

MANCHESTER
1824

The University of Manchester

Control of the quadcopter Crazyflie



Author:

M. Fabrice POIRIER
CI2019 - ROB

Team :

Mr. Alexandru STANCU
Mr. Eduard CODRES
Mr. Mario MARTINES
GUERRERO

September 30, 2018

Résumé

Ce rapport présente le quadrirotor "Crazyflie 2.0" et l'avancée qu'il permet dans l'étude de ce genre de drone. Ce petit drone est un projet "open-source" développé par la compagnie Bitcraze qui s'appuie sur une communauté active pour progresser. De par sa nature, il est utilisé par de nombreux étudiants et chercheurs pour l'application de théories dans la localisation, la régulation et l'électronique. C'est un projet touchant plusieurs domaines de connaissance, ce qui permet d'en apprendre d'avantage sur toutes les connaissances utiles en robotique. Les programmes nécessaires pour interagir avec le drone sont implémentés dans un projet ROS, et fait appel à des programmes de localisation et de régulation embarqués. Toutefois, les explications nécessaires pour correctement commencer à travailler avec le quadrirotor ne sont pour la plupart ni complètes ni à jour. Dans l'objectif de faciliter l'apprentissage de la partie robotique du projet, il était nécessaire de démontrer le potentiel du drone par la réalisation d'un contrôleur.

Abstract

This report presents the quadcopter "Crazyflie 2.0" and its possibilities concerning the study of this type of drones. This small drone is an open-source project developed by the company Bitcraze supported by an active community. As such, many students and researchers use it to apply theories on fields such as localisation, regulation or electronics. It is in itself a project one can learn different aspects of the robotics. The programs necessary to interact with the quadcopter are implemented in a ROS project, and they exploit the on-board localizer and regulator. However, most of the explanations to start working with the quadcopter aren't complete nor up-to-date. In order to ease the learning process concerning the robotic approach of this drone, a controller attesting the potential of the drone was mandatory.

Contents

1	Introduction	5
1.1	University of Manchester : Autonomous system research centre	5
1.1.1	Background	5
1.1.2	Economic analysis	5
1.1.3	Internship in the research centre	5
1.2	About the Crazyflie	6
1.3	Goals	6
1.4	Motivation	7
2	State of the art	8
2.1	Hardware	8
2.2	Firmware	10
2.3	Softwares	10
3	Stabilizer presentation	12
3.1	Localisation	13
3.2	Command	16
3.3	Regulation	16
4	ROS project	21
4.1	Camera localisation	21
4.2	<i>Crazyflie_ros</i>	23
4.2.1	<i>crazyflie_driver</i>	24
4.2.2	<i>crazyflie_controller</i>	25
4.2.3	<i>crazyflie_simulation</i>	26
5	Experimentation	27
5.1	methodology	27
5.1.1	observation of the drone behaviour	27
5.1.2	position control with off-board position estimation . .	29
5.1.3	position control with on-board position estimation . .	30
5.2	results	31
5.3	discussion	31

6	Conclusion and Future	33
6.1	Future possibilities	33
6.2	Conclusion	34
6.3	Outcomes on my project	35
A	Archived work	38
B	Flight videos	39
C	Evaluation report	40

Chapter 1

Introduction

1.1 University of Manchester : Autonomous system research centre

1.1.1 Background

I did my internship in the Control Systems Group (CSG), School of Electrical and Electronic Engineering (SEEE) at the University of Manchester. My internship supervisor was Dr Alexandru Stancu. He is a Lecturer in Control Engineering in the CSG and is also researcher at Dalton Cumbrian Facility (DCF), research centre for radwaste and decommissioning. His expertise is in nonlinear control, on-line and off-line identification, neural networks, fault monitoring and fault tolerant control — and its applications to ground mobile robots and mobile manipulators — and to biochemical processes.

1.1.2 Economic analysis

Unlike French school system, the university of Manchester doesn't allocate money to the researchers, and the latter have to pay for the office rent. This means funds have to be found from other sources such as industry partnership, hence the variation of the budget size year-round.

The impact of the engineers is that they all need to often publish research papers to stay relevant, and they need to work in collaboration with companies most of the time.

1.1.3 Internship in the research centre

The students in robotic engineering in the university of Manchester are working on theoretical subjects as well as practical ones (eg. "embedded systems", "process control and automation", "intelligent control and robotics" and "applied control" units). For this reason, they need to use learning-friendly robots. They were eager to add a drone to their list of robots to work on,

and choose the Crazyflie. However, none of them has enough knowledge on it, and had no time to study it. This is why Mr. Stancu gave this internship subject.

1.2 About the Crazyflie

Bitcraze is a Swedish company created in 2009. It started with a small team developping a quadrotor outside their job. In 2010, they shared a video of their work that became popular and encourage them to create the company Bitcraze in order to finance the development and manufacturing of the quadrotor. Their first version of the Crazyflie has been used for 4 years.

The Crazyflie 2.0 commercialised afterwards is an improved version of the initial quadcopter. It is an open-source project giving a complete control over the hardware, the firmware running on the quadrotor as well as the client library running off-board.

The Crazyflie is a small quadrotor with plastic blades. Indeed, it has a diagonal of 92mm and weighs 27g. It could fly for 5 minutes. Its size is ideal to fly indoors, as it could be easily studied flying from up close. Bitcraze hosts all of its codes on Github.



Figure 1.1: crazyflie 1.0

The Crazyflie is not the only product developped by Bitcraze. The CrazyRadio is a USB-radio dongle used to connect to the Crazyflie. There have also various expansion decks to improve their drone.

1.3 Goals

The autonomous systems field of research is working on different kind of robots, like ground robots and drones. These robots have to follow some criteria:

- Measurable: It is necessary to have access to all types of data to be able to interpret them;
- Modifiable: Being able to modify at least some part of its behaviour is mandatory;

- Repairable: Modify the robot to test algorithms could easily lead to breakages, and spare parts greatly reduce the cost of these relatively expensive devices.

The BitCraze Crazyflie matches all these requirements, and it's for these reasons that it is used in many universities like the one in Manchester.

As this is still a new system in constant evolution, it was necessary to see the potential of this drone. For this reason, the subject of my internship was at first to use the softwares associated to the Crazyflie. The next step was to master the algorithms and programmes implemented in the open-source project and make an autonomous controller for this drone. The following objective was to prepare a guide for the ones that will work on it. The points to look into were the following:

- How the open-source project is structured?
- What technologies could be the most interesting in regards of autonomous systems research?
- What could be a good starting point when starting to work on this device?
- What could be the tracks to follow afterwards?

1.4 Motivation

Quadcopters are nowadays a important part of the robotic field of research. they necessitate knowledge in physics, electronics, aviation, and computer science. They could carry objects in areas difficult to reach, with better manageability than most of autonomous systems. As it is attractive for commercial applications such as surveying and delivery, it will be a necessity to know how to handle these robots in the coming years.

Chapter 2

State of the art

To fully understand the potential of the Crazyflie, it is necessary to have an overview of the hardware¹, the firmware², and the softwares³⁴⁵ associated to this project.

2.1 Hardware

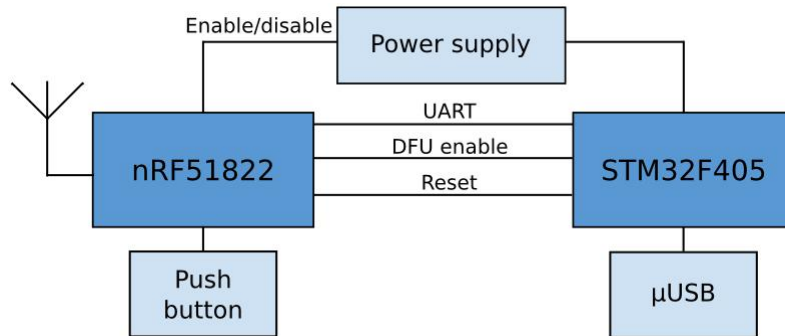


Figure 2.1: Global hardware architecture

The Crazyflie is divided into two main chips, the radio (nRF51822) and the main processing chip (STM32F405). The nRF51822 handles all of the raw radio communication. The received packets are then sent to the Crazyflie firmware in the STM32F405.

The Crazyflie Nano Quadcopter is equipped with an IMU (MPU-9250). The IMU contains a 3-axis gyroscope and a 3-axis accelerometer. The

¹<https://wiki.bitcraze.io/projects:crazyflie2:hardware:specification>

²<https://github.com/bitcraze/crazyflie-firmware>

³<https://wiki.bitcraze.io/projects:virtualmachine:index>

⁴<https://github.com/bitcraze/crazyflie-clients-python>

⁵https://github.com/whoenig/crazyflie_ros

Crazyflie is also equipped with an on-board magnetometer (AK8963) as well as a barometer (LPS25H).

The CrazyRadio is a USB-radio dongle. When connected to a computer, It permit to interact with the Crazyflie. It is important to note that the radio protocol limits the size of the packets being sent from the Crazyflie to the Crazyradio. This prevents from sending the entirety of the sensor measurements with each packet without reducing the frequency at which the state estimator gets sensor updates.



Figure 2.2: Crazyflie battery

The battery used by the Crazyflie 2.0 is a Lithium-Polymer (LiPo) battery. It supplies 3.7V and has a capacity of 240mAh. The battery also comes with a Protection Circuit Module (PCM) attached to it that prevents the user from under or over charging the battery or from shorting it. The PCM is located under the orange tape on the top side of the battery where the wires come out.

The battery is easily removable from the quadrotor, which means it could be swapped with another one and charged in parallel, reducing delays between experiments. The battery's discharge rate is 15C, which in theory should provide 4 minutes of continuous flight.

The Crazyflie 2.0 has four brushed DC motors. The motors are coreless which should provides a faster acceleration. However, the Crazyflie 2.0 motors require more torque at higher speeds in order to fight the increased air resistance and therefore more current. That means the battery voltage tends to drop. Attached to the four motors are plastic propellers, which mitigate the possible damage in case of accident, but are therefore more fragile.

The quadcopter can also support extension decks, which permits to add more possibilities to the drone. For example, the loco deck⁶ grants better positioning system.



Figure 2.3: Loco positioning deck

⁶<https://www.bitcraze.io/loco-pos-system/>

2.2 Firmware

The Crazyflie firmware is based on FreeRTOS, an open source operating system. it handles the scheduling of processes and control the flight calculations.

The firmware is in C and is composed of several source code and headers, structuring the code in different layers of abstraction. The first ones (*drivers* and *hal* folders) handle the hardware, whereas all the operations are done in the high-level functions (*module* folder). This last one is the one studied in detail in another chapter. There are also layers covering non-classical use of the quadrotors, such as the interfacing of extension decks.

The radio communication use a protocole developped by the company, the *Crazy RealTime Protocol* (CRTP). The compilation necessitates to have a wired connected with a computer, with the quadrotor in the bootloader mode. The firmware can be customised at compile time.

2.3 Softwares

The company shared a Xubuntu virtual machine hosting all the firmware and a software (a client) to interact with a Crazyflie using the Crazyradio dongle. Once connected, the client has access to the drone logs, and could plot them easily. It could also change all parameters defined in the firmware for as long as the drone isn't rebooted. The user-friendly interface of this software helps to promptly start working on the project. Android and iOS applications have also been developed to use smartphones as controllers.

In the research field, the ROS project based on the work of W.Hoenig [1] is more suitable. It will be detailed more precisely in another chapter.

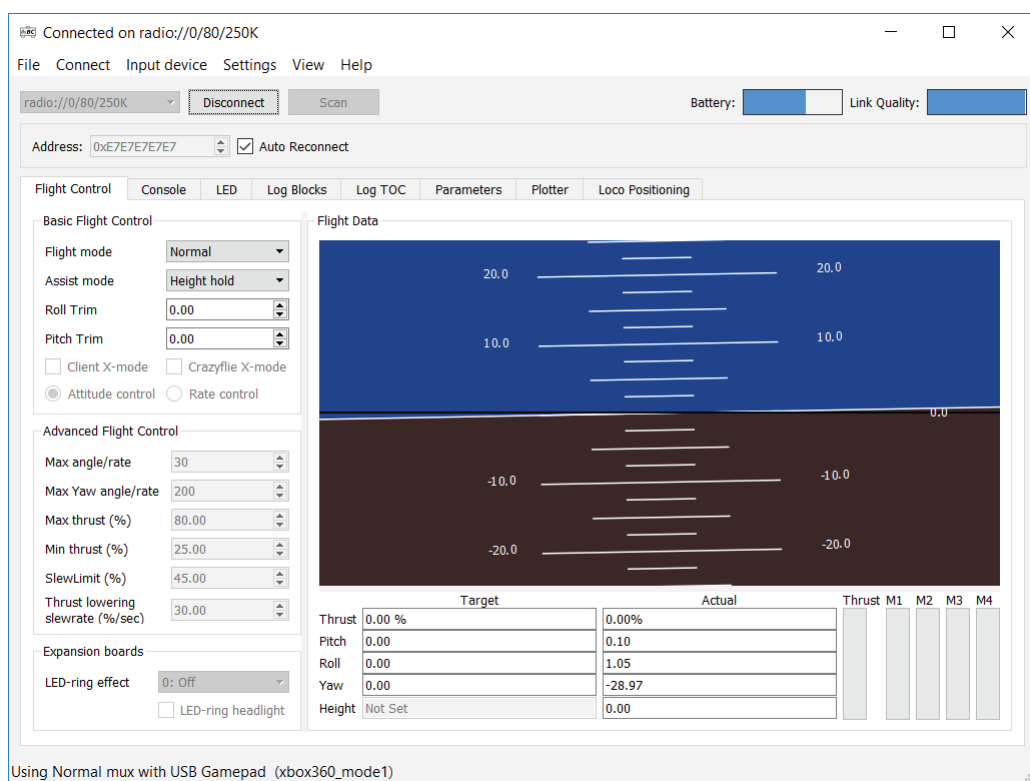


Figure 2.4: The interface of the Bitcraze client

Chapter 3

Stabilizer presentation

The small size of the Crazyflie presents many challenges in using it as a research platform. Its small inertia requires controllers that can react with very little latency, or else it won't fly properly. To be the most efficient possible, all the calculations necessary to stabilize the drone is done by the Crazyflie firmware.

The stabilizer is in three parts, each of them depends on the type of stabilization wanted, here it will be in position. The first one is the localisation, that gives the absolute position. Then comes the command part, which gives the positions to go. Finally, the regulator moves the drone to the position wanted.

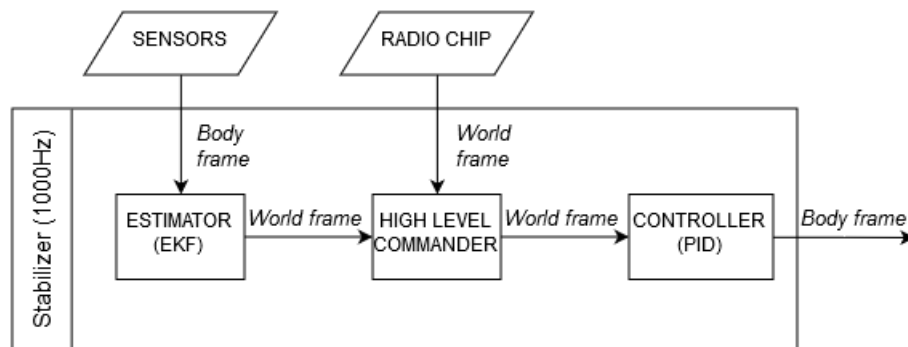


Figure 3.1: Global architecture of the stabilizer in the position mode. The coordinate frame is in *italic*

The *stabiliser.c*¹ file is the main file for the stabilisation loop.

¹<https://github.com/bitcraze/crazyflie-firmware/blob/master/src/modules/>

3.1 Localisation

In most autonomous missions, it is necessary to track the position of a robot. It could be done in the body frame or in the world frame. As the first one could be done easily, the second one is preferred here. Indeed, drones couldn't ignore gravity like most of ground robots, which is a force working in the world frame. Moreover, the command part necessitates to give a new position to go, and it isn't intuitive to work in the body frame.

There are different methods for localisation. Nevertheless, for a drone with the type of sensors presented in the previous chapter, and with the speed of calculation of the processor, some are better than others. Indeed, in systems where we need to obtain continuous or dynamic measurements from sensors, the sensors' measurements are probably uncertain due to reasons which include, errors from sensor and discrepant measurement from multiple sensors. The Crazyflie source code contains both a complementary filter and an Extended Kalman Filter [7] [8] (EKF) to do an estimation of the drone attitude. However only the latter can estimate both orientation and position, as the first only estimate the orientation. Only the EKF will be presented here.

Kalman filter is an algorithm that obtains reliable estimations on a system state. It works if with have a sequence of observed measurements, possibly from multiple sensors, and the dynamic model of the system (under the from $x_{k+1} = f(x_k, u_k)$, x the state, u the command, k a discrete value of the time). The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft.

For a robot where the position is estimated, the filter is organized under the following steps that are repeat at each loop:

- The prediction: At a certain time, the current estimated position is known, as well as the dynamic model. From that, the next position is predicted, which becomes the new estimated position.
- The correction (or update): The estimated position need to be corrected to correspond more to the real position, The measures are then interpreted to update the estimated state.

As the estimation accumulates errors from different sources, the cumulated error is defined by a covariance matrix that is changed at each steps of the algorithm.

For the original Kalman filter, the hypotheses are that the system has to be linear and the noises have to be gaussian. But for non-linear ones such as the Crazyflie, it doesn't work properly. The EKF is a variant of Kalman filtering that can more easily deals with this type of system. It does a first

order Taylor expansion² to obtain a linear approximation of the non-linear measurements and dynamic models.

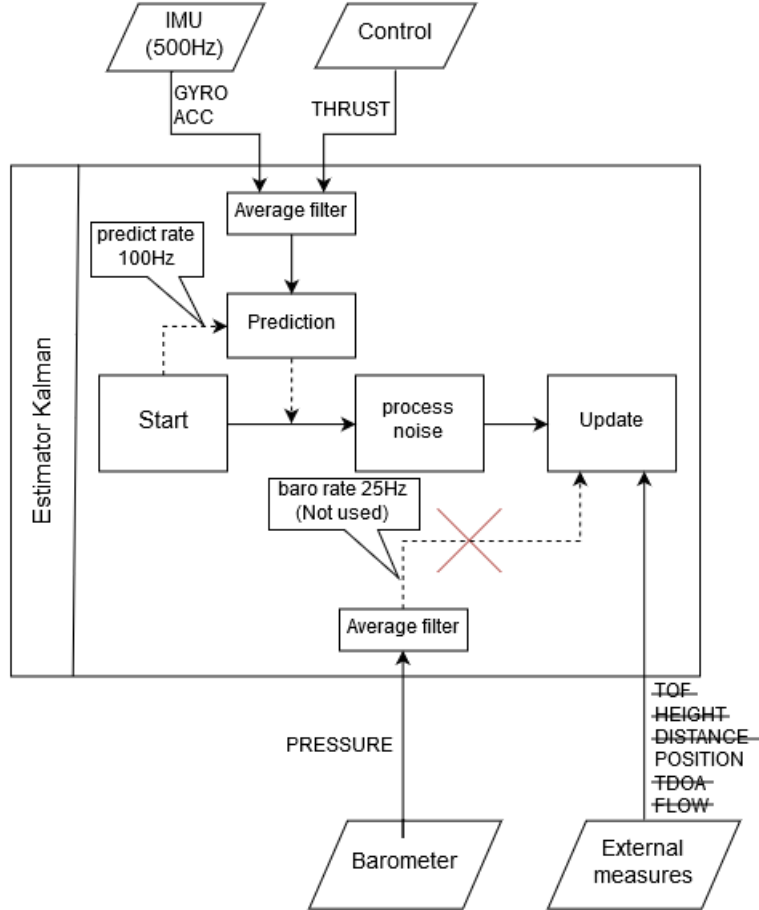


Figure 3.2: Architecture of the EKF

In the Crazyflie, the EKF, represented in the figure above, works as such:

- The EKF works at the stabilizer rate, which is 1 kHz;
- The filter necessitates sensor measures for its prediction step, and more precisely for its quadrotor dynamic model. However, the data are noisy and arrive at an irregular rate, which is why the data are averaged. The prediction couldn't be done at the data received rate, hence a predict rate at 100Hz, which is 5 times lower than the IMU rate;

²For illustration purpose, the Taylor extension of a dynamic model is under the form : $x_{k+1} = f(x_k, u_k) = f(\hat{x}_k, u_k) + \frac{df}{d\hat{x}}(\hat{x}_k, u_k) \times (x_k - \hat{x}_k)$ with \hat{x}_k being in our case the previous estimated state.

- Even if the prediction isn't done at each loop, the algorithm adds process noise to simulate all the possible errors the system could have;
- The estimated position is then update with various measures from external sources (e.g. from an on-ground position estimation). This process refines the position and the error associated. The one by default is the position measure.

The barometer has not been interfaced here as the results were thought unsatisfying by the community.

The complementary filter is compiled by default. The EKF can be chosen by adding "ESTIMATOR=kalman" to the file tools/make/config.mk in the Crazyflie firmware before compiling. The two estimators are in the *estimator_complementary.c* and *estimator_kalman.c* files.

▼ kalman			
initialX	float	RW	0.5
initialY	float	RW	0.5
initialZ	float	RW	0.0
mNBaro	float	RW	2.0
mNGyro_rollpitch	float	RW	0.10000000149011612
mNGyro_yaw	float	RW	0.10000000149011612
pNAcc_xy	float	RW	0.5
pNAcc_z	float	RW	1.0
pNAtt	float	RW	0.0
pNPos	float	RW	0.0
pNSkew	float	RW	9.999999747378752e-06
pNVel	float	RW	0.0
quadIsFlying	uint8_t	RW	0
resetEstimation	uint8_t	RW	0

Figure 3.3: parameters of the EKF

The EKF parameters shown in the figure above are the following:

- initial[...]: The Kalman filter need an initial position to start the algorithm;
- quadIsFlying: Define if the drone is already flying when the algorithm starts could be sometimes useful;
- resetEstimation: It is necessary to reset the Kalman filter to prevent error to cumulate before starting a mission. This parameter can be set from the ground at any moment. However, it has to be done when idling, or the drone will crash otherwise.
- mN[...] and pN[...]: The parameters starting with "mN" are the measure noises (linked to the quality of the measurements), and the ones starting with "pN" are the process noises (linked to the possible positions the drone could move to during a loop).

3.2 Command

The command part isn't the most complicated part in theory, as it is just to give a position to go, here with an absolute positioning. However, if a position given is too far from the current position, the regulation could move the drone with extreme command, and could eventually lead to flips, huge oscillations, or crashes. Most regulation algorithms choose to avoid this problem by creating intervals of values that cannot be exceeded. However, it seems better to create intermediary setpoints to smoothen the trajectory. A high-level commander is implemented in the firmware to do so. Unfortunately, at the time of the experimentation, the algorithm wasn't operational.

The operator has to choose positions that will not create perilous behaviours, even if regulation algorithms handle these situations. Command setpoints (X,Y,Z,yaw rate) are sent to the Crazyflie by radio. The setpoints are then directly redirected to the regulation algorithm.

3.3 Regulation

Once the current and the desired positions are known, the drone has to move appropriately. Indeed, moving too fast could lead to dangerous behaviours (eg. flips and crashes). On the opposite, a slow movement could lead to drifts and a low reactivity. A bad regulation will never stabilize the drone on the desired position.

The Crazyflie hosts two different regulation algorithms. The first one, and the most used when it boils down to regulation, is the Proportional-Integral-Derivative controller (PID). A PID controller calculates an error value from the difference between a desired setpoint and a measured variable at each control loop. It then applies a correction based on the sum of a proportional (P), an integral (I), and a derivative (D) terms. Each term is adjustable with coefficients as each of the latter represents a different aspect of regulation.

The proportional term is the current error value. The further the desired value is, the bigger this term will be.

The integral term is the sum of error values over time. It is proportional to both the magnitude of the error and the duration of the error.

The derivative term is the slope of the error over time. The stronger the slope is, the bigger this term will be.

The coefficients are responsible for the stability, the accuracy and the speed of convergence of the system. There are different methods of fine-tuning the coefficients. However, this is rarely the best regulation algorithm possible, and gives poor results on some systems. The PID regulator of the Crazyflie is in fact composed of 4 PID controllers, each of them controls a different aspect of the drone, and helps to obtain the desired motor control

parameters — Roll-Pitch-Yaw (RPY) rates and thrust — from the desired position.

In the Crazyflie, the PID control algorithm, represented in the figure below, work as such:

- The whole regulation works at lower rate than the stabilizer as it would use unnecessary computational time otherwise. The rate is then of 500 Hz.
- The command algorithm, when set to receive positions, transmit XYZ and a yaw rate to the regulation algorithm. In velocity mode, the yaw rate is primarily integrated to change the desired yaw;
- As the desired orientation estimation takes a significant time to compute, its frequency was lower than the whole regulation. the estimation rate is of 100 Hz, and work as follow:
 - The desired drone velocity is calculated from a PID on its position in the World frame;
 - The thrust-roll-pitch are calculated from a PID on velocity.
- Then, the last desired roll-pitch-yaw estimated are compared to the drone RPY with another PID, which gives the desired RPY rate. For the position regulation, the desired yaw rate is directly given by the command;
- A final PID updates the actuator force with the RPY rates.

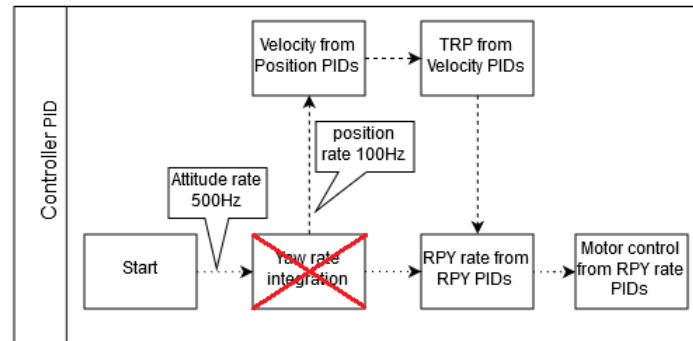


Figure 3.4: Architecture of the PID in the position mode

The other regulation algorithm is the Mellinger one [6].

In constrained settings typical of indoor environments, it is necessary to constrain the behaviour of a quadrotor, but that doesn't mean the trajectory couldn't be optimize. This is what the Mellinger algorithm does. It is in two parts:

- a nonlinear control algorithm that could follow aggressive trajectories requiring large accelerations;
- a trajectory generator that enables the generation of optimal trajectories through a series of positions and yaw angles, while ensuring safe passage through specified corridors and satisfying constraints on achievable velocities, accelerations and inputs.

The trajectories are optimal as they minimize the cost functionals that are derived from the square of the norm of the snap (the fourth derivative of position). The corridors are defined by their widths. This parameter changes the cost functional (eg. a large one reduces position constraints, and the drone may take a less direct and lower cost path). Aside that, the trajectories can be done at a slower pace for more accuracy, and therefore be safer.

The control algorithm is aggressive in the sense that it allows significant excursions of the attitude from the hover state. It is similar to another dynamically exponentially stable controller³, but does not quite satisfy all the assumptions needed to be as stable. Nevertheless, the controller yields good tracking performance even with very large roll and pitch angles, according to experimental results in fast motion three-dimensional slalom courses done by D. Mellinger.

For its application in the Crazyflie, only the control algorithm has been implemented.

In the Crazyflie, the Mellinger control algorithm work as such:

- The vector force (in the body frame) necessary to reach the desired attitude is calculated with a PID using the differential between the drone position, linear speeds and accelerations, and the desired ones. The command thrust is compute here.
- The vectoral moment is calculated with a PID using the differential between the drone orientation and angular speeds, and the desired ones.
- Finally, the control orientations (RPY) are calculated from the moment. This step is achieved in the previous step, as it directly uses these results.

It is important to note the integral terms aren't in the original theory. It has been implemented to compensates practical issues (battery voltage drops over time, unbalanced centre of mass due to asymmetries, uneven wear on propellers and motors).

³T. Lee, M. Leok, and N. McClamroch, "Geometric tracking control of a quadrotor uav on SE(3)," in Proc. of the IEEE Conf. on Decision and Control, 2010

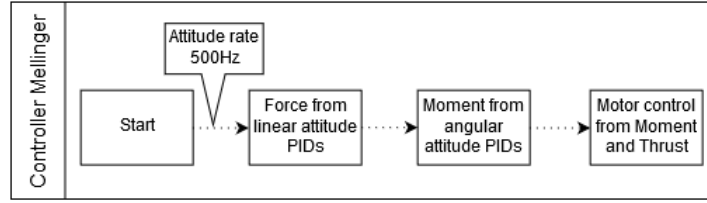


Figure 3.5: Architecture of the Mellinger regulation in the position mode

For indoor (manoeuvres), without any external perturbations, the Mellinger regulation is supposedly better. However, this regulation algorithm isn't fit for outdoor missions, and the PID one prevails there.

The PID regulation was chosen over the Mellinger regulation for the experimentation for two reason:

- First, the mission was to hover at some positions without obstacles. Therefore, the speed and trajectory to reach the position is neglected.
- The other reason is that the PID has a straight forward approach of regulation, and could be easily understood for that.

The two regulators are in the *controller_mellinger.c* and *controller_pid.c* files.

PID regulation necessitate good coefficients to ensure stability, accuracy and speed to fit a certain situation. For example, indoor environments are suitable for extreme reactivity to avoid obstacle, or else the drone would drift too much. However, these parameters couldn't be kept for the outside, the wind couldn't exaggerate the regulation movement, and could overturn and eventually crash. Each PID has its own set of parameters that could be modify if needed as shown in the figure below. They are related to 3 coordinate systems, that could be the XYZ or RPY system. The final PID controls the motor, and therefore it takes also physical limitations as well.

▼ velCtlPid				▼ pid_attitude			
vxKd	float	RW	0.0	pitch_kd	float	RW	0.0
vxKi	float	RW	1.0	pitch_ki	float	RW	3.0
vxKp	float	RW	25.0	pitch_kp	float	RW	6.0
vyKd	float	RW	0.0	roll_kd	float	RW	0.0
vyKi	float	RW	1.0	roll_ki	float	RW	3.0
vyKp	float	RW	25.0	roll_kp	float	RW	6.0
vzKd	float	RW	0.0	yaw_kd	float	RW	0.3499999940395355
vzKi	float	RW	15.0	yaw_ki	float	RW	1.0
vzKp	float	RW	25.0	yaw_kp	float	RW	6.0

Figure 3.6: Velocity from position PID

Figure 3.7: Thrust-Pitch-Yaw from velocity PID

▼ pid_rate			
pitch_kd	float	RW	2.5
pitch_ki	float	RW	500.0
pitch_kp	float	RW	250.0
roll_kd	float	RW	2.5
roll_ki	float	RW	500.0
roll_kp	float	RW	250.0
yaw_kd	float	RW	0.0
yaw_ki	float	RW	16.700000
yaw_kp	float	RW	120.0

Figure 3.8: Roll-Pitch-Yaw rate from RPY PID

▼ posCtlPid			
rpLimit	float	RW	20.0
thrustBase	uint16_t	RW	36000
thrustMin	uint16_t	RW	20000
xKd	float	RW	0.0
xKi	float	RW	0.0
xKp	float	RW	2.0
xyVelMax	float	RW	1.0
yKd	float	RW	0.0
yKi	float	RW	0.0
yKp	float	RW	2.0
zKd	float	RW	0.0
zKi	float	RW	0.5
zKp	float	RW	2.0
zVelMax	float	RW	1.0

Figure 3.9: Motor command from RPY rate PID

As for the Mellinger algorithm, the coefficients of the two PID can be modify if felt necessary. In case the mass of the Crazyflie is changed, the change could be pass on here.

▼ ctrlMel			
i_range_m_xy	float	RW	1.0
i_range_m_z	float	RW	1500.0
i_range_xy	float	RW	2.0
i_range_z	float	RW	0.4000000059604645
kR_xy	float	RW	70000.0
kR_z	float	RW	60000.0
kd_omega_rp	float	RW	200.0
kd_xy	float	RW	0.20000000298023224
kd_z	float	RW	0.4000000059604645
ki_m_xy	float	RW	0.0
ki_m_z	float	RW	500.0
ki_xy	float	RW	0.05000000074505806
ki_z	float	RW	0.05000000074505806
kp_xy	float	RW	0.4000000059604645
kp_z	float	RW	1.25
kw_xy	float	RW	20000.0
kw_z	float	RW	12000.0
mass	float	RW	0.03200000151991844
massThrust	float	RW	132000.0

Figure 3.10: Parameters of the Mellinger regulation

Chapter 4

ROS project

Robotic is a predominant subject in our society. They are asked to be more reliable, more complex, safer, and have to work along us and between themselves as packs. It necessitates multi-disciplinary knowledges, which means normalization is needed to facilitate the clarity of their structures. This is done by middleware such as the Robot Operating System¹ (ROS), and it is essential to learn to work with them.

A middleware is a software that permit the communication between two tasks (eg. detect with a sensor, and control an actuator). It helps to simplify the implementation of algorithm by normalising the transmission of data, simplify teamworking, as well as giving access to more advanced tools (eg. 3D visualisation). For the ones not familiar with ROS, there is a complete tutorial² that helps to master this middleware.

Here are the Ros projects needed for the experimentations described in the next chapter:

- *usb_cam_node*: creates a ROS topic for a camera device;
- *Whycon* [4] [3] [10] [9]: a tool for localisation of certain patterns;
- *crazyflie_ros*: different projects to interact with one or more Crazyflies.

The ROS workspace can be found in the Appendix A.

4.1 Camera localisation

As explained previously, The EKF needs external positions measurements to gain in precision. A relatively cheap solution is to use a camera. A simple monocular camera only render the video flow, therefore a software is necessary to interpret it. As ROS is the middleware to work on, the video flow interpretation will be done by nodes on it.

¹<http://www.ros.org/about-ros/>

²<http://wiki.ros.org/ROS/Tutorials>

On the one hand, the video flow has to correspond to a ROS node. *usb_cam_node* is one of the simplest ones to do so. It creates a node that gather the video flow and the camera header, and publishes them on two different topics. It takes some parameters to work properly, the most important ones are:

- *video_device*: The device the camera is on;
- *image_width*: width in pixel;
- *image_height*: height in pixel;
- *pixel_format*: The possible formats are mjpeg, yuyv and uyvy;

On the other hand, we need to detect the position of the Crazyflie on the video. Recognize the shape and the colours of an object with a camera is one thing, and detect the positioning of the detected object is even more complex. Therefore, there are various methods to do so. *Whycon* is the one chosen for its efficiency, accuracy and utilisation friendliness. The system is being used in several research projects across the globe ³.

It is composed of an efficient circular black ring pattern detector and a detector of multiple targets on a single image. It then computes the 3D position of each circle. It creates a node that subscribe to a video flow topic, and publishes the 3D positions of the detected circles.

As the detection depends on the experimentation frame, it is important to understand its parameters:

- *axis*: A file containing the user-defined transformation from camera-space to world-space could be used;
- *targets*: It indicates how many targets are to be tracked;
- *max_refine*: maximum number of refinement steps to be performed after successfully detecting a given circle (1 = no refine). For a given image, the detection process for a given circle can be improved iteratively until convergence by increasing this number;
- *max_attempts*: maximum number of attempts to detect a given circle while processing a single frame;
- *outer_diameter*: It is the diameter (in meters) of outer portion (black) of the circle(s);
- *inner_diameter*: It is the diameter (in meters) of inner portion (white) of the circle(s).

The balance between precision/robustness and computational time to process each frame could be change with the *max_attempts* and the *max_refine* parameters.

³See <https://github.com/lrse/whycon>

4.2 *Crazyflie_ros*

Crazyflie_ros is the ROS project primarily developed by a Bitcraze community member. Now, it is the fusion of different projects under the guidance of the Bitcraze team. It is composed of 7 sub-modules on its repository, but the one in the Appendix A has one more sub-module, which is a step to accomplish the objective presented by this report.

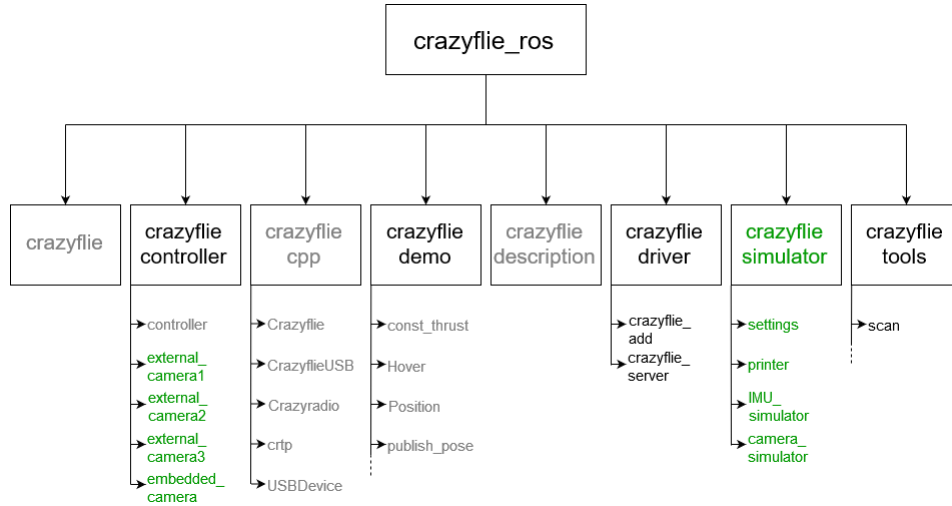


Figure 4.1: *Crazyflie_ros* project. Gray: not directly used in the experimentation; Green: created

Some of these projects aren't useful enough to be fully presented in this report:

- The *crazyflie* sub-module is the metapackage necessary for the whole *crazyflie_ros* project.
- The *crazyflie_cpp* sub-module define classes of different objects (e.g. Crazyradio, Crazyflie). It adds a layer of abstraction that ease the use of these devices.
- The *crazyflie_demo* sub-module regroups bricks of algorithms that accomplish simple tasks, such as hovering, holding a position and execute trajectories. However, they are experimental works and most of them don't work anymore, or with certain untold settings. Nevertheless, it helps to roughly apprehend the functioning of the ROS project.
- The *crazyflie_description* sub-module contains an URDF model of the Crazyflie, which is a 3D model for the *rviz* software.

- The *crazyflie_tools* sub-module is composed of useful tools, but aren't up-to-date. The most helpful one is the scan tools, that check if the Crazyradio and the Crazyflie are detected. it takes the Crazyflie address defined in the drone firmware⁴ as a parameter.

4.2.1 *crazyflie_driver*

This sub-module contains a server (*crazyflie_server*) that could communicate with multiple Crazyflies with a single Crazyradio. The other sub-module is the *crazyflie_add* file which is a program that dynamically add Crazyflies to the server. The server does not communicate to any Crazyflie initially, hence *crazyflie_add* needs to be used. These two programs work as drivers for a remote control.

The server creates the node which subscribes to various types of command to send to the Crazyflie, and publishes what it receives from the drone. There are different methods to command the quadrotor. Once chosen one, you can swap to another. However, some of them are flawed. For example, the *cmd_hover* isn't always stable, and even though the localisation algorithm is in the Crazyflie firmware, using the *cmd_position* topic means having no information on the estimated position from the off-board.

⁴The method to change it is described in the Appendix A

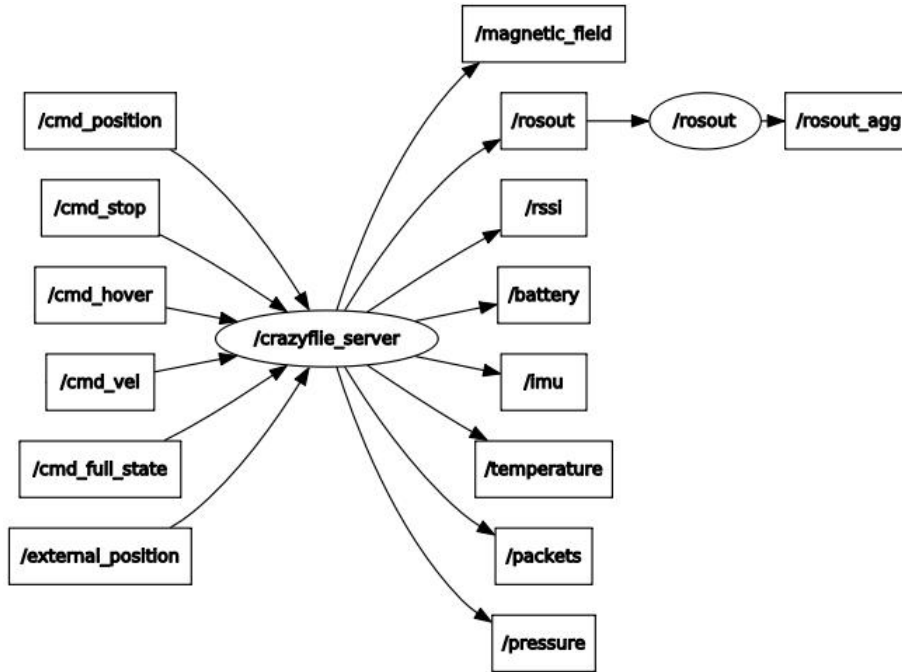


Figure 4.2: Node graph of the subscribed and published topics of the Crazyflie server (with RQT)

The process to add drones to the servers takes some parameters⁵ to successfully link the node to the quadrotor:

- The *uri*: It is the address of the drone;
- The *tf*⁶ *prefix*: Gives the *tf prefix* for the crazyflie frame(s);
- *Roll trim*: Set the roll trim in degrees;
- *Pitch trim*: Set the pitch trim in degrees.;
- *enable logging parameters*: Permit to choose which data are necessary to be send by radio from the Crazyflie. Only the *rssi*, which measures the strength of the signal isn't deactivable, as it confirms the drone can still communicating with Crazyradio.

4.2.2 *crazyflie_controller*

This package contains on its repository a simple PID controller for hovering or waypoint navigation. It can be used with external motion capture systems,

⁵https://github.com/whoenig/crazyflie_ros/blob/master/crazyflie_driver/launch/crazyflie_add.launch

⁶tf is a ROS package that lets the user keep track of multiple coordinate frames over time. It is necessary for *rviz* visualisation

such as *VICON*⁷. The programs necessary to fulfil the objective have been added here. There are different controllers with different method of control. Three of them use an external camera and a target on top of the Crazyflie facing the front side, another one uses an with embedded camera with the same target facing the drone from a distance. Their functioning will be thoroughly described in the next chapter, except for the Embedded one. Unfortunately, the latter has not been tested, as the camera needed to be calibrated and the usual way to do so wasn't effective.

4.2.3 *crazyflie_simulation*

This sub-module has been created for this work, and could only be found in the Appendix A. It can simulate a Crazyflie using the IMU only, or in addition with an external camera.

The *camera_simulation* and the *IMU_simulation* scripts create nodes that subscribe to some sensor topics, estimate the drone attitude from it, and publishes the estimated position and orientation. The *printer* script creates a node that could receive the position published by one of the simulator, and display a simple 3D representation of the drone according to the position. This node uses a schematic representation of a quadcopter. All units are in the SI base unit.

⁷<https://www.vicon.com/motion-capture/engineering>

Chapter 5

Experimentation

5.1 methodology

The objective was to hover the drone autonomously at any given position. As the ongoing regulations forbid the flight of drones in open areas, it was necessary to restrain the accessible positions based on the initial position. Hence reaching a position of half a meter above the ground will be considered as a success. The strategy to follow will be to localize and regulate the position of the drone thanks to its internal sensors and an external camera.

The practical steps for this methodology are in the Appendix A.

5.1.1 observation of the drone behaviour

Flying robots are more difficult to control than most of ground and aquatic robots, and errors could lead to crashes and eventually casualties. Fortunately, this is a small drone with plastic propellers which mitigate the risks. It is then important to consider every step before starting the flight tests. For that, the real-time simulator nodes were implemented.

The first simulator used the pressure and the IMU data sent by the Crazyflie for its position estimation. It is based on a second order Runge-Kutta method. However, as it didn't take into account that all the IMU measurements were noisy and bias, this simulation has been put aside.

The second simulator took into account the position estimation from the *Whycon* node as well as the IMU measures. Indeed, the orientation measurements from the *Whycon* seemed less accurate than the one from the IMU measurements. This means the only computation here were a simple integration of the angular speed from the IMU to give the drone orientations.

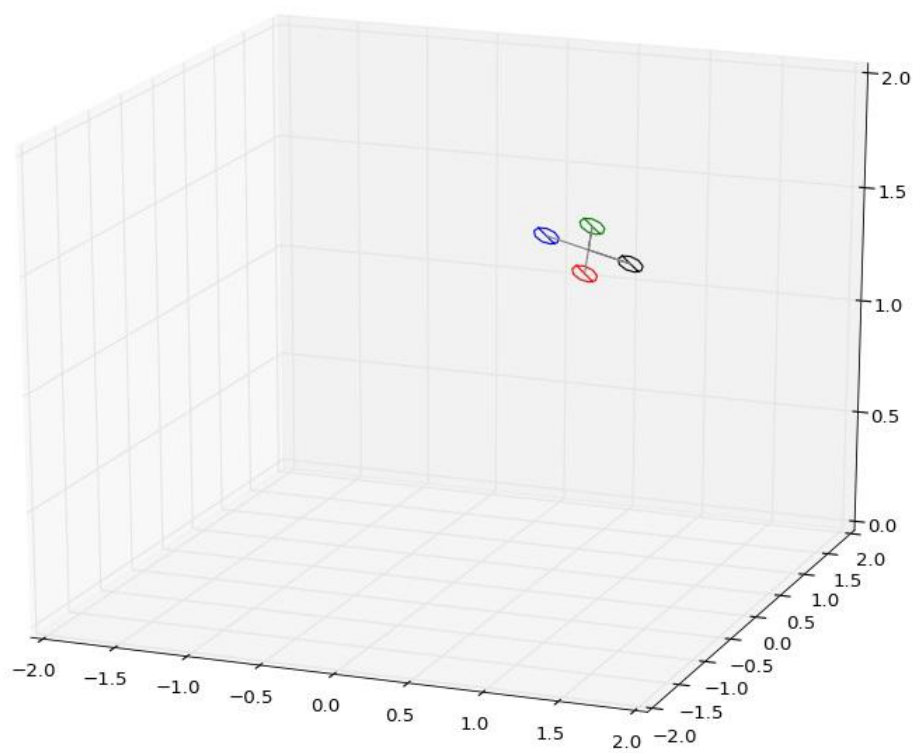


Figure 5.1: Overview of the simulation (with *Matplotlib*)

5.1.2 position control with off-board position estimation

The first attempt was to detect the drone without using an already existing method. It could have been a way to eventually detect the Crazyflie with a less restrictive and more customized method than with a target on top of the drone. It used the OpenCV tools to do so. The results were that even recognize the marker pattern on a motionless quadrotor was an unnecessary difficulty. The bulkiness of the detection system was then neglected, as long as it didn't obstruct the drone movement, and provide a reliable method of localisation.

The *Whycon* method was the solution chosen to do so. The node controlling the drone subscribed to the IMU and the *Whycon* topics. A simple PID have been implemented for the regulation. Finally, the node published on the *cmd_vel* topic, which takes speeds to give to the Crazyflie in the world frame XYZ axes. The node graph is represented on the next figure.

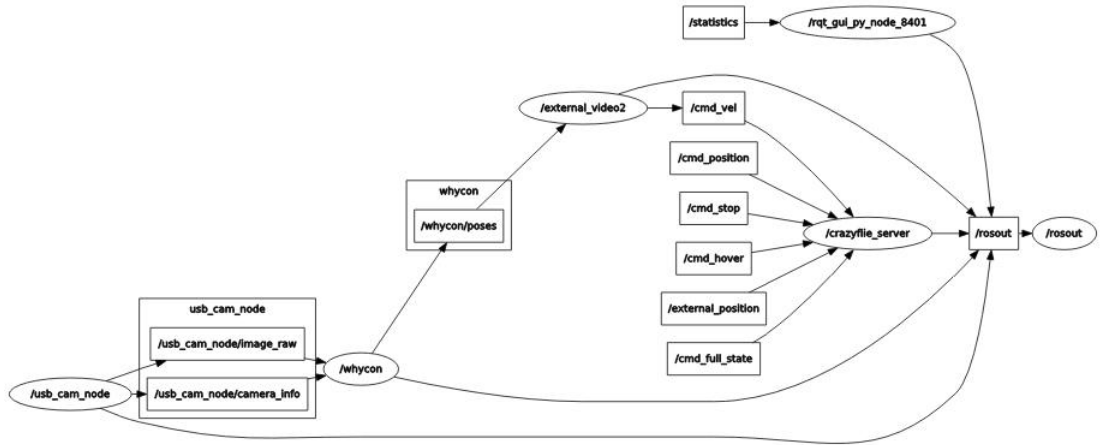


Figure 5.2: Node graph of the *external_video2* setup (with *RQT*)

The results publish by the *Whycon* node were good for the position estimation, but were less accurate than the drone IMU measures for the orientation. The solution was to use both of them for their respective best measurements. The new node graph is in the figure below.

The results were the position estimation provided acceptable values, but the PID regulation led to drifts and instabilities. It could have been resolved by finetuning the PID coefficients, but that would not show the strength of the stabilizer implemented in the firmware. For this reason, it was necessary to go to the final step.

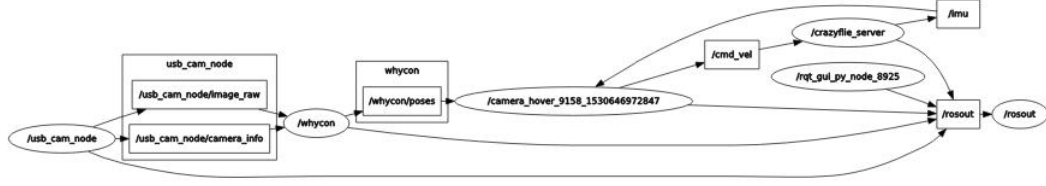


Figure 5.3: Node graph of the final *external_video2* setup (with *RQT*), named here *camera_hover_...*

5.1.3 position control with on-board position estimation

The stabilizer is composed of a position estimator and a regulation algorithm¹. Both of them are more complex and optimized than the controllers done in the previous steps.

It creates a node that subscribed once again to the IMU and *Whycon* topics. It then publish the estimated drone attitude on the *external_position* topic, that refined the Kalman filter estimation. Then the node published the desired position on the *cmd_position* topic. The following figure is the node graph obtained.

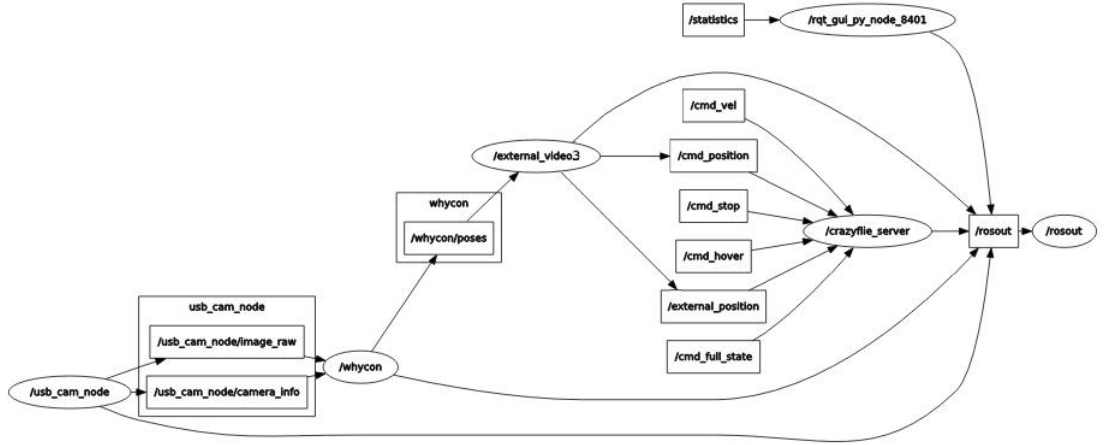


Figure 5.4: Node graph of the *external_video3* setup (with *RQT*)

This method should permit a better stabilization, as the computation are mostly done on-board, and is therefore faster. The drone could know regulate itself properly for short duration (a few seconds) even without new update from the ground, which allow to reduce the radio transmission rate previously necessary for stabilisation.

¹See the corresponding chapter for more details

5.2 results

The two videos in appendix B show the drone attitude for two different desired positions. One can see the drone reaches the desired position in both cases, then return approximatively to the initial position. Other attempts have been done, leading a few time to crashes, and sometimes to movements around the desired point.

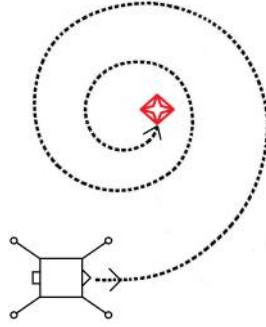


Figure 5.5: Trajectory with oscillations in the front-rear and left-right axes

5.3 discussion

The approximate return position was due to the fact the initial position was situated on the ground, which created a discontinuity on the regulation (oscillations led to bumping on the ground). Moreover, this position was given just before the ending of the algorithm. The regulation couldn't work long enough to do its job.

From the data collected, all the crashes were due to the fact the position of the drone couldn't be detected any longer at a certain time of the autonomous flight. There were three reasons possible to that:

- The paper with the landmark on it bended. If this type of localisation (landmark on the drone) will be keep in the future, it will be necessary to use another material that doesn't deteriorate easily, without adding too much weight, and therefore changing the centre of inertia;
- The drone fly out of the camera frame. It could be that the position given were too far away, or the regulation moved the drone with an exaggerated amplitude. Choosing safer positions or looking into the regulation algorithm parameters could be solutions to this problem;
- The drone wasn't facing the camera anymore. The orientation is only estimated by the IMU, whose measurements are noisy and could easily lead to errors other time. Moreover, the regulation changes the roll

and the pitch to move the drone adequately. Both of these reasons could hide the marker. One could change the yaw rate dynamically to always have the head of the Crazyflie facing the camera.

As said earlier, the drone moved around the desired position before stabilizing. This is mostly due to the fact the PID regulation parameters are set to work outside. Indeed, because of the wind, the drone couldn't be as nervous as indoors. As these settings are safer, the drone is bound to react slowly.

Overall, it was a success, and the objectives were accomplished.

Chapter 6

Conclusion and Future

6.1 Future possibilities

The experimentation was thought as an overview of the Crazyflie potential. As seen in the previous chapter, it could be finetuned to obtain a more reliable result if felt necessary.

Another solution to improve it would be to change the localisation algorithm. Indeed, the EKF is currently the only position estimator working in the firmware. There are works on other localisation methods [2] [5] that could be interesting to develop.

The sensors currently used could be completed or replaced by others. The Loco positioning system¹ developed by the company, for example, uses beacons to improve the localisation. The target on the Crazyflie could be modified to allow changes of orientation as in the picture below took from the *Whycon* website.

Once flying the drone as wanted will no longer be an issue, it could be interesting to give the Crazyflie a mission to do. One could map a room with obstacles. A first step would be for an embedded camera to detect markers with unknown positions, and determine their positions in the room. It could alternatively help to improve the localisation.

If the Hardware is thought as insufficient, improving it could be a solution. The payload capacity is limited, which makes it challenging to add additional piece of technology. To help doing so, Bitcraze developed decks for different uses, working with one of these could be useful (the flow deck², the Loco positioning deck³, the Qi charger deck⁴, or the prototyping deck⁵).

Adding a camera on the drone was done, but one could choose another camera, or embed it more adequately (e.g. use the micro-USB port to supply

¹<https://www.bitcraze.io/loco-pos-system/>

²<https://www.bitcraze.io/flow-deck/>

³<https://www.bitcraze.io/loco-pos-system/>

⁴<https://www.bitcraze.io/qi-charger-deck/>

⁵<https://www.bitcraze.io/prototyping-deck/>

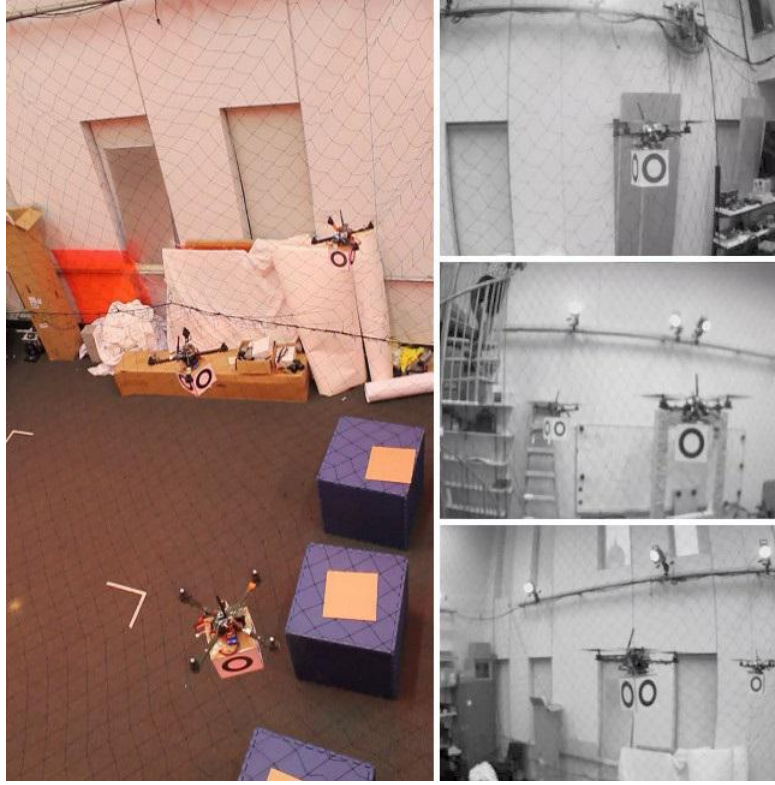


Figure 6.1: localisation quadrotor formation performed by the *Whycon* method

it, and possibly communicate with it). Moreover, one could find another way to calibrate the camera, as the chessboard didn't work properly on the embedded camera selected.

Working on the interaction between multiple quadrotors could be achieved by studying the *Crazyswarm*⁶ project. This is a ROS project to control multiple Crazyflie at once with only one crazyradio. The implementation in the official Crazyflie ROS project is in progress.

6.2 Conclusion

This project was to create a controller that represent a research tool that the Crazyflie is. This was done by presenting the state of art, as well as the theories applied to make this project possible. The most valuable parts for the theoretical approach are the localisation and the regulation algorithms implemented in the firmware of the quadrotor. The ROS project used for the experimentation is also an important tool to interact with the drone reliably.

⁶<https://github.com/USC-ACTLab/crazyswarm>

Eventually, this work highlights what could be interesting to work onto.

It is important to insist on the fact it is an open-source project with an active community. Stayed tune to new community progresses is a necessity.

6.3 Outcomes on my project

As the place of drones in today's society is more and more prominent, the need of engineers that have knowledges in these devices should increase. Therefore, working on the Crazyflie is a good opportunity to improve my professional profile. Moreover, I learned more about localisation and regulation, which are important tools a robotic engineer has to master. Finally, the interaction I had with the other members of the research centre permit me to learn more about organisation in this type of infrastructure.

Bibliography

- [1] Wolfgang Hoenig, Christina Milanes, Lisa Scaria, Thai Phan, Mark Bolas, and Nora Ayanian. Mixed reality for robotics. In *IEEE/RSJ Intl Conf. Intelligent Robots and Systems*, pages 5382 – 5387, Hamburg, Germany, Sept 2015.
- [2] Manon Kok, Jeroen D. Hol, and Thomas B. Schön. Using inertial sensors for position and orientation estimation. *Foundations and Trends in Signal Processing*, 11(1-2):1–153, 2017.
- [3] T. Krajník, M. Nitsche, J. Faigl, T. Duckett, M. Mejail, and L. Přeučil. External localization system for mobile robotics. In *16th International Conference on Advanced Robotics (ICAR)*, Nov 2013.
- [4] Tomáš Krajník, Matías Nitsche, Jan Faigl, Petr Vaněk, Martin Saska, Libor Přeučil, Tom Duckett, and Marta Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 2014.
- [5] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays, April 2010.
- [6] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011.
- [7] Mark W Mueller, Michael Hamer, and Raffaello D’Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, May 2015.
- [8] Mark W Mueller, Markus Hehn, and Raffaello D’Andrea. Covariance correction step for kalman filtering with an attitude. *Journal of Guidance, Control, and Dynamics*, pages 1–7, 2016.
- [9] Matias Nitsche and Tomáš Krajník. WhyCon: Stable release, GitHub. May 2014.

- [10] Matias Nitsche, Tomáš Krajník, Petr Čížek, Marta Mejail, and Tom Duckett. Whycon: An efficient, marker-based localization system. In *IROS Workshop on Open Source Aerial Robotics*, 2015.

Appendix A

Archived work

You will find in the same archive than this report all the files necessary to start working from where a stopped. You should have:

- An installation tutorial
- The two virtual machines (as presented in the tutorial)
- The Solidworks parts of an embedded camera casing
- the assembly instructions for this casing

Edit: Some of the above files aren't attached to fit the repository size. Please, contact me (fabrice.poirier@ensta-bretagne.org) for more information.

Appendix B

Flight videos

Two videos show what look like autonomous flights with the on-board Kalman filter fed by an external camera (position determined by whycon). The parameters were:

- **X-Y-Z__0.0-0.0-0.1.mp4** : reach position $(x,y,z) = (0.0,0.0,0.1)$ for 10 seconds, then lands at $(0.0,0.0,0.0)$
- **X-Y-Z__0.0-0.1-0.1.mp4** : reach position $(x,y,z) = (0.0,0.1,0.1)$ for 10 seconds, then lands at $(0.0,0.0,0.0)$

Appendix C

Evaluation report


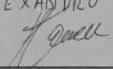
ERASMUS+  Erasmus+

Table D - Traineeship Certificate by the Receiving Organisation/Enterprise

Name of the trainee : <person.surname><person.first_name>	POIRIER Fabrice
Name of the Receiving Organisation/Enterprise :	University of Manchester
Sector of the Receiving Organisation/Enterprise :	School of Electrical and Electronic Engineering
Address of the Receiving Organisation/Enterprise [street, city, country, phone, e-mail address], website:	Oxford Road, Manchester M13 9PL www.manchester.ac.uk UK
Start date and end date of traineeship: from [day/month/year] to [day/month/year]	from 29/08/18
Traineeship title:	Interval Robotics
Detailed programme of the traineeship period including tasks carried out by the trainee:	Learning and application of a stabilizer implementation on a quadcopter, communicating with a computer and ROS middleware
Knowledge, skills (intellectual and practical) and competences acquired (achieved Learning Outcomes):	<ul style="list-style-type: none"> - Python, C++, C, ROS - Localisation (Extended Kalman Filter, complementary filter), regulation.
Evaluation of the trainee :	The work was satisfying. He worked on application of new techniques on new projects.
Date :	24/8/18
Name and signature of the Supervisor at the Receiving Organisation/Enterprise:	DR. ALEXANDRU STANCU 

Kit mobilité de stage 2017/2018 V1