# Semi-Autonomous Indoor Drones

**Bernardo Rocha**
Instituto Superior Técnico
Lisbon, Portugal
bernardorocha@tecnico.ulisboa.pt

## ABSTRACT

The latest advances in Micro Aerial Vehicle (MAV) manufacturing have made these tiny robots very good development tools for both researchers and students. This work[1] aims to provide a system that students from IST, namely in Computer Engineering courses, can easily deploy and use in a laboratory environment to control a MAV, using their own computer and a python API, here described. This system is built on top of the Crazyflie platform and is accompanied by a set of laboratory guides for students to follow during laboratory classes. The topics covered range from manual navigation to mapping, autonomous exploration and drone-to-drone interaction. Experimental results show the system's ability to perform in complex indoor environments.

## KEYWORDS

MAV; Crazyflie; Indoor environment; Mapping; Exploration.

## 1 INTRODUCTION

Unmanned aerial vehicles (UAV), commonly known as drones, have been used for a long time in several civilian and military domains, including weather observation, surveillance, search and rescue operations or civil engineering inspections. In fact, as these robots' capabilities keep increasing, they also keep getting smaller and cheaper, and new uses for them continue to be explored. More recently, research has been done towards the development of small human-friendly drones that can fly autonomously in indoor environments. Unlike a regular-sized UAV, a micro aerial vehicle (MAV) can operate in confined and GPS-denied environments, and since it is usually a low-cost solution, it can be easily replaced in case of damage or total loss. Moreover, they can be an excellent development tool for students and researchers, in a big diversity of fields such as embedded systems, robotics or control theory. Because of their small size and weight, it is very safe to use them in a laboratory environment, including near people. More specifically, they make up a new and engaging learning opportunity for university students in the Computer Systems branch, as they would able to modify both software and hardware of the drone and build their own drone-oriented applications during laboratory classes.

[1]Supervised by Prof. Alberto Manuel Ramos da Cunha

### Objectives

The main focus of this work is in providing a full-fledged system that students and researchers from Computer Engineering can easily use and deploy in laboratory environment, such as the facilities at Departamento de Engenharia Informática (DEI) in IST, to develop drone-oriented applications. The system aims to take advantage of the unique and engaging learning opportunities provided by MAVs, specifically in the Computer Systems / Cyber-Physical Systems branch, as they are complex cyber-physical systems, with real-time sensing and control requirements, and at the same time, highly resource-constrained systems, as their small size does not allow powerful sensors, computing units and batteries. To that end, a set of 3 laboratory guides will be produced that will give students the opportunity to tackle these unique challenges, using MAVs, during laboratory classes of a subject in the field. For them to solve the proposed exercises, a high-level API is here developed and documented, which offers a few high-level calls that allow easy control of the drones. Finally, this work also aims to provide a solid base for future students that wish to use these drones for their own projects, or otherwise anyone that wants to extend the system here proposed in any way. The goals here described are addressed mainly as a software challenge rather than focus on enhancing or automating hardware.

## 2 RELATED WORK

There are some MAV based systems already being used in research and education at university level. The SLIM [1] system, for instance, was built to enable a big diversity of use cases to be implemented. It has been used only in multiple research projects but also lecture courses and student projects and competitions. In [2], a system to fly autonomously in complex and unknown environments using ground control station architecture is presented. This work was important to help understand the lower-level modules necessary to produce position estimates. In the system here presented though, these modules are provided off-the-shelf in the drone's default firmware build. It was also presented some research [3][4] and education [5][6] systems involving the Crazyflie platform. They showcase the platform's potential to run swarming architectures and build high-level applications. All in all, these systems helped to converge into the topics to be addressed in laboratory classes, such as Mapping, Exploration and

Swarming (later simplified into drone-to-drone communication).

Most of the presented techniques and technologies for indoor localization assume one or more transmitters that communicate a signal to one or more receivers [7]. Most of the time, this implies that this set of transmitters (or receivers) that need to be previously installed in the room, which does not promote easy deployability. While this does not mean that a mobile robot system would not benefit from that kind of setup, especially in a scenario where multiple drones need to be aware of each other, a more device-centered approach needs to be taken, where most of the hardware required is in the robot. In doing so, there is a tradeoff between localization accuracy and deployability,
Finally, some research is done on topics addressed in the laboratory guides, with the goal to provide and high-level implementation through the developed API. In particular, the occupancy grid mapping [8] and frontier-based exploration [9] algorithms were implemented, as mapping and exploration were not provided off-the-shelf by the Crazyflie platform.

## 3 SYSTEM ARCHITECTURE

This is the system that will be used by each group at laboratory classes. It comprehends a Crazyflie 2.1 quadcopter, a ground control station and a python API. Each component will be described over the next sections. In the last section some design choices are discussed, including the choice of the Crazyflie platform.

The Crazyflie is a small and versatile MAV developed with research and education purposes in mind. *Bitcraze AB* [10], the company that develops and manufactures the Crazyflie, also maintains a wide ecosystem of expansion decks, clients and development tools that enable rapid development, flexibility and ease of use. In addition, all their projects are open source with extensive documentation available. The system that will be used in laboratory context has three main components, the Crazyflie drone, a corresponding ground control station, which is a computer enabled by a Crazyradio PA USB dongle, and a python API, that is extending the functionality of the python library *cflib* [11], made available by *Bitcraze*. It provides a full-fledged solution that can be easily deployed in a laboratory and allows students to focus on software development while having to respect the constraints and challenges adjacent to controlling a flying robot. The architecture is depicted in Figure 1.

### Crazyflie

The Crazyflie 2.1 drone (in Figure 2) contains an EEPROM memory for storing configuration parameters and a 10-DOF IMU with accelerometer, gyroscope, magnetometer and a high precision pressure sensor. The MAV is also equipped with low latency/long-range radio and Bluetooth LE, which gives the user the option of either downloading the official app and use a mobile device as a controller or, in combination with the Crazyradio PA, flying with a game controller. This is the fastest way to start flying right out of the box, but it’s not how students will be controlling it. The firmware of the drone is written in C and can be easily modified and flashed over the radio. The drone weighs only 27g and it's so small that it fits in the palm of a hand. Despite its size it is designed to be durable, as it will, in most cases, remain intact in minor crashes and break at the cheapest components, like propellers and motor mounts, in the event of a major accident (as verified during the development of this system). These characteristics make it ideal for flying indoors. Although small, the four 7mm coreless DC-motors in the Crazyflie grant it a maximum take-off weight of 42g, which enables it to carry multiple expansion decks for extra capabilities in sensing, positioning or visualization. There is an extensive range of decks available, but the platform is also designed to make it easy to design and add custom decks, enabling the user to use sensors and other devices on the platform. From the expansion interface the user can access buses such as UART, I2C and SPI as well as PWM, analog in/out and GPIO. In the system presented in this work, two expansion decks will be used.

The first, the flow deck v2, is attached on the bottom of the drone and gives the ability of performing improved relative localization. The deck achieves this using two

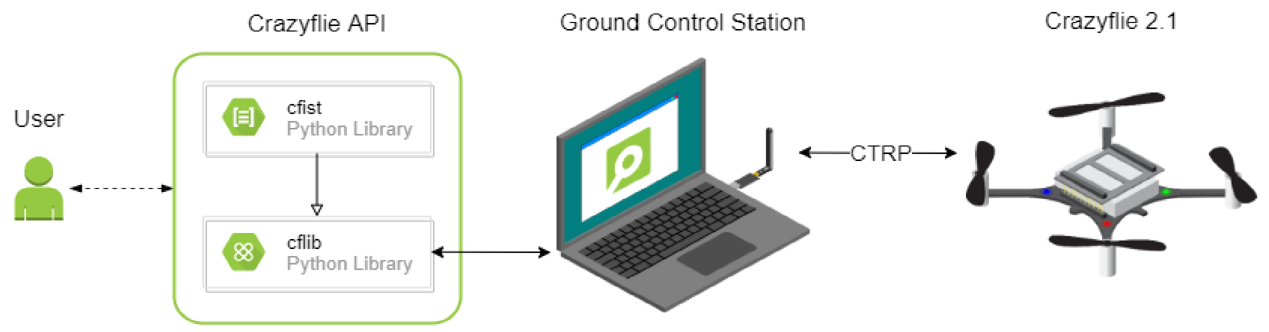


**Figure 1- System Architecture**

sensors. First, a PMW3901 optical flow sensor that measures movements in relation to the ground. Internally, it uses a low-resolution camera and predictive algorithms that try to detect motion of surfaces. The second sensor is a VL53L1x TOF sensor that measures the distance to the ground with high precision. This is a laser-ranging sensor that can accurately measure distances up to 4 m with a ranging frequency up to 50Hz. The flow deck gives much more control over the drone, as it not only can now be pre-programmed to fly specific distances in any direction but also greatly improves overall flying stability. It plays a key role in pose estimation. Usually, pose estimation using only odometry sensors (dead reckoning) is too unreliable to be considered, as the relative position will drift too much when considering only accelerometer or gyroscope sensors. The flow deck greatly improves this estimation process and allows dead reckoning to be considered in systems that require location updates, but do not need highly accurate estimations, like this one. The second deck it is called multi-ranger. It is attached to the top of the drone and adds the ability to detect obstacles around the Crazyflie.
It contains 5 VL53L1x TOF sensors (the same as in the flow deck) that will measure distances in the directions front, back, left, right and up. This deck is essential to perform collision avoidance or work on environment-aware problems like mapping a room. With these two decks, the Crazyflie can now be an interactive autonomous platform. This setup can be seen in Figure 2.

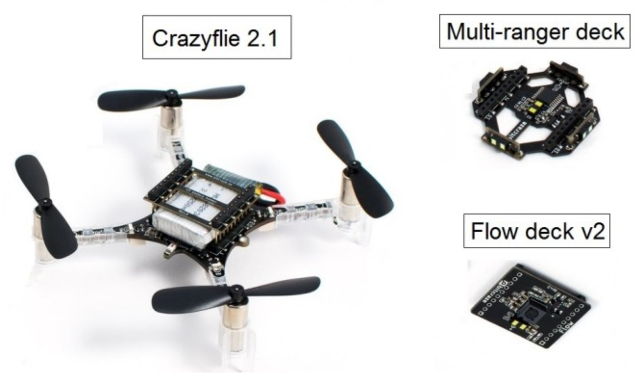


**Figure 2 - Crazyflie 2.1 and expansion decks**

## Ground Control Station

A ground control station (GCS) is a land-based infrastructure that has the necessary hardware and software for human control of UAVs. All commands to the drone and all readings from it, go through the GCS, to the human pilot. In the context of this work, a GCS will be a laboratory computer or laptop, plugged with a USB dongle called Crazyradio PA. This dongle relays the CTRP (Crazy Real Time Protocol) from the python library to and from the Crazyflie. The CTRP is a high-level communication protocol developed by *Bitcraze* to send and receive data in either direction, but in most cases the communication will be driven from the host, the CGS.

When flying using the python API, the Crazyflie relies on constant communication with the GCS. The GCS is the one running the application and has the obligation to continuously send control commands to keep the drone flying. The drone will autonomously kill its motors if it stops detecting the radio signal from the Crazyradio PA. Such behavior is implemented by default in the firmware, for safety reasons, and it's why the drone is not considered “fully autonomous” as it requires constant communication with the host. In order to grant higher degree of autonomy, one could remove the GCS and python API entirely from the architecture and do all the programming in firmware. However, there are two big disadvantages that invalidate this option. First, because of usability. Programming directly in the drone’s firmware would require much deeper knowledge of how the firmware is structured and working with a programming language that is less user-friendly than python, C. And secondly, because of performance reasons. Having a GCS always ready allows the drone to offload computing power when performing heavy tasks, which is especially relevant in such a small and low-cost device that, consequently, has limited computational power, storage and energy.

## Crazyflie API

The Crazyflie API is the gateway intended for users to interact with the Crazyflie, which assumes they have some basic knowledge of the python programming language. *Bitcraze* maintains a python library called cflib [11], which is the main connection point for programs and scripts to communicate with the drone. The version used in this system is 0.1.11. This library contains all the core functionality needed to implement a simple semi-autonomous mission, such as how to connect to a Crazyflie using an URI that identifies a communication link; how to set up logging configurations that will request the drone firmware to send specific variables at a predefined time interval (in ms) to the GCS, like a reading from a sensor; how to read and set parameters on the drone (they differ from the logging as the variable is not changed by the Crazyflie but by the client and is not read periodically) and how to send control set-point commands.

However, the laboratory guides will naturally start to introduce slightly more complex problems over time. Sometimes, those problems may require knowledge that is outside of what is intended for a computer engineering student to know. This was the case with the problem of building a floor plant, a 2D representation of the room where the drone is flying, introduced in laboratory guide 2. Other times it was just necessary to enforce some common rules that all groups should obey and agree, mainly for safety reasons. This was particularly important in laboratory guide 3, where drones from multiple groups share the same airspace and must navigate in the same airway, as a functional, yet very basic, intelligent transportation system (ITS). These two factors were the main motivation to develop *cfist*, a python library that extends the functionality of the *cflib*, with everything that students would need to solve the exercises proposed in the laboratory guides, which includes building a floor plan, using an occupancy grid mapping algorithm; running and developing a custom autopilot, using frontier-based exploration; detecting other Crazyflies, using drone-to-drone communication and basic Traffic Management. This functionality is explained in the next section. As for the *cflib*, is still the only way of communicating with the GCS, which means that users are still expected to use all functionality made available from that library. Together, these libraries make up the API that students will use and learn from to build their first drone applications.

## 4 LABORATORY PROJECTS

In this section the implementation of the developed components is described. This includes the laboratory guides that were written, the *cfist* library developed and the modified firmware that was flashed to all the drones.

Each guide is planned for a 90 min laboratory class and all follow a similar structure, which includes at least these sections:

- **Goal**: This tells right away to the students what they will be doing and gives a quick idea of what will be needed to accomplish that task.
- **Crazyflie API**: This is a briefing of the functionality they will need to use or implement to accomplish the task. For each functionality, it is always included a description of what it does and references to usage examples and to the respective implementation in the API.
- **Safety Warnings**: There will always be some risk involved. This section includes some practical measures that students must take, before and during flight, to lower that risk and avoid injure themselves and the equipment.
- **Exercise**: A description of the task to be developed divided into up to 3 smaller exercises.

### Laboratory Guide 1

In this first laboratory guide [12], the goal is to introduce students to the Crazyflie development environment. This includes the installation process of everything needed for all the guides, the run of a demonstration script and the development of their first flight script.

Here, the main contribution from the *cfist* library is the *Manpilot* module. This module contains the *KeyboardPilot* class, which was created to make it easy to control a Crazyflie using the keyboard of the computer, so that students don't need to import some 3rd party library themselves, which would most likely result in a lot of repeated code to create the key maps. Instead, the API has a pre-defined key map that students can extend by adding their own callbacks that will be called at a key press that they choose. It is still the user responsibility to send the control command to the drone. The class can be used as a context manager to start and stop the observer that is listening for keyboard input automatically.

### Laboratory Guide 2

This laboratory guide [13] focus on three components: Logging, Mapping and Autopilot. The *log* module is provided by the *cflib* and is one of the main features provided by this library. It is the standard way of reading values from the Crazyflie firmware. In this laboratory guide it will be used to monitor the drone's battery and automatically land based on the drone's response. The *Mapping* module of the *cfist* will be used by students to build a 2D representation of the room while flying the drone. It is very easy to use; with only a couple lines of code a map can be created and update itself passively when it needs to. The *Autopilot* module, also from the *cfist* library, gives the drone the ability of understanding where it needs to go, solely based on its perception of the world and a pre-programmed algorithm. It will output a command that can be redirected to the Crazyflie instead of using the command from the *KeyboardPilot* as in the previous guide. For the students, this is where most of the work will be focused on.

The *Mapping* module is responsible for the implementation of an occupancy grid mapping algorithm [8], and the drawing functionality of that map, using a 3rd party library[2].

The *Autopilot* module was created to be a simple way of implementing autonomous behavior in the Crazyflie. Each object needs to have a method *run*, which receives multi-ranger measurements from the Crazyflie and outputs a command ready to be forwarded to the Crazyflie, like the *KeyboardPilot* does. There is a predefined subclass called *Follower*, which implements a simple wall-follower algorithm. It used for usage example and later in guide 3. There is also another type of *Autopilot* called *Explorer*. Its goal will be to conduct a frontier-based exploration, as proposed in [9]. Students will be able to use to generate in-map goals that they can then navigate to. There are also other useful methods that allow, for example, calculating the distance to the current goal or the yaw needed to be facing the goal.

### Laboratory Guide 3

In this lab [14], each group will be simulating a Drone Delivery Application. Delivery Services are one of the most promising applications for drones and here students will have the opportunity to implement their own system, while being part of a larger intelligent transportation system (ITS), where every group will have to follow basic traffic control rules. This is of course a simplistic version of the

[2] Matplotlib: https://matplotlib.org/. Last Accessed: 1 September 2020

system, which doesn't have to deal with some of the biggest challenges that real-world systems that are currently being developed have, such as flying in very complex environments, like crowded cities, or compliance with the local laws. This is also a simulation because it will be done indoors, in a controlled environment with permanent human monitoring. The behavior of picking up and dropping off a package will also be simulated, although a system that could physically lift a light package using the Crazyflie would be an interesting extension to this project.

When designing such a system, where multiple Crazyflies will be flying at the same time and sharing the same airspace, there is a need to enforce some common traffic rules to help reduce the probability of traffic congestions and collisions between drones, much like common traffic rules in the road greatly reduce car accidents for every driver.

On one hand a semi-automated detection mechanism was developed, which allows students to ask their drone if they "see" other drones nearby. This is necessary because since the Crazyflie is such a small drone, the multi-ranger sensors can easily miss other nearby drones. They can then act depending on the estimated distance to the detected drone, as well as who is being detected. There are two sides to the implementation of this functionality. The *Radar* module of the API, which exposes the functionality just mentioned to the user, and the firmware of the drone[12], which had to be modified so that it could start broadcasting messages that other drones could receive and send to its respective GCS. Figure 3 illustrates how these parts interact with each other in a typical use case, when the user tries to detect neighboring drones.

On the other hand, there is still the need to enforce the common rules that will decrease the risk of two drones even getting close to each other. This is achieved using the *DDS* module. *DDS* stands for Drone Delivery Service, and besides providing the route that a drone should take when it is travelling to a destination, it also provides basic logistics, like methods to manage locations available for delivery that every application might need. These locations are relative to each system.

Globally, this results in an ITS where each group's system is unaware of the other systems until the moment their drones need to avoid each other, since they still need to operate in the same environment. On one hand this allows systems implemented in different ways to be able to live together. On the other hand, the responsibility of enforcing common rules and check for drone collisions falls upon each individual GCS, which is susceptible to implementation error from the user.

## 5 EVALUATION

This section describes the experiments conducted to evaluate and validate the suggested system. Core features of the system will be tested: mapping, exploration and drone-to-drone communication. All tests were performed in an Asus K550J laptop, with an Intel Core i7-4710HQ CPU @2.50GHz and 8.00 GB of RAM. As for software versions, the *cflib* library is v.0.1.11, *cfist* is v.0.0.1 and the firmware is a modified version of v.2020.06.

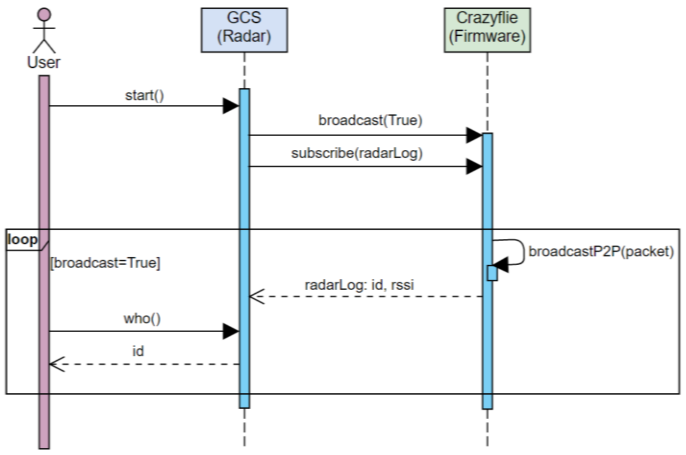


**Figure 3 - Sequence diagram for detecting a neighboring drone**

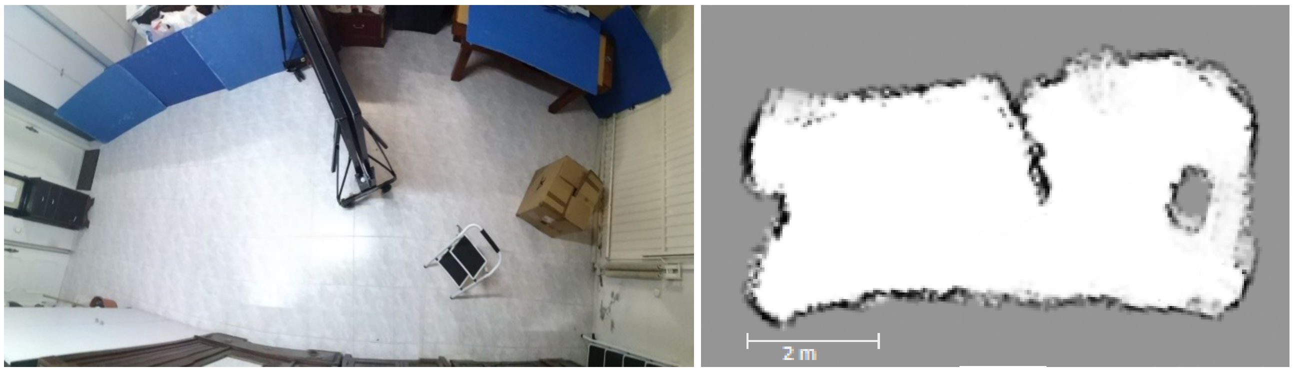
### Mapping

The performance of the occupancy grid mapping algorithm will now be tested. First, the quality of the generated map will be evaluated by manually flying a Crazyflie drone in an indoor environment and comparing it with the ground-truth top view of this room. And secondly, it will be tested how the algorithm parameters can vary the resulting quality of the map as well as how fast it can be generated. These parameters are the cell size of the grid (which define the grid resolution) and the sensor FOV.

The testing environment can be seen in Figure 4 (a), from where it's possible to distinguish two different zones. A wide-open obstacle-free space (on the left) and a smaller cluttered space (on the right). This is to evaluate how the complexity of the environment affects the quality of the generated map. The space is approximately 6,5 m x 2,5 m. All the experiments described will have the Crazyflie flying at 0,2 m/s.

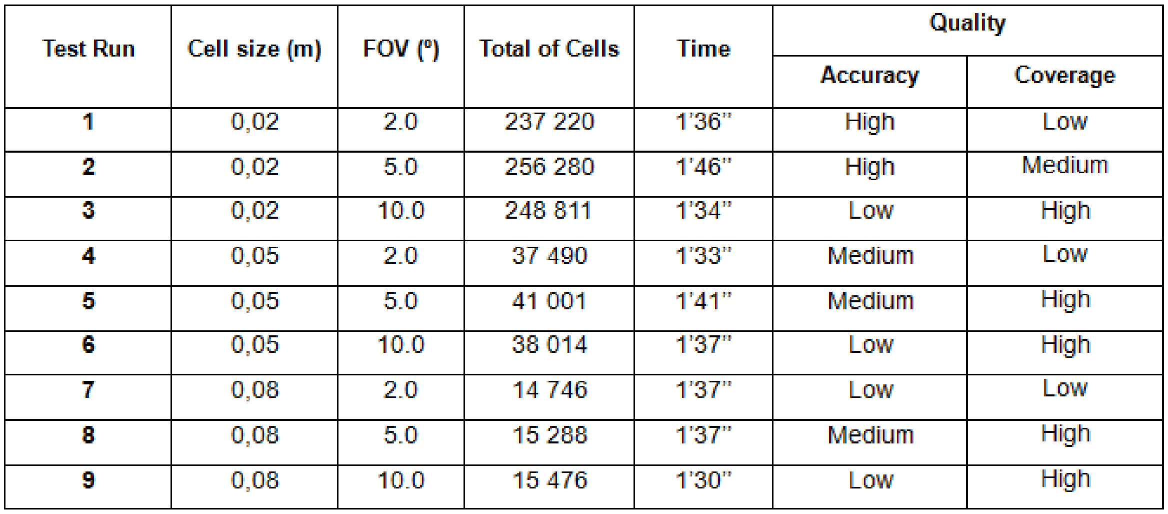
From the first experiment, the map that resulted from manually flying a Crazyflie around the room for 2'30" is presented in Figure 4 (b). A side-by-side comparison shows that the map is fairly accurate, as it is possible to distinguish the general shape of the room. One thing that stands out however, it's how the small stairs, as well as the table legs, are completed ignored. This is because, at the height that the drone was flying, the obstacles are so thin that even if a few cells register an obstacle a few times, most of the time the obstacle is dismissed and so the cell is considered more likely to be free than occupied. In addition, it was possible to see clear performance differences while navigating in the two zones of the room, namely, the clustered zone was mapped much faster because of its smaller size.

In the second experiment, to evaluate how the cell size and FOV would vary the quality of the resulting map, multiple test runs were executed where only one of these



(a) (b)

**Figure 4 - Top view of testing room (a) and occupancy grid map generated during manual flight (b)**



| Test Run | Cell size (m) | FOV (º) | Total of Cells | Time | Quality | |
|---|---|---|---|---|---|---|
| | | | | | Accuracy | Coverage |
| **1** | 0,02 | 2.0 | 237 220 | 1'36" | High | Low |
| **2** | 0,02 | 5.0 | 256 280 | 1'46" | High | Medium |
| **3** | 0,02 | 10.0 | 248 811 | 1'34" | Low | High |
| **4** | 0,05 | 2.0 | 37 490 | 1'33" | Medium | Low |
| **5** | 0,05 | 5.0 | 41 001 | 1'41" | Medium | High |
| **6** | 0,05 | 10.0 | 38 014 | 1'37" | Low | High |
| **7** | 0,08 | 2.0 | 14 746 | 1'37" | Low | Low |
| **8** | 0,08 | 5.0 | 15 288 | 1'37" | Medium | High |
| **9** | 0,08 | 10.0 | 15 476 | 1'30" | Low | High |

**Figure 5 - Occupancy grid mapping performance experimental results**

parameters was changed. The cell size values tested were 0,02 m, 0,05 m and 0,08 m and FOV values were 2.0º, 5.0º and 10º. Because of the probabilistic nature of occupancy grid mapping algorithms, the longer the Crazyflies is flying the more certain it will be about the occupancy probability of the cells it sees, which, assuming perfect pose estimation, would translate to a more accurate map. For this reason, to ensure accurate test trials, the time taken during each run was approximately the same. The results are shown in Figure 5.

By analyzing the table, the first conclusions we can draw are that, generally speaking, map accuracy increases with finer grids and coverage speed increases with a bigger FOV. This is to be expected, as a smaller cell size will produce a more realistic map and a wider FOV will assume a lot more cells to be occupied. Secondly, it is also possible to see that, generally speaking, there is a tradeoff between accuracy and coverage. This is because on one hand, smaller cells mean more cells that the drone will need to "see" and on another hand, a wider FOV means that the probability of assuming wrongly occupied cells greatly increases. Third, it is possible to see that using a FOV value as big as 10º always make the map too inaccurate, regardless of cell size. Finally, we can conclude that the optimal parameters to express the quality requirements that were stated before are a FOV of around 5.0º and a cell size that can vary between 0.05 m and 0.08 m. If someone who is looking to extend this system needs a high accuracy representation, it is recommended to decrease the cell size, while maintain a similar FOV value. This will of course require more time to fully map the same area.

## Exploration

Now, the performance of the frontier detection algorithm will be tested. This is the algorithm used by students in laboratory guide 2 to calculate a goal during autonomous exploration. To evaluate the quality of the generated goals, the drone will be flying in the same testing environment with the following behavior:

1. Take-off;
2. Do a 360º scan, to maximize information gained;
3. Run the frontier detection algorithm, which will generate an in-map goal;
4. If a goal is returned, manually fly there and repeat from first step;
5. Otherwise, the map is considered fully discovered. Land.

Figure 6 shows the generated occupancy grid map. Generated goals are marked as blue dots and numbered by order of appearance, defining the path traveled. Mark "0" is the take-off position. The presented map was generated in 1'54'' time with a grid resolution of 5 cm per cell and FOV parameter at 4.0º. When analyzing the generated map, there are two metrics here defined to evaluate the algorithm performance. The quantity of goals, in the sense that the ideal map would contain the minimum amount of goals possible that allow to fully discover a room. And the quality of those goals, in the sense that goals should be generated in a position that offers as much information gain as possible. As for quantity, the amount of goals generated look very good. Because of the sensor maximum range is capped at 3 m (for more accuracy), each new goal would be ideally at approximately that distance, to ensure the minimum goals generated and fastest map generation. This is possible to verify by looking at the estimated distance between a goal and the next. As for quality, there are some goals better than others. Goal 1 would ideally be place at the center of the wide zone, however it is place very near the take-off position. This can be attributed to the fact that the scan that was done in position 0 gained very little information because of its cornered position, so the drone didn't have enough information to decide the best position to scan the zone. Goals number 3 and 4 are particularly interesting. After the scan done at 2, the algorithm saw the obstacle in the middle of the cluttered zone and defined two frontier regions, one for each side of the box. Because the goals are such in a good position there was no need to have more than two goals to learn all the map of the cluttered zone.

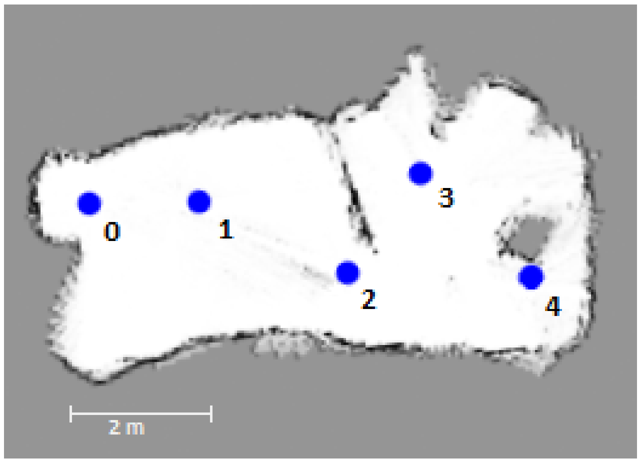


**Figure 6 - Occupancy grid map generated during exploration. Blue dots are goals generated by frontier detection algorithm.**

**Drone-to-drone communication**

In this section, multiple requirements are being validated. In a first experiment, only the performance of the RSSI distance estimation will be evaluated. Then the experiment is scaled, to test how many drones can be detected at the same time without compromising other requirements. Safety is also being validated, by evaluating how many collisions is the system able to avoid.

The first experiment consists on having a Crazyflie (A) hovering still, while another drone (B) slowly flies towards it at 0,2 m/s, starting from 3 meters away. When one of the drones detects it is in dangerous proximity with another drone, they both land and the distance between them is measured. Proximity is considered dangerous if the RSSI value returned is bigger than –48 dBm. This value was obtained by trial and error and its validity should also be considered when analyzing these results. This experiment is repeated 8 times to account for RSSI noise. Because the floor in this environment is very reflective, the drones will be flown higher than usual, at 0,5 m, to try to mitigate some noise in the signal propagation. The results are shown in Figure 7.

As it is possible to conclude, the accuracy of the distance estimation is quite poor, with values ranging from 23 cm to 197 cm in only 8 trials. This can be justified not only by the natural lack of accuracy of the technology in indoor environments, namely because of the multipath effect, but also because no proper signal propagation model is being used, namely one that makes use of software filters that help to counteract this effect. This is something that should be investigated in future extensions of this work. Nevertheless, at this velocity with the reference RSSI of -48 dBm, it was still possible to automatically prevent the collision in all trials. To this end, no value greater than –48 should be used, as the signal could easily be dismissed. On the other hand, values smaller than –48 dBm can quickly start to be detected in the whole room. This means that the system should not be too conservative either or it will quickly become unusable as a proximity detector. Being hovering still or in motion doesn't appear to have any effect on accuracy.

The second experiment is very similar to the one just described, but instead of only one drone (A) hovering, there will be a cluster of 3 drones. The behavior is still the same. Another drone (B) will approach and whoever detects each other first (between A and B) makes both land. The goal is to measure how difficult it is for A to detect B, since it must process the messages coming from the other two drones of the cluster too. The drones in the cluster were about 1 m apart in a triangle formation. Since there were only 4 Crazyflie drones available this was the maximum it was possible to scale up the experiment. Results are shown in Figure 8.

| Test Run | Distance from A to B (in cm) | Detected by |
|---|---|---|
| **1** | 156 | A |
| **2** | 90 | B |
| **3** | 195 | B |
| **4** | 54 | A |
| **5** | 170 | B |
| **6** | 197 | A |
| **7** | 58 | A |
| **8** | 23 | B |
| **Mean** | 117.88 | - |
| **Standard Deviation** | 65.02 | - |

**Figure 7 - Drone-to-Drone collision detection experimental results**

| Test Run | Distance from A to B (in cm) | Detected by |
|---|---|---|
| **1** | 174 | A |
| **2** | 73 | A |
| **3** | 187 | A |
| **4** | 35 | B |
| **5** | 188 | A |
| **6** | 165 | A |
| **7** | 81 | A |
| **8** | 126 | A |
| **Mean** | 128.63 | - |
| **Standard Deviation** | 55.28 | - |

**Figure 8 - Cluster-to-Drone collision detection experimental results**

Contrary to what was expected, detection accuracy not only did not decrease, it improved a little. The mean shows that Crazyflie B was generally being detected sooner than in the previous experiment. Not only that, but measures taken were also more consistent, as shown by the standard deviation. This may be explained by the signal propagation from the other two drones from the cluster interfering with the reflected weaker waves, coming from the multipath effect, which ultimately results into only the stronger direct signal being received. This would require further testing to be proved, as well as expertise that is not part of this work. As for the previous experiment, all collisions were automatically prevented without the need for manual intervention.

Due to the global health crisis, that occurred during most of the development period, live tests with users were not possible to conduct, in order to assess difficulties in apprehending and solving the proposed exercises, as well as obtaining more realistic time estimation. Therefore, they should be further evaluated in the future, before being used in laboratory classes. For instance, a teacher that wants to use this system in its class should evaluate the guide not only to further ensure overall correctness, but also to make any necessary modification to account for specific needs.

## 6 CONCLUSION

In this dissertation, a system to help students and researchers implement drone applications is suggested. The system is built on top of the Crazyflie platform and exposes a python API so that users can easily interact with the MAV. A set of 3 laboratory guides [12] [13] [14] were also developed, that aim to help students from IST to use this system over the course of 3 laboratory classes. In addition, an admin guide [16] has also been created to help an admin user to prepare a newly bought Crazyflie to be used in the suggested system.

Results obtained are overall very positive and show that the goals that were presented were successfully achieved. This work also serves as proof-of-concept that validates the choice of the Crazyflie platform [13] as an option for research and education, namely in the fields here addressed. Because of the broad scope of this work, it was only possible to "scratch the surface" at each one of the topics covered. To this end, the modular implementation of each main feature in this work hopes to greatly facilitate future extensions that can be made in a particular field of interest.

### System Limitations

Just like any architecture, this system has some known limitations. Some come from the Crazyflie platform itself and others from the implementation described in section 4.
As was said before, *Bitcraze* regularly maintains their libraries and firmware code. Whether it is in fixing existing bugs or delivering new features, by freezing the version of the *cflib* (v. 0.1.11) and the base firmware (without Radar support, v. 2020.06) the system will lose all those benefits. Readers that want to extend the proposed system in any way are encouraged to use the latest versions of *cflib* and the firmware instead, and test the system themselves to ensure its validity, knowing that compatibility with the presented system is not assured.

Swarming using the Crazyradio PA dongle and the *cflib* library (specifically, the *Swarm* module) provided by this system, it is only possible when controlling up to 3 or 4 Crazyflies at the same time. After that, the packet loss will start to become problematic. However, there are external projects, such as Crazyswarm [14], that allow controlling more than 15 Crazyflie drones with a single Crazyradio, by using more efficient communication schemes.

The Crazyflie platform also offers the possibility of communicating via BLE which is intended to be used with mobile phone apps. A reader interested in implementing such application should know that, using the *cflib* and firmware versions used in this system, the Crazyflie drone cannot communicate concurrently via Crazyradio PA and BLE, as there will be a small amount of packet loss that increases with the number of Crazyflies added. In addition, it is also not possible to use the Radar module developed for the system here presented, as it requires BLE to be disabled in the NRF chip in order to use P2P communication between drones. Therefore, using a Crazyflie that was previously used as part of this system requires the BLE to be activated again.

### Future Work

The system here suggested was designed to be easy to extend, either by adding new functionality to the existing modules, or by adding new modules. Some interesting suggestions that aim to either fix an existing problem or simply add a new feature are here described:

- **Radar:** To improve the Radar performance, a signal propagation model for estimating distance from RSSI radio signal could be used, along with various filtering techniques.
- **Path Planning:** Most path planning algorithms are usually tested in simulation tools and programs, instead of real robots. This system provides an easy way for developers and researchers to test their algorithms in a real flying robot. The system even provides, through the Mapping module, the drawing functionality and the occupancy grid mapping code which is commonly used to implement these algorithms.
- **Hardware Extensibility:** Adding extra sensors to the Crazyflie is a feature that holds great potential for future projects. For instance, an early idea was related to adding simple temperature and LDR sensors to the drone to make a mobile weather station that could be programmed to go take reading at specific places and predefined times of the day.

## 7 REFERENCES

[1] W. A. Isop e F. Fraundorfer, "SLIM - A Scalable and Lightweight Indoor-Navigation MAV as Research and Education Platform," em *Robotics in Education*, 2020, p. 182–195.

[2] A. Bachrach, R. He e N. Roy, "Autonomous Flight in Unknown Indoor Environments," *International Journal of Micro Air Vehicles,* vol. 1, n.º 4, pp. 217-228, 2009.

[3] E. A. Cappo, A. Desai e N. Michael, "Robust Coordinated Aerial Deployments for Theatrical Applications Given Online User Interaction via Behavior Composition," em *DARS*, 2016.

[4] B. Araki, J. Strang, S. Pohorecky, C. Qiu, T. Naegeli e D. Rus, "Multi-robot Path Planning for a Swarm of Robots that Can Both Fly," em *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 2017.

[5] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński e P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," em *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, Miedzyzdroje, Poland, 2017.

[6] J. Noronha, "Development of a swarm control platform for educational and research applications," Master's Thesis, Iowa State University, 2016.

[7] F. Zafari, A. Gkelias e K. K. Leung, "A Survey of Indoor Localization Systems and Technologies," *IEEE Communications Surveys & Tutorials,* vol. 21, n.º 3, pp. 2568-2599, 2019.

[8] S. Thrun, W. Burgard e D. Fox, Probabilistic Robotics, The MIT Press, 2005.

[9] B. Yamauchi, "A frontier-based approach for autonomous exploration," em *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, Monterey, CA, USA, 1997.

[10] Bitcraze AB, "Bitcraze Home Page," [Online]. Available: https://www.bitcraze.io/. [Accessed 12 September 2020].

[11] Bitcraze AB, "cflib 0.1.11 docs," [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/0.1.11/. [Accessed 9 September 2020].

[12] B. Rocha, "Lab Guide 1: Crazyflie - Manual Navigation," Laboratory Guide, Instituto Superior Técnico, Lisbon, 2020.

[13] B. Rocha, "Lab Guide 2: Crazyflie – Autonomous Navigation and Mapping," Laboratory Guide, Instituto Superior Técnico, Lisbon, 2020.

[14] B. Rocha, "Lab Guide 3: Crazyflie – Drone Delivery," Laboratory Guide, Instituto Superior Técnico, Lisbon, 2020.

[15] Bitcraze AB, "Crazyflie firmware v.2020.06 docs," [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/. [Accessed 9 September 2020].

[16] B. Rocha, "Getting a Crazyflie 2.1 ready for Lab Classes (Admin Guide)," Instituto Superior Técnico, Lisbon, 2020.

[17] B. Rocha, "Semi-Autonomous Indoor Drones," Master's Project Report, Instituto Superior Técnico, Lisbon, 2019.

[18] J. A. Preiss, W. Honig, G. S. Sukhatme e N. Ayanian, "Crazyswarm: A Large Nano-Quadcopter Swarm," em *IEEE International Conference on Robotics and Automation*, 2017.