offset $\phi(t)$ to enforce continuity of the frequency-varying sinusoidal signal. By imposing this condition :

$$2\pi \cdot f(t+\delta t)(t+\delta t) + \phi(t+\delta t) = 2\pi \cdot f(t) \cdot (t+\delta t) + \phi(t) \tag{2}$$

Which imposes :

$$\phi(t+\delta t) = 2\pi \left(f(t) - f(t+\delta t)\right) \cdot (t+\delta t) + \phi(t)$$

We ensure that if the frequency changes between t and $t+\delta t$, the argument of the sinus will be independent of the frequency variation $f\,(t+\delta t) - f(t)$ (see figure 29).

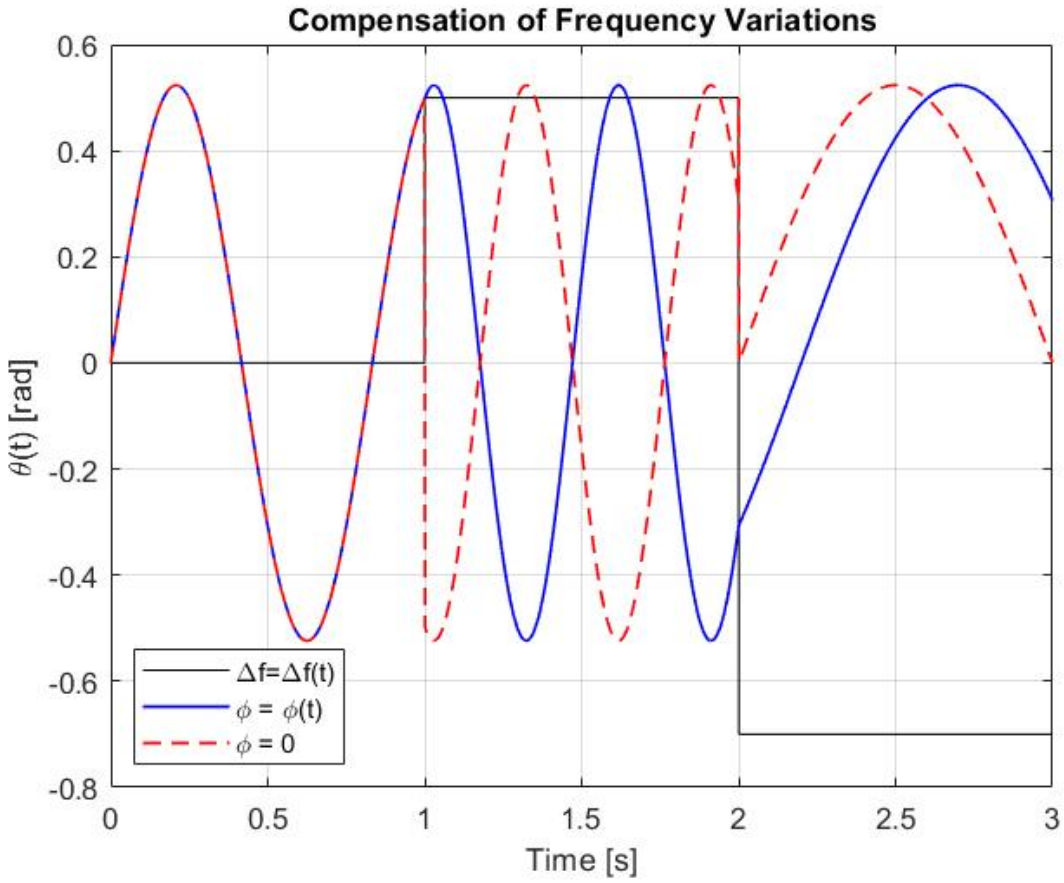


Figure 29: A frequency variation $\Delta f(t)$ can creates signal jumps if no phase compensation is done (i.e. $\phi = cste$). Jumps clearly appear on the uncompensated signal (dashed red) while the compensated one (blue) remains continuous. The black line represents the frequency error with respect to the nominal frequency (1.2 Hz). Frequency changes suddenly at $t = 1\,s$ and $t = 2\,s$.

7.1.2 Sawtooth Mode

In this mode, the oscillations follow a saw-tooth trajectory. The generation of this latter is done using two piece-wise affine functions designed depending on the desired amplitude, frequency of oscillations and shape of the sawtooth trajectory. This latter is controlled by a factor $\epsilon \in [0, 1]$ which defines if the trajectory is symmetrical or if the slope varies between up and down movement. For an oscillation of amplitude $A[rad]$ at a frequency $f = 1/T[Hz]$ ($T[s]$ is the period) around a mean position $\bar{\theta}[rad]$, the angular position θ of a wing is given by :

$$\theta(t) = \begin{cases} \frac{2A}{\epsilon T}\left(t - \frac{\epsilon T}{2}\right) + \bar{\theta} & t \in [0; \epsilon T[\\ \frac{2A}{(\epsilon-1)T}\left(t - \frac{T(\epsilon+1)}{2}\right) + \bar{\theta} & t \in [\epsilon T; T[\end{cases} \tag{3}$$

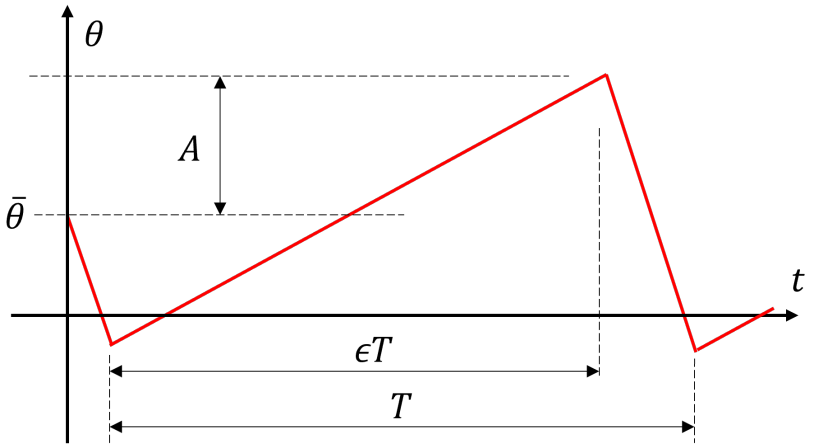


Figure 30: Illustration of a sawtooth trajectory generated with $\epsilon = 0.85$. It creates a slow movement in the up-flapping phase and a fast one in the down-flapping phase.

7.1.3 Binary Mode

This mode simply switches the angular position θ of each motor between $\bar{\theta} - A$ and $\bar{\theta} + A$ during one period T (see equation 4).

$$\theta(t) = \begin{cases} \bar{\theta} + A & t \in [0; T/2[\\ \bar{\theta} - A & t \in [T/2; T[\end{cases} \tag{4}$$

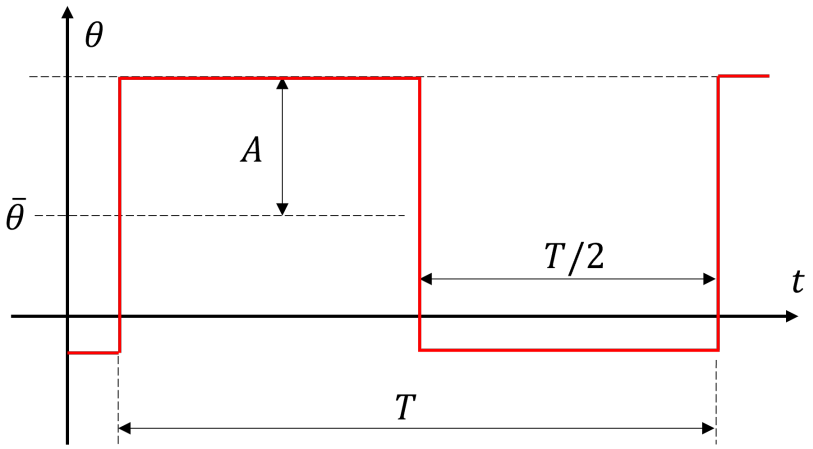


Figure 31: Binary trajectory simply switches the target value between two opposite values.

If this trajectory profile is used at high frequencies (more than 5Hz for this motor, according to figure 28), the motor will probably fail to reach the reference amplitude and the trajectory will therefore look like a symmetrical saw-tooth trajectory.

7.2 Tail Control

Since the tail position has a direct effect on the pitch angle, it has been directly mapped to one of the remote controller's joystick. Its mean position is about 30 degrees and can vary between 0 and 60 degrees.

7.3 Remote Controller Mapping

The four available channels have been mapped in the following way :

- Vertical Left (VL) joystick (channel 0) : This channel allows to modify the flapping frequency of the two wings. Therefore it has an important effect on the thrust and lift of the drone. When the joystick is held down, the wings stop their oscillations ($f = 0Hz$) and return to their average value $\bar{\theta}$.
- Vertical Right (VR) joystick (channel 1) : This channel controls the position γ of the tail and has therefore an effect on the pitch of the drone.
- Horizontal Right (HR) joystick (channel 2) : This channel controls the difference of amplitude $A_R - A_L$ between the two wings. If the joystick is fully held on the right, then the amplitude of the right wing will be divided by two while the amplitude of the left wing will remain the same. The opposite effect is obtained when the joystick is fully held on the left.
- Right Potentiometer (channel 3) : This channel controls the dihedral angle α and is therefore important for the stability of the drone.

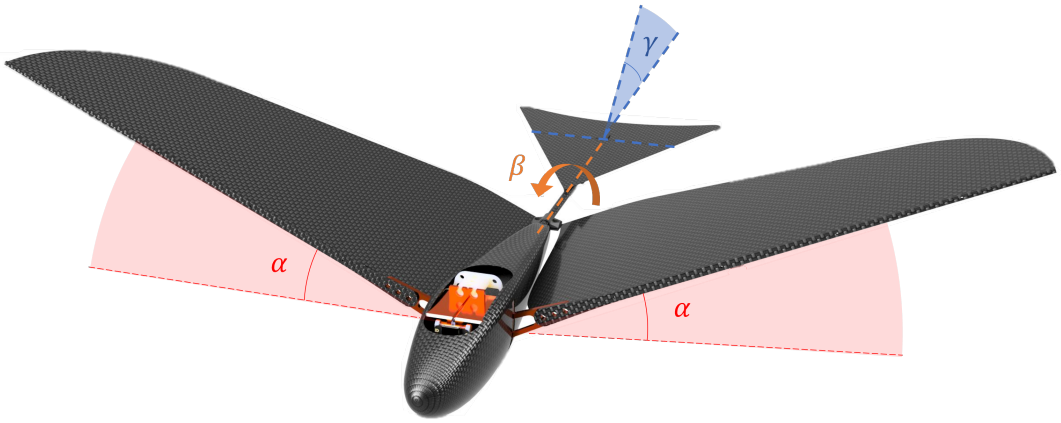


Figure 32: Illustration of the dihedral angle α, tail's position γ and orientation β. α and γ are actuated while β is supposed to be fix.

This mapping corresponds to the channel mapping of the RC shown by figure 33.

Figure 33: Mapping of the remote controller with some parameters of the controller.

7.4 Potential Improvements

The first improvement concerns the trajectory generation. While sinusoidal signal remain continuous thanks to the phase compensation, sawtooth and binary trajectories may have jumps when the frequency changes. During flight tests, this will force the pilot to keep the frequency constant.
The second improvement is about the tail's position, which is for now simply set manually by the pilot using the remote controller. To reduces flight oscillations of the UAV, it might be a good test to impose small oscillations to the tail with the same frequency as the wings.
Finally, an attitude controller could be implement such that parameters of the drone (tail's position, wing's amplitude) are not directly linked to the RC. The command from the receiver could then be high-level parameters such as the pitch, roll and yaw angles. Nevertheless, such a controller requires, among other things, to measure the attitude of the drone, which might be a hard problem because of its oscillations.

8 Flight Tests

Two flight tests have been done during this project using the control methods presented in section 7. The first one took place on the 8th of December 2022 on the campus EPFL and the second one on the 20th of December at the same place. This section presents for each flight test the drone setup, a qualitative analysis of the results obtained, and finally propose some improvements.

8.1 First Flight Test

8.1.1 Drone Setup

Figure 34 shows how the drone was prepared for the first flight test. Shock absorbers are disposed in the front of the drone and a blue cover on the top is protecting the electronic devices. The antennas of the receiver are taped to the outside of the drone.

Figure 34: Illustration of the drone at the take-off area before the first flight.

Before testing, the complete drone (including the battery) was weighted at 194 grams, very close to the 200 grams target.

8.1.2 Results

The very first tests consisted of simply launching the drone without moving the wings, with dihedral and tail angles of approximately 10° and 30° respectively. The take-off is done by simply launching the drone by hand, just like a paper plane. During this test, the drone remained stable and hovered for about ten metres. To verify the control authority of the tail, the same test was done with a tail angle of 0°, which led to the drone's nose dropping.

(a) (b) (c)

Figure 35: Take-off procedure used for the flight test. At the beginning (a), the wings are static and they start flapping during the launching phase (b).

During the second test, only sinusoidal signals were used to control the wings (see section 7.1.1). The take-off is done the same way as for the first test, except that the flapping frequency is quickly increased during the launching (from 0 Hz to 10 Hz). For the tail's position, the launching angle was generally around 20° and then adjusted to try to maintain a good angle of attack. Nevertheless, the thrust generated by the wings was not sufficient to reach a speed that provides enough lift. The drone was still stable, even for dihedral angles of 0°, but not able to maintain itself in the air. One of the main reason was an error in the control policy implementation which was limiting the wings position between ±30° instead of the full range (mechanical limits) which is around ±40°.
Finally, the drone withstood all the shocks experienced during this test, mainly thanks to its shock absorbers and the flexibility of its wings.

8.2 Second Flight Test

8.2.1 Drone Setup

The second flight used the same setup except for the wings, to which we added camber (see figure 37) to improve the lift / drag ratio.

Figure 36

Figure 37: The wings profile is curved using strings that bend the supporting rods (b). The bending can be adjusted by twisting the strings.

8.2.2 Results

For this second flight test, three control methods for the wings have been tested (sinus, sawtooth and binary). The first trial used the sinusoidal signal and approximately the same parameters as for the first flight test (dihedral angle $\approx 10°$, initial tail position $\approx 20°$). The only difference was the maximum flapping frequency which was limited to 8 Hz.
During these tests, the drone covered longer distances of about 20 metres (see figure 38). This improvement is probably due to the increased wing flapping amplitude, but also partly due to the slight headwind present during this test. Neverteheless, it is still not able to maintain itself in flight, probably because the wings motors are not powerful enough.

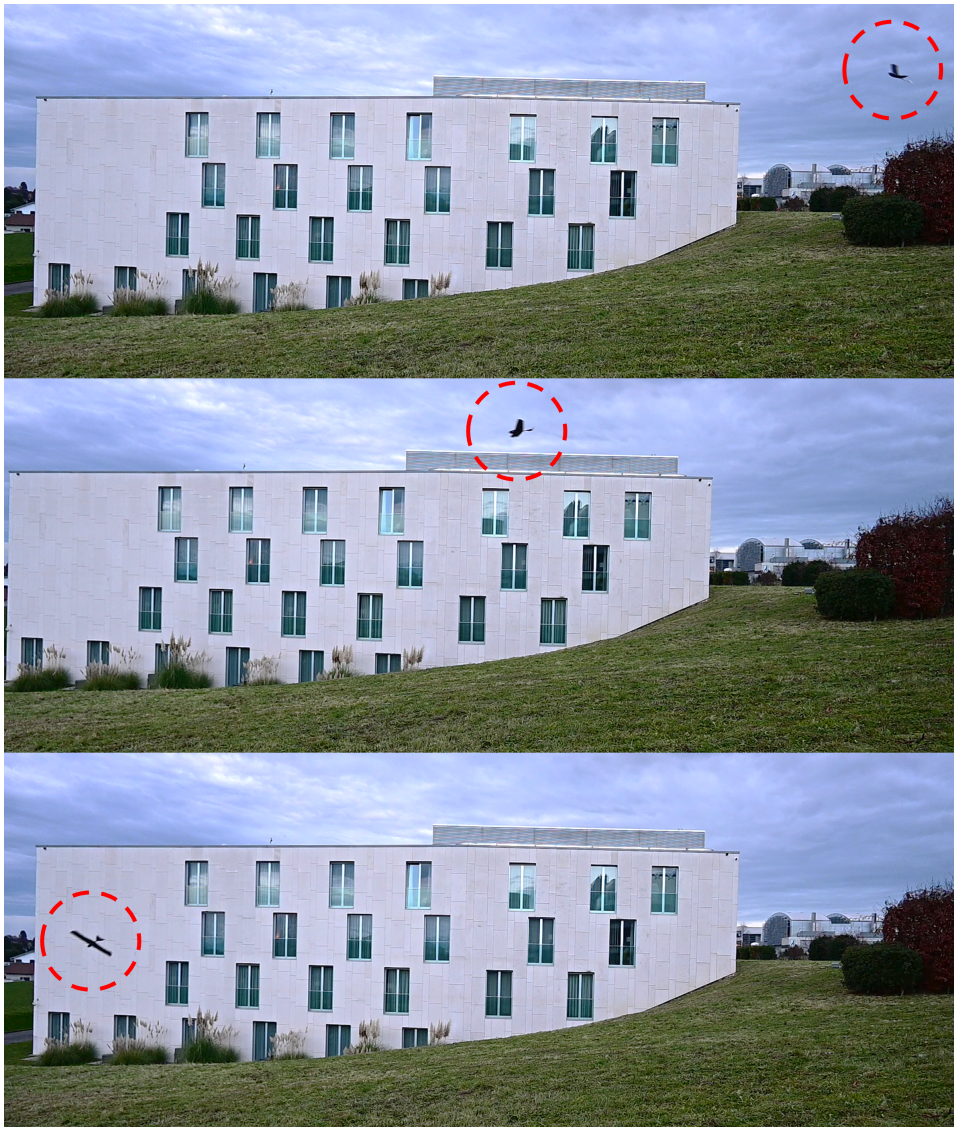


Figure 38: Flight test using sinusoidal signals to control the wings and a dihedral angle of approximately 10°.

The drone remained stable again, even at very low dihedral angles (down to 0°). Some tests showed the sensitivity of the tail's orientation β (see figure 32). Indeed, a bad orientation of the tail will result in a systematic deviation of the drone toward one direction.
Finally, The front part of the drone cracked during this test, but it did not prevent it from working properly. Again, the robustness of the drone is probably mainly due to its lightness, the flexibility of the wings and the schock absorbers.

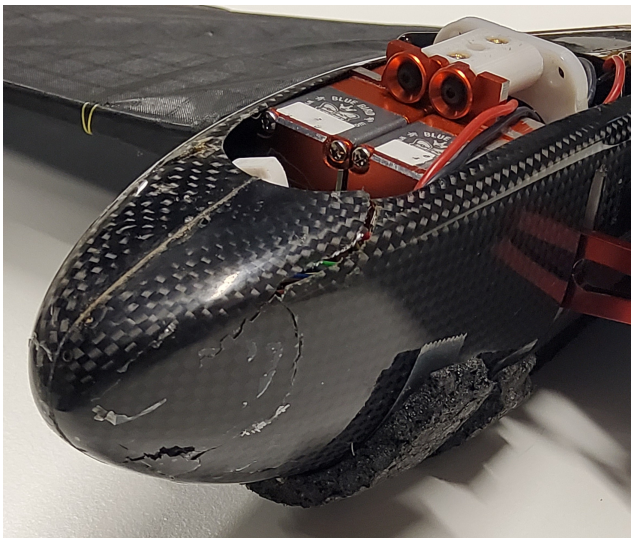


Figure 39: Crack in the front of the drone after the second flight.

8.3 Possible Improvements

To increase the mechanical power deployed by the drone, several options are available. The most promising one simply consists of increasing the maximum motor supply voltage of the wings motors. This will allow the servo motors to operate within their full power range. If it is not sufficient, a brushless motor might be a good options to test, along with rods to transform the rotation into oscillations.

9 Conclusion

As a reminder, the objectives at the beginning project were the following :

(i) Develop the software framework that will run the autopilot.

(ii) Develop and test an oscillation-compensating controller.

(iii) Interface the drone with BLE, long-range radio and sensors.

(iv) Build a user interface to quickly change settings remotely from a computer or a smartphone.

The first part of the project has focused on interfacing the drone with BLE and a long-range radio receiver. These two features are now available to communicate with the robot and will be useable for other project in the form of Arduino code examples / skeletons.

The second part focused on the control of the three onboard servo motors and the reading of the IMU, which has been done using standard Arduino libraries. Then, a first controller, which mainly consists of imposing periodic signals to the wings, has been deployed on the drone. The attitude of the robot was controlled indirectly, using the wings difference in amplitude, flapping frequency, tail's position and dihedral angle, which were mapped to the remote controller.

Then, the electronic setup of the drone has been developed and assembled on the robot despite the limited available space.This step has enabled the drone to realize its very first take-off.

Two flight tests have been done, during which the drone showed a very stable behaviour, even for very low dihedral angles. Furthermore, tests have shown a good control authority on the pitch angle using the tail's position. Nevertheless, the motors power is apparently not sufficient to keep the drone flying. This problem could potentially be solved by increasing the motor supply voltage and therefore exploit its full power.

Finally, a graphical user interface has been developed to quickly change settings remotely from a computer. The GUI is able to scan its environment and connect to the desired device. After the connection, the user can fill in the number of motors of the drone along with their type (brushless, servo) and the parameters that can be set from the GUI (controller gain or flapping frequency for example). The motor's state can also be adjusted from the GUI as well as its limits (position or speed).

Even though this project focused on flapping-wings UAV, the tools that came out of it such as BLE, long-range radio communications and the GUI can be used for any type of UAV. It can therefore be viewed as a versatile research platform, which is intended to be used in other projects.

Stephen Monnet

References

[1] Nico Nijenhuis Gerrit Adriaan Folkertsma Wessel Straatman. "Robird: A Robotic Bird of Prey". In: *IEEE Robotics & Automation Magazine* (2017). DOI: 10.1109/MRA.2016.2636368.

[2] *SmartBird documentation, Festo website*. https://www.festo.com/net/SupportPortal/Files/46270/Brosch_SmartBird_en_8s_RZ_110311_lo.pdf. Accessed: 2023-05-01.

[3] *Getting Started with Seeed Studio XIAO nRF52840 (Sense), Seeed Studio website*. https://wiki.seeedstudio.com/XIAO_BLE/. Accessed: 2023-04-01.

[4] *N3 flight controller, DJI website*. https://www.dji.com/ch/n3/info. Accessed: 2023-05-01.

[5] *Pixhawk 4 flight controller, Pixhawk website*. https://docs.px4.io/main/en/flight_controller/pixhawk4.html. Accessed: 2023-05-01.

[6] *mRO Pixracer flight controller, Pixhawk website*. https://docs.px4.io/main/en/flight_controller/pixracer.html. Accessed: 2023-05-01.

[7] *KISS flight controller serie, Flyduino website*. https://flyduino.gitbook.io/kiss-documentation/english-kiss-docu. Accessed: 2023-05-01.

[8] *F7 AIO flight controller, SpeedyBee website*. https://www.speedybee.com/product_images/download/manual/SBF7_AIO_manual.pdf. Accessed: 2023-05-01.

[9] *Matek F411-WSE flight controller, Matek website*. https://inavfixedwinggroup.com/shopping-list/flight-controllers/. Accessed: 2023-05-01.

[10] *MP2307 datasheet*. https://cdn-shop.adafruit.com/datasheets/MP2307_r1.9.pdf. Accessed: 2023-04-01.

[11] *BMS-A921+ datasheet, Blue Bird Technology website*. https://www.blue-bird-model.com/products_detail/478.htm. Accessed: 2023-04-01.

[12] *BMS-A10V+ datasheet, Blue Bird Technology website*. https://www.blue-bird-model.com/products_detail/74.htm. Accessed: 2023-04-01.

[13] *R9 Slim+OTA user's manual, FrSky website*. https://www.frsky-rc.com/r9-slim-ota/. Accessed: 2023-04-01.

[14] *nRF52840 datasheet*. https://files.seeedstudio.com/wiki/XIAO-BLE/Nano_BLE_MCU-nRF52840_PS_v1.1.pdf. Accessed: 2023-04-01.

[15] *ACCESS Protocol, website of Oscar Liang*. https://oscarliang.com/frsky-access-protocol/. Accessed: 2023-04-01.

[16] *SBUS protocol, github repository of the Robotics and Perception Group, ETH*. https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/SBUS-Protocol. Accessed: 2023-04-01.

[17] *pulseIn documentation, Arduino website*. https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/. Accessed: 2023-04-01.

[18] *Multiplexer CD4051BE datasheet*. https://datasheet.octopart.com/CD4051BE-Texas-Instruments-datasheet-7280354.pdf. Accessed: 2023-05-01.

[19] *Bolderflight library, github repository*. https://github.com/bolderflight/SBUS. Accessed: 2023-05-01.

[20] *ArduinoBLE documentation, Arduino website*. https://www.arduino.cc/reference/en/libraries/arduinoble/. Accessed: 2023-04-01.

[21] *Bleak documentation*. https://bleak.readthedocs.io/en/latest/. Accessed: 2023-04-01.

[22] Bluetooth SIG Inc. *Specification of the Bluetooth System*. Vol. 4. 2010.

[23] *Bluetooth Low Energy (BLE): A Complete Guide, website of Novelbits*. https://novelbits.io/bluetooth-low-energy-ble-complete-guide. Accessed: 2023-04-01.

[24] *Universally Unique Identifier, Wikipedia*. https://en.wikipedia.org/wiki/Universally_unique_identifier. Accessed: 2023-04-01.

[25] *Tkinter documentation, Python website*. https://docs.python.org/3/library/tkinter.html. Accessed: 2023-04-01.

[26] *asyncio documentation, Python website*. https://docs.python.org/3/library/asyncio.html. Accessed: 2023-04-01.