Iby and Aladar Fleischman Faculty of Engineering Tel Aviv University
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן אוניברסיטת תל-אביב

XTEND
Extending Reality Skywards

# CRAZYFLIE 2.0 DRONE STABILIZATION FINAL PROJECT

*Project Number: Project Report*

## QUADROTORS, DRONES AND VIRTUAL REALITY

## UNDER THE SCOPE OF CONTROL SYSTEMS

**BY: LAURENE BOUSKILA ID:949271043**
**SHIR BARLAS ID: 301358776**

**Project Performed at: Xtend, TAU VENTURES**

**I approve the project Plan:**

Supervisor's Signature

# INTENTIONS

Before diving into the deep content of this project, we wanted to reveal our main motivation while writing this booklet.

This project book was written with the concern of making the readers interested in the field of unmanned aerial vehicles (UAV's) and giving them a taste of the work we had the opportunity to do.

In our expanding technological world, the field of control theory appears to be more and more useful.

Especially when it comes to ensuring the stability of aerial vehicles, the theory of feedback systems melds into practice in order to give to the world the chance to explore new perspectives.

As a matter of fact, a various range of spheres seems to benefit from this high-technological progress: going from agriculture, through photography domain to security concerns.

As you understood, infinitely many different sources are also contributing to the expansion of this sector and I will make sure to mention some of them through this paper.

One of them, Xtend, which is an expanding and growing young startup, have given us the golden chance to complete our final project at TAU Ventures.
Shir Barlas, my project partner and I, worked for months on this project, trying to understand the working process of the drone control itself as well as the different platforms and softwares coordinating all the information the drone needs in order to fly correctly. This led us to the observation and understanding of the Kalman estimator filter, the Mellinger controller and the role they play in the big picture. We also had to analyse the code that implements them and modify it in order to improve under specific conditions the stability of flight.

I would like to thank the CEO of this exciting company: Aviv Shapira for this.
I would also like to personally express my gratitude to Yoni Mandel, for all the work, efforts and support he devoted to this project.
I am deeply beholden for the care taken while helping us carrying this research out.
Moreover, it is important for me to underlie the chance I had to collaborate with my project partner Shir Barlas who committed himself from the beginning until the end into this project.
Last, but not least, I thank my family for their infinite support.

Laurène Bouskila.

# Contents list

# INTRODUCTION

As previously mentioned, this project is taking us through the analysis of the working process of the Crazyflie 2.0 drone.


The Crazyflie 2.0 is a UAV (unmanned aerial vehicle) created by the Bitcraze firmware. The code implementing its control is available on the Github platform as an open source project, opening possibilities for changes and modifications.

Our concern was oriented towards the stabilisation of the former, understanding the different flight modes it can support and try to reach them from a more balanced way.

The main issue we attempted to solve is this:
Imagine the drone getting an impulse (such as being thrown in the air). We would like the drone to start its motors, stabilize itself, perhaps fly in a certain trajectory and land safely.

For that, we needed the onboard sensors to send their collected data to the control system. As soon as they cross a certain threshold, we instruct the device to stabilise itself - hover, then follow a pre-defined, programmable trajectory, and finally land.
Hovering is a flight manner in which the UAV is stable at a certain position in the air and remains in this steady state.

The final goal of this research was to improve the stability of the device, we went through different steps in order to achieve it
.
First, we had to understand from a theoretical point of view, how is the Crazyflie flight regulated, meaning answering the following questions:
What are the different flight modes it supports and how are they achieved?
Which filters are implementing the control of the device?
What role do they play in the big picture?
What is the physical/mathematical theory underneath them?
Which interfaces are relating the sensors and the quadcopter?
And finally, how can we influence the stabilisation?

We'll try to explain as clearly as possible how we answered these questions along this book and keep your interest awake !

# THE BIG PICTURE

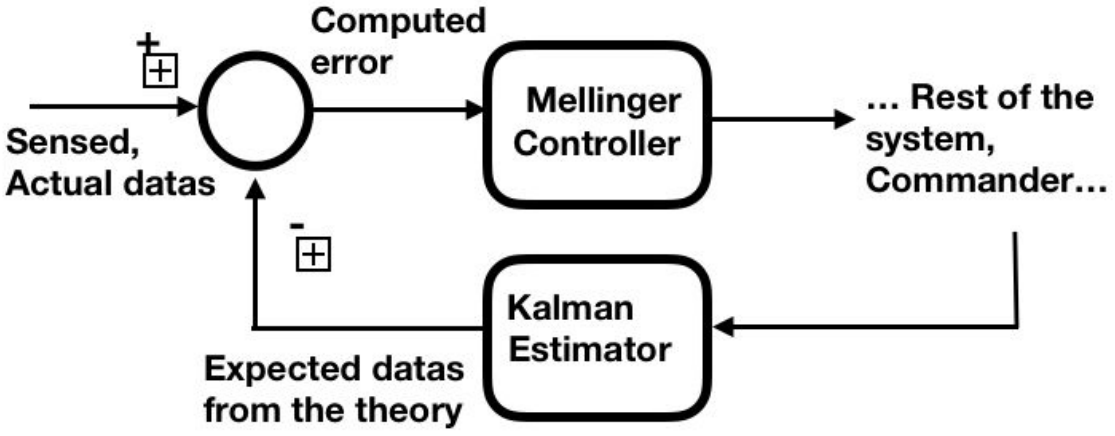
## - GENERAL SCHEME

As stated in the intentions part of this booklet, there are different filters that are representing the control of the Crazyflie drone.

Taking us to the fields of Practical feedback systems and Control theory, we thought it would be relevant to detail them for the understanding of the project.

The general scheme responds to the principles of the feedback theory according to a typical pattern:
The actual (although potentially noisy) data of certain parameters (location and velocity along each axis; pitch, roll, yaw angles, etc.) is being read from the onboard sensors.
expected values are being calculated based on previous state and the state of the motors, using several components (filters,controller,estimator, etc.).

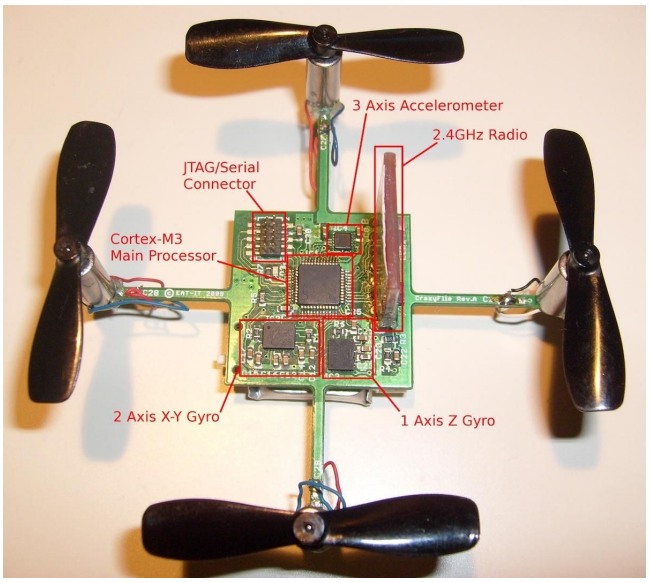
Both of them allows us to compute (by a simple subtraction) the actual error that will eventually decrease as time goes on, allowing the trajectory to be more and more accurate to the desired one.



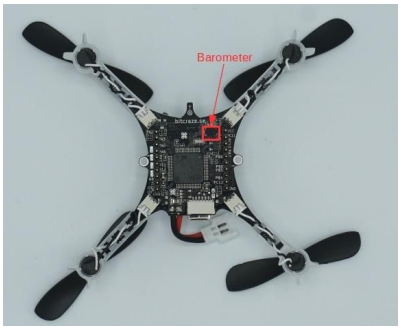

Sheme of the working process of the crazyflie 2.0

# - DIFFERENT SENSORS

The Crazyflie 2.0 can read data from a wide range of sensors. We won't use all of them in this research, only the one that interests us.
In this section, we'll present them and explain what kind of data they collect.




Crazyflie 2.0 and different sensors

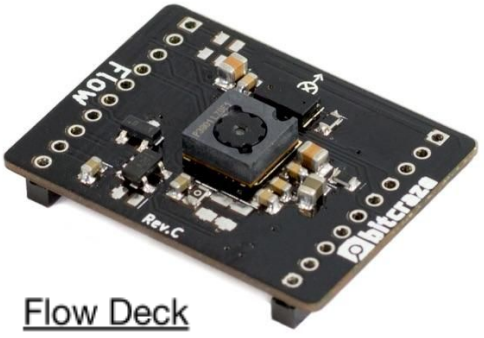



* The gyroscope and the accelerometer are IMU sensors that measures respectively the orientation and the non-gravitational acceleration of the object.
They are gathered on a single chip which is fixed to the body of the drone as can be seen in the image.

Crazyflie 2.0 and barometer

* The barometer: measures the air pressure applied on the drone.

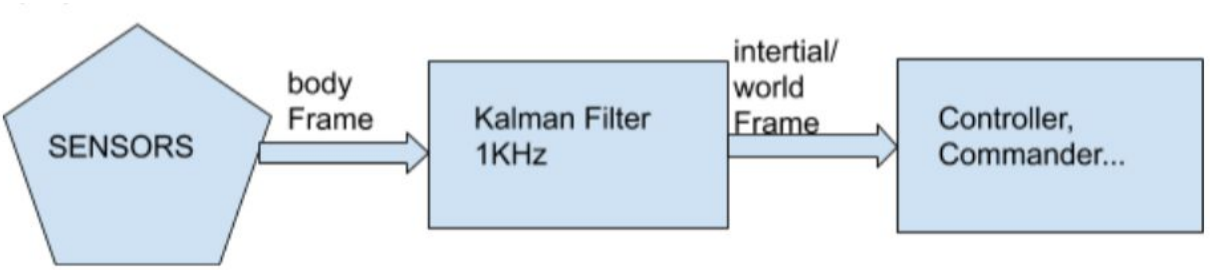
* The flow deck: is a new expansion board for Crazyflie 2.0 that contains an optical flow sensor and a ranging sensor.
These two sensors allows the Crazyflie to understand how it is moving above the floor and using this information the Crazyflie can fly itself. It is fixed under the drone in order for the camera to face down.




Flow Deck

# - KALMAN ESTIMATOR FILTER

In this part of the document, we'll try to give an understanding of the use of the Kalman Filter in the Crazyflie 2.0.




Kalman Estimator role in the big picture

The Kalman filter, as other control filters, is the bond between the sensors and the rest of the system (controller, commander, etc.).

As known, the Kalman Filter is one of the most known figures to use in state estimation.
Nowadays, Kalman filter is widely used in all kinds of fields such as radar, electronic vision, communication…

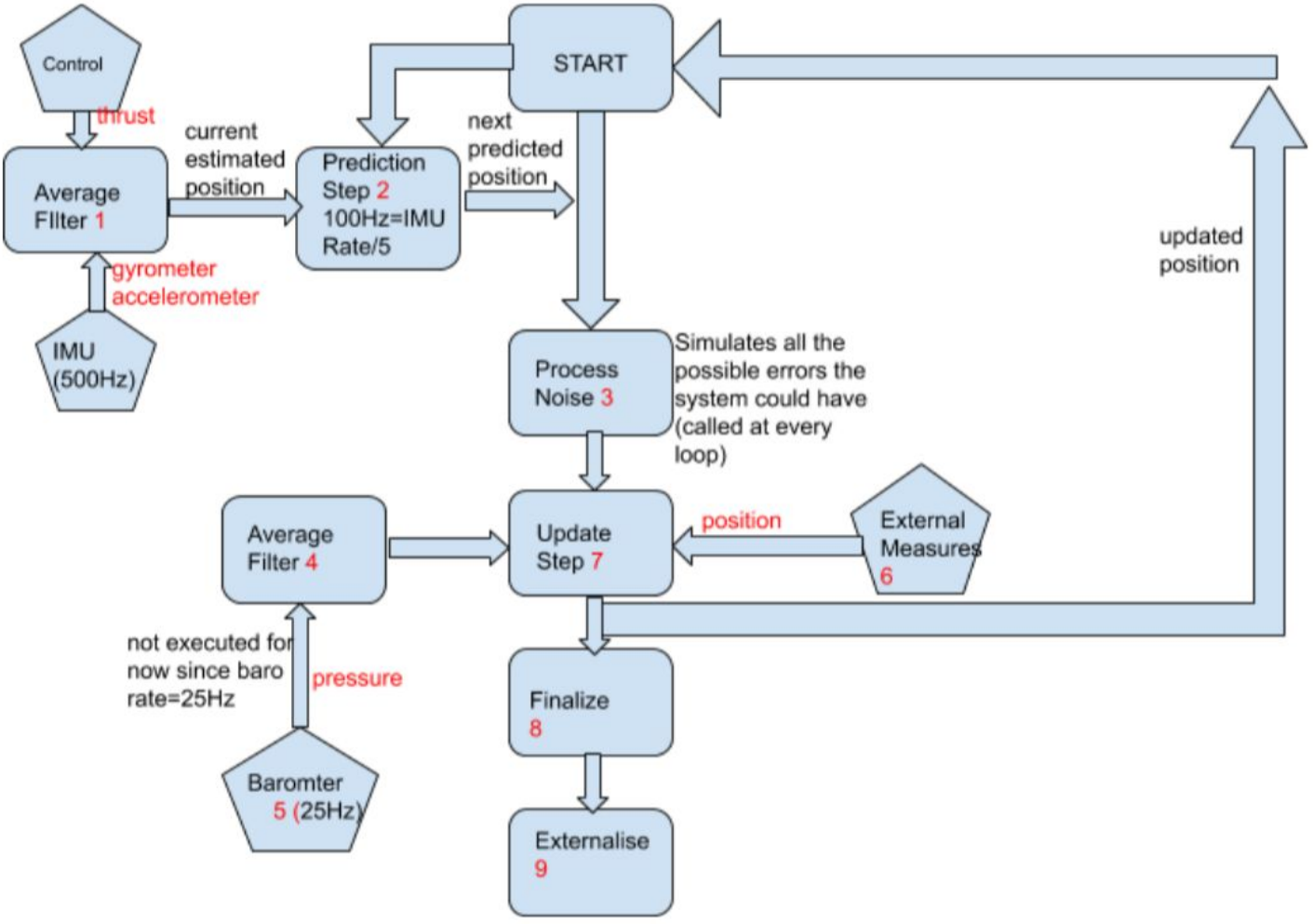
One example of interest can be the continuous availability of information such that the position or the velocity of an object relative to its position, including eventually relative error such as the standard deviation or noise.

That's exactly the kind of usage the Crazyflie 2.0 makes out of it.

The code of the Kalman Estimator manipulates the state vector and control it according to sensors and external measurements mathematical model.
The states included in the vector are position (x,y,z directions), velocity (px,py,pz) and attitude (d0,d1,d2).

## THEORY

The Kalman filter comports two main steps performed successively: the prediction step and the update step.
The prediction step tries to anticipate the motion of the drone based on control and command and mathematical model while the update step tries to correct the prediction by including measurements and observations.




Kalman Filter Architecture

## ARCHITECTURE

### 1.Average Filter

The Average filter is implemented by summing all of the measurements received by the IMU and Control sensors, and normalising them by a counter incremented each time a new data is received from the sensors.

### 2.Prediction step

This stage consists into converting the instant estimated position from the sensors into the next guess. According to a set of equations and matrices, this phase “predicts” the following points in space (x,y,z) the drone will occupy.

### 3.Process Noise
The process noise simulates all the possible errors the system could suffer from.

### 4.Average filter
Same as before except that it averages the data from the barometer sensor.

### 5.Barometer
Measures the pressure.

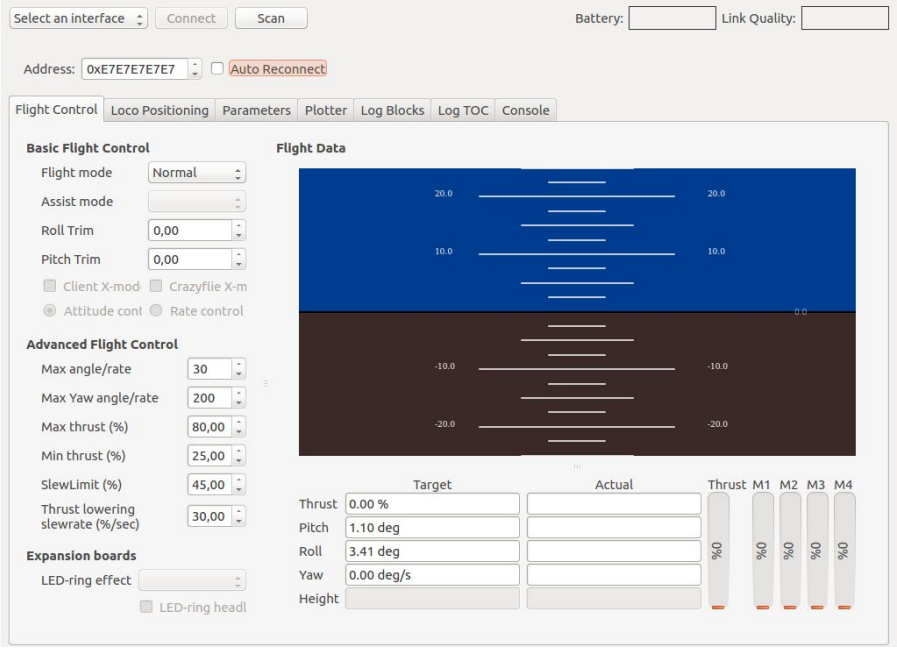
### 6.External Measures
As this depends on which type of sensors we have, we'll analyse only some of them. But the process remains similar as the goal of this stage is to sample external measurements sensors in order to include them in the update step.

### 7.Update Step
The update step is the stage that will "correct" the estimation done by the prediction step by taking into account additional data as well as the environment noise.

# DIFFERENT INTERACTIONS

## -The Client

Bitcraze Client Software Interface




The client is a UI connected to the crazyflie and allows the user to control it and get telemetry back from it.

This UI allows to read data from the different sensors connected to the crazyflie.

In this way, we can say the client is the software that enables the user to control the flight of the device.
The Crazyflie PC client, is what we consider the reference client.

It supports connecting one Crazyflie using the Crazyradio (PA) dongle or direct USB connection to Crazyflie 2.0. It supports the full Crazyflie telemetry (ie. log), parameters (ie. params) and firmware update.
It has support for all the Crazyflie 2.0 deck that can use client support.
It is also configuring the joystick configuration in order to pilot the drone.

## -The code interface

The code is an open source one available on GitHub in several programming languages, C for the stabiliser (we worked on eclipse), Python for the client.

The code is implementing functions of flight of the crazyflie as well as the sensors datas detection and their coordination.

```
        pk = CRTPPacket()
        pk.port = CRTPPort.COMMANDER_GENERIC
        pk.data = struct.pack('<Bffff', TYPE_ZDISTANCE,
                              roll, pitch, yawrate, zdistance)
        self._cf.send_packet(pk)

    def send_hover_setpoint(self, vx, vy, yawrate, zdistance):
        """
        Control mode where the height is send as an absolute setpoint (intended
        to be the distance to the surface under the Crazflie).

        vx and vy are in m/s
        yawrate is in degrees/s
        """
        pk = CRTPPacket()
        pk.port = CRTPPort.COMMANDER_GENERIC
        pk.data = struct.pack('<Bffff', TYPE_HOVER,
                              vx, vy, yawrate, zdistance)
        self._cf.send_packet(pk)
```

## -The joystick


The joystick is for the user, the tool to direct the crazyflie.
As stated earlier, the commands are set in the client interface, and the datas sensed are read directly in the client interface.

# COMMUNICATING WITH THE CRAZYFLIE

Communicating with the Crazyflie over radio is done using an ad-hoc protocol, called CRTP (Crazy RealTime protocol).
The client's joystick driver translates commands produced by the user to CRTP commands which are sent to the Crazyflie. It is also possible to send CRTP commands directly to the Crazyflie without using a control device. This can be used in order to fly in a pre-defined trajectory.
The Crazyflie's control loop reads and performs those commands.
There are several high level libraries in github, implemented in python, which wrap the CRTP and provide an extensive API for common actions.

# CLIENT PYTHON CODE AND MODIFICATIONS

We noticed that there are 2 possible ways to achieve our goal:
First is to dive deep into the Crazyflie flight control mechanism, monitor the Crazyflie's status, and inject motor commands directly.
This approach will give us a very direct, low level control over the Crazyflie, but at significant costs - This approach is very complex and is very challenging thechnicaly. It also breaks the architecture of the flight controller - we would have to bypass or re-implement much of the existing controller code.

The second way to do it is to treat the Crazyflie flight controller as a 'black box' by using the CRTP protocol to generate high level commands, and letting the Crazyflie 'work its magic'.
In this approach we will have to monitor the Crazyflie's status (namely pitch and roll angles), and if it deviates more than a designated threshold it should take off, stabilize, fly a trajectory and land.

The second approach is definitely easier, cleaner and better utilises existing code (assuming the Crazyflie behaves as expected). Therefore this approach was chosen.

# - MODIFICATIONS

Here's what we changed in order to achieve our goal:

In the crazyflie-client-python code, the gui.py file initialises the input device and enters an infinite loop:
main_window = MainUI() //init is done inside here (l.159)
main_window.show()

When the input device is selected, a daemon thread starts. This will poll the joystick for data which is modified and translated to JSON (CRTP protocol to communicate with the drone).
We want to start a similar thread for the autonomous takeoff that will monitor the state of the drone. And if any of the parameters exceeds the threshold, it will trigger an appropriate flight plan, stabilize and land.

We created a loop called MonitorThread that acts as above, in the following :
**crazyflie-clients-python/src/cfclient/ui/main.py**

All the explanations are noted in green in the following code.

The entire code section that follows had been added to the former path, this way we could achieve hover mode from a random initial position, angle, velocity...

```
def enable_monitor(self):
```
define a new function in order to activate the monitor Thread

```
        import time
        from threading import Thread


        class MonitorThread(Thread):
```
Create a new class called MonitorThread which inherits from threading.Thread

```
            THRESHOLD = 10
```
The threshold angle to be used (degree)

```
            POLL_INTERVAL = 2
```
Angles polling interval (sec)

```
            def __init__(self, main):
                super(MonitorThread, self).__init__()
                self.main = main
```
'main' is the UI object. it is used as an entry point to access needed variables

```
            def run(self):
                def _get_flight_tab():
                    for tab in self.main.loadedTabs:
                        if tab.tabName == "Flight Control":
                            return tab
                    raise Exception("Flight tab not found")
```

The UI's flight tab enables access to the actual pitch and roll angles

```
                while True:
                    try:
                        if len(Config().get("link_uri")) > 0:
                            uri = Config().get("link_uri")
                            break
                    except KeyError:
                        time.sleep(1)
```

Wait for the client to connect to the Crazyflie

```
                print("DEBUG: URI = ".format(uri))
                flight_tab = _get_flight_tab()

                while True:
                    time.sleep(MonitorThread.POLL_INTERVAL)
                    if flight_tab.actualRoll.displayText() and
abs(float(flight_tab.actualRoll.displayText())) \
                            > MonitorThread.THRESHOLD or \
                            flight_tab.actualPitch.displayText() and
abs(float(flight_tab.actualPitch.displayText())) \
                            > MonitorThread.THRESHOLD:
                        print("###Threshold exceeded - prepare for
takeoff!###")
```

```
                    for _ in range(10):
self.main.cf.commander.send_hover_setpoint(0,0,0,0.5)
                        time.sleep(0.1)

                    self.main.cf.commander.send_stop_setpoint()

                    break

        monitor = MonitorThread(self)
        monitor.setDaemon(True)
        monitor.start()
```

# TESTS AND RESULTS

As our project is about unmanned flying vehicles, it's difficult to show paper based simulations.
But on the power-point presentation of the project, you'll be able to see that we indeed managed to get the drone flying off one's hand, starting with random initial conditions (angles, tilting, height, velocity…) and stabilising.

# CONCLUSION

## - WRAP UP

-Analysis of the different sensors and collected datas, read through the client.

-Understanding of the code and of the different interactions governing the Crazyflie drone.

-Modification of the code in order to reach hover mode as soon as the sensors read datas overcoming some predefined threshold -> improve stability

-Setting of an anticipated trajectory and checking, landing.

-Finally, this project enabled us to get familiar with the UAV (unmanned aerial vehicles) world and to deal with programming language analysis and modifications. It also provided us skills in experimenting: tries, tests, verifications.

## - PROPOSALS FOR FURTHER WORK

The benefit of this project is the ease of access to the open source code.

For the ones who got interested in this project and who wish to extend it further, there exist infinite ways to improve and explore new features of the Crazyflie 2.0.

The Crazyflie 2.0 was designed to be as flexible as possible. In addition to the default functionality that comes with it, there is a flexible expansion interface where a variety of expansion decks can be attached, both on the top and the bottom of the Crazyflie 2.0. From this expansion interface the user can access buses such as UART, I2C and SPI as well as PWM, analog in/out and GPIO.

# Appendix: debugging configurations

For efficiency concerns, we decided to debug the code through a native (ubuntu) rather than through a VM (Virtual Machine).
Here are the steps to follow in order to be able to debug the code:

1. Modify the code this way:
debug 1
cload 1
ESTIMATOR ?=kalman
CONTROLLER?= Mellinger
2. Type the following commands on the terminal:
make clean
make
3. Put the crazyflie in boot loader mode (Press approximately 5 seconds on the on button up until you see a blue light blinking).
4. Type the following commands on the terminal:
make flash
5.reset the HW (STMicroelectronics and drone)
6. debug

Note that there is an automatic breakpoint set at the beginning of the main loop, so think about removing it and set your own in order the reach the inner code.

# REFERENCES

Through this paper, a lot of different sources were used; in this part of the book we mention most of them:


**Hyperlink:**

*https://www.bitcraze.io*

*https://github.com/bitcraze/crazyflie-lib-python/blob/master/examples/flowsequenceSync.py#L59*

*https://wiki.bitcraze.io/projects:crazyflie2:debugadapter:index*

*http://www.openstm32.org/forumthread5033*

*https://arc.aiaa.org/doi/10.2514/1.G000848*

**Books:**

*https://www.ensta-bretagne.fr/jaulin/rapport2018_poirier.pdf*

**Open Courses:**

*Coursera, Robotics Flight*

*Estimation Course :*
*https://drive.google.com/file/d/1pUrLW4iKYh6siV8WFv-nV5fyvYD5Lkft/view*

**Data Sheets:**

*https://www.st.com/content/ccc/resource/technical/document/user_manual/group0/26/49/90/2e/33/0d/4a/da/DM00244518/files/DM00244518.pdf/jcr:content/translations/en.DM00244518.pdf*

# THANK YOU !