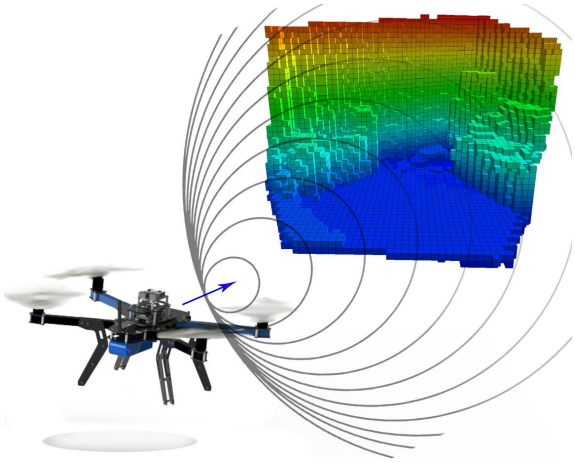
TÉCNICO LISBOA

academia militar



Integrated Architecture for Vision-based Indoor Localization and Mapping of a Quadrotor Micro-air Vehicle

Nuno Miguel dos Santos Marques

Thesis to obtain the Master of Science Degree in

Electrical and Computers Engineering

Examination Committee

Chairperson:	Professor Doutor João Sequeira
Supervisor:	Professor Doutor Rodrigo Ventura
Member of the Committee:	Professor Doutor Eduardo da Silva

October 2014

“You can know the name of a bird in all the languages of the world,
but when you’re finished, you’ll know absolutely nothing whatever about the bird...
So let’s look at the bird and see what it’s doing - that’s what counts.”
Richard P. Feynman

Acknowledgments

The work done for this thesis is the result of a great passion for drones subject and their applications. Believing that in the near future, the drones will be considered a personal object that each of us will want to have, and for seeing countless applications arise every day that involve its use, I decided to move to a personal investment in this area.

As such, all the hardware used in this thesis was a personal investment, with which I intend to continue working even after the thesis delivery and its defence. Also, while being connected to a branch of the Portuguese Armed Forces, I strongly believe that the potential applications, which are the result of development carried out for this thesis, but also what can be accomplish in the future work, will be a great asset to the Armed Forces to accomplish its different missions, which include SAR, aid in civil protection and other non-warlike applications.

Obviously, having reached this stage of my education, not only as a future engineer, but as a military and, especially, as a person, is due to the effort and personal sacrifice, but also to the people and institutions that were part of my life.

So, first, the biggest thanks goes to my family, being the foundation of my education and my guide to happiness throughout my life. Second, I want to thank the Military Academy, for the excellence of education and for giving me the opportunity to develop myself as a better person, better soldier and a better example for myself. Thirdly, I want to thank my girlfriend for all the support and patience for not being able to give her the time she deserves, because of having to develop this thesis.

Also, a special thanks to my Autonomous System course group members, who participated in the development of the support work that helped develop what was achieved in this thesis: João Santos, Miguel Morais, Rafael Pires and Tiago Fernandes.

A special thanks to Délio Vicente, for his professionalism and for helping me with the full 3D model of the quadrotor and to Ferdinand Meier and Maurício Martins, for the constrution of the quadrotor shield.

I also want to thank, in a more general way, the following individuals, who by e-mail, chat, forums or social networks, helped me move forward and develop the various components of my thesis in both theoretical component and the practical component: Felix Endres (University of Freiburg), Vladimir Ermakov (Carnegie Mellon), Anton Babushkin (PX4 Dev team), Lorenz Meier (ETH Zürich), Alessio Tonioni (University of Bologne), Thomas Gubler (PX4 Dev Team), Ji Zhang (University of Pittsburgh), Glenn Gregory (SG Controls Pty Ltd), Tony Baltovsky (University of Ontario), Kabir Mohammed (St. Xavier's Collegiate School) and Rainer Küemmerle (University of Freiburg). The same gratitude goes to the nearest community: Henrique Silva (ISR), Filipe Jesus (ISR), João Mendes (ISR) and, obviously, to my thesis advisor, Professor Rodrigo Ventura. I apologize if I forgot someone.

Kind regards to everyone! Take care,

Nuno Miguel dos Santos Marques

Tenente-Aluno de Transmissões

Exército Português

Azambuja, Novembro de 2014

Resumo

ATUALMENTE, os sistemas de pilotagem autónoma de quadricópteros estão a ser desenvolvidos de forma a efetuarem navegação em espaços exteriores, onde o sinal de GPS pode ser utilizado para definir *waypoints* de navegação, modos de *position* e *altitude hold*, *returning home*, entre outros.

Contudo, o problema de navegação autónoma em espaços fechados sem que se utilize um sistema de posicionamento global dentro de uma sala, subsiste como um problema desafiante e sem solução fechada. Grande parte das soluções são baseadas em sensores dispendiosos, como o LIDAR ou como sistemas de posicionamento externos (p.ex., Vicon, Optitrack). Algumas destas soluções reservam a capacidade de processamento de dados dos sensores e dos algoritmos mais exigentes para sistemas de computação exteriores ao veículo, o que também retira a componente de autonomia total que se pretende num veículo com estas características.

O objetivo desta tese pretende, assim, a preparação de um sistema aéreo não-tripulado de pequeno porte, nomeadamente um quadricóptero, que integre diferentes módulos que lhe permitam simultânea localização e mapeamento em espaços interiores onde o sinal GPS é negado, utilizando, para tal, uma câmara RGB-D, em conjunto com outros sensores internos e externos do quadricóptero, integrados num sistema que processa o posicionamento baseado em visão e com o qual se pretende que efectue, num futuro próximo, planeamento de movimento para navegação.

O resultado deste trabalho foi uma arquitetura integrada para análise de módulos de localização, mapeamento e navegação, baseada em hardware aberto e barato e *frameworks* state-of-the-art disponíveis em código aberto. Foi também possível testar parcialmente alguns módulos de localização, sob certas condições de ensaio e certos parâmetros dos algoritmos. A capacidade de mapeamento da *framework* também foi testada e aprovada. A *framework* obtida encontra-se pronta para navegação, necessitando apenas de alguns ajustes e testes.

Palavras-chave: veículos aéreos, não-tripulados, pequeno porte, quadricóptero, odometria visual, SLAM-6D, fusão sensorial, mapeamento, navegação interior, planeamento de movimento

Abstract

NOWDAYS, the existing systems for autonomous quadrotor control are being developed in order to perform navigation in outdoor areas where the GPS signal can be used to define navigational waypoints and define flight modes like position and altitude hold, returning home, among others.

However, the problem of autonomous navigation in closed areas, without using a global positioning system inside a room, remains a challenging problem with no closed solution. Most solutions are based on expensive sensors such as LIDAR or external positioning (f.e. Vicon, Optitrack) systems. Some of these solutions allow the capability of processing data from sensors and algorithms for external systems, which removes the intended fully autonomous component in a vehicle with such features.

Thus, this thesis aims at preparing a small unmanned aircraft system, more specifically, a quadrotor, that integrates different modules which will allow simultaneous indoor localization and mapping where GPS signal is denied, using for such a RGB-D camera, in conjunction with other internal and external quadrotor sensors, integrated into a system that processes vision-based positioning and it is intended to carry out, in the near future, motion planning for navigation.

The result of this thesis was an integrated architecture for testing localization, mapping and navigation modules, based on open-source and inexpensive hardware and available state-of-the-art frameworks. It was also possible to partially test some localization frameworks, under certain test conditions and algorithm parameters. The mapping capability of the framework was also tested and approved. The obtained framework is navigation ready, needing only some adjustments and testing.

Keywords: MAV, quadrotor, visual odometry, 6D-SLAM, sensor fusion, mapping, indoor navigation, motion planning

Contents

List of Tables

List of Figures

Abbreviations

ARM Advanced RISC Machine
COM Computer-On-Module
CRC Cyclic redundancy check
DOF Dimensions of freedom
EKF Extended Kalman Filter
ENU East-North-Up
ESC Electronic Speed Controller
FCU Flight Control Unit
FPV First Person View
GPS Global Positioning System
iSAM Incremental Smooth and Mapping
IMU Inertial Measurement Unit
ISR Intelligence, Surveillance and Reconnasissance
ITU International Telecommunication Union
KLT Kanade-Lucas-Tomasi feature matching
KPIECE Kinodinamic Planning by Interior-Exterior Cell Exploration
MAV Micro Air Vehicle
PID Proportional-Integral-Derivative
QVGA Quarter Video Graphics Array
NED North-East-Down
ORB ORiented Binary robust independent elementary features
PRM Probability RoadMap
RANSAC Random Sample Consensus
RGB-D Red, Green, Blue - Depth
ROS Robot Operating System
RPM Remotely piloted vehicles
RPY Roll, Pitch, Yaw
RRT Rapidly Exploring Random Trees
SIFT Scale-invariant feature transform
SLAM Simultaneous Localization and Mapping
SURF Speeded-Up Robust Features
SXGA Super Extended Graphics Array
sUAS small Unmanned Aircraft System
TOF Time-Of-Flight
UAV Unmanned Air Vehicle
UTM Universal Transverse Mercator
VGA Video Graphics Array

Chapter 1

Introduction

1.1 Motivation

A MAV system capable of doing autonomous navigation in confined areas still is a challenge to the researchers and designers that are looking to create a reliable and during system, given that there are many variables that come to the challenge.

Since the starting issue of this kind of systems is the localization, and so that this work can be continued in the future, a pre-framework/hardware selection process and a comparison between the existent state-of-the-art localization is indeed needed, so to evaluate its performance on the pose estimation of the quadrotor, and it can properly be forwarded to a framework that, in combination with the mapping capabilities, can perform the planning of a path issued by the user in a static environment, where the volume occupation of the obstacles and the localization of the MAV is known.

1.2 Objectives and Contributions

The global objective of the work done for this thesis is to provide a system able to navigate autonomously a micro-air vehicle, more specifically, a quadrotor, within a 3D environment that is being online reconstructed using data from a RGB-D sensor mounted on-board the quadrotor. The particular goal for the thesis itself is to prepare a system for the navigation task, resulting from the integration of different subsystems, having as a milestone provide a consistent comparison of different localization frameworks and testing the mapping capability inside a subsystem that is able to, after receiving localization of the vehicle and mapping from the environment, can proceed to navigation.

The navigation and path planning tasks will be active based on the estimated pose of the vehicle relative to the constructed map given by the sensor fusion of the main sensors aboard the quadrotor: a 10DOF IMU together with an RGB-D camera and a facedown Optical Flow with Sonar unit to give a 3D pose and velocity estimate of the quadrotor.

The discussion from the theoretical point of view will focus primarily on the state-of-the-art methods for vision-based localization and three-dimensional reconstruction of an environment. Navigation algo-

rithms that can be used to control of the quadrotor will also be summarily described, since the main point of attention was given to the localization and mapping issues.

From the practical point of view, instead, the ultimate goal will be the realization of a system able to autonomously fly a quadrotor within an environment that is online reconstructed in a map. For that, a comparison between different localization algorithms is done, so that the performance of both can be compared in terms of localization and time. The navigation task, which was already proved to work in a simulation environment, was already prepared to be tested in this framework, but it will be forwarded for future work.

Since the purpose is not to create new solutions to this matter, some open-source solutions are used and integrated to get a localization and mapping system working, so it can serve as a starting point and test bench for future algorithms and implementation works, not only in this tasks, but also to allow advancing to navigation tasks.

1.3 State-of-the-art

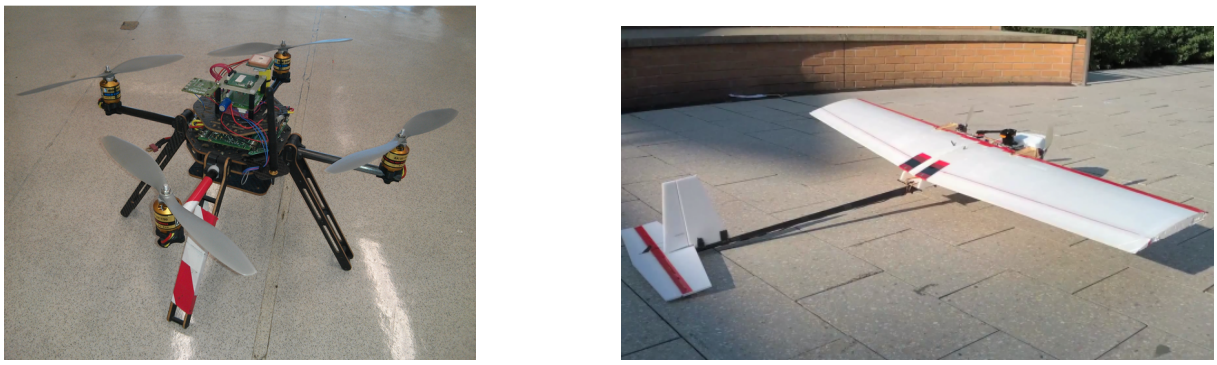
Small Unmmaned Aircraft Systems (sUAS), or usually called MAVs (micro aerial vehicles), are UAV's with most of the capabilities of large UAV's but with smaller footprints. Are the most common type of UAV's since they are also already present in the R/C community and already have several civilian applications besides the common military ones for which they were already used.

One of the primary applications for a sUAS is for ISR (Intelligence, Surveillance, and Reconnaissance), but there are many applications. Some examples include: determining the direction a fire is moving on hot spots, monitoring a forest for new fires, scouting dangerous areas (situational awareness), aiding the pursuit of suspects, intruder detection/security, search and rescue (SAR) (especially in inaccessible areas), inspecting tall objects (towers to find damage, eagles nests to count eggs), finding invasive plants, counting endangered wildlife, surveillance of pipelines or high tension lines or volcanoes, accident report assistance, real time mapping, mapping with Colour Infrared (CIR), Short Wave Infrared (SWIR), Long Wave Infrared (LWIR), cameras and environmental testing for radiation and chemical leaks, just to name a few.

The common type of these vehicles are the remotely piloted vehicles (RPV), which are UAV's that are stabilized and have sensors, but do not operate with an autopilot. Some of them are equipped with what is called FPV (first person view), which allows the pilot to fly by monitoring a camera mounted on the aircraft. Both RPV and FPV are complementary to sUAS because UAS seldom fly completely autonomously, especially during take-off and landing, when we are considering outdoor vehicles.

sUAS primarily are divided into two type of vehicle categories: the multirotors (Figure 1.1 (a)) and fixed wing (Figure 1.1 (b)). Given that this work has as its focus on indoor navigation, the primary focus will be on the first ones, since the fixed wing are usually bigger vehicles and the type of flight made is not adequate for indoor flights.

Autonomous MAVs (micro aerial vehicles), primarily multirotors, are getting more and more attention within robotics research, since their applications are tremendous. One of them is easily identified: SAR



(a) Multirotor (b) Fixed Wing

Figure 1.1: Types of vehicles comparison

[11] in inaccessible areas, where a common pilot will not have access (a FPV system in this case is impossible to use given that the signal, for example, in a mine, will be lost).

From all multirotor types, a quadrotor MAVs is an ideal choice for autonomous reconnaissance and surveillance because of their small size, high manoeuvrability, and ability to fly in very challenging environments. In any case, for getting true autonomy, a MAV must neither rely on external sensors (i.e. an external tracking system) or on off-board processing to an external computer for autonomous navigation.

To perform these tasks effectively, the quadrotor MAV must be able to do precise pose estimation, navigate from one point to another, map the environment, and plan navigation and (if able) exploration strategies. Ideally, all these processes have to run on-board the MAV, especially in GPS-denied environments.

One of the most essential problems for autonomous MAVs is localization or pose estimation. Common ground robots can just stop and wait for its pose estimate but a MAV typically cannot do it. In order not to crash, it needs reliable pose estimates in 6D (position and orientation) at a high frequency. If the autonomous MAV should also be able to navigate in previously unknown environments, it has to solve the SLAM (simultaneous localization and mapping) problem.

Many research projects targeted some of these abilities using different kinds of technologies. One of those is using laser range finders [4, 11, 30, 47], which usually is an expensive piece of device given that it gives good range and precision. Other technology can be the use of artificial markers [8, 13]. Common sense says that this is can be used on a lab environment but in a real world situation, we will not have markers everywhere so the MAV can position itself and navigate. Same thing with motion capture systems like VICON [24, 38] or OptiTrack [23]. They are extremely expensive systems with the highest precision of all the technologies that can be used, but again, they are expensive and the world does not have those system cameras spread everywhere. Other systems like [49] use low cost sensors, but that mean a lower precision on the vehicle localization.

On evaluating this systems, we see that the own sensors becomes an impediment factor when designing a MAV that can achieve fully autonomy in mapping and/or exploration. The primary disruptive facts are the weight and the power consumption, where an equilibrium must be found to potentiate both autonomy and reliability of the system.

A higher reliable system mostly is a system which is semi-autonomous, meaning it depends on

external processing units, which removes its complete autonomy. That is the case of using an external global positioning system, like a motion capture system. Or it is a heavy system with a heavy payload, which turns out to be a less flexible vehicle, with a higher power consumption. An example of it is using a laser range finder, which considerable weight and power consumption pose a problem for MAVs with stringent payload and power limitations.

Projects with RGB-D cameras [5, 29, 46], i.e. cameras that provide registered colour and depth images, seem to be the perfect approach for the indoor navigation task: They are small, light-weight, and provide rich information in hardware already, without requiring additional expensive post processing computations by the on-board computer, like stereo cameras do [20, 53]. Still, the applications in [5, 29, 46] lack a general explanation on how to transpose to a usable open-source system. [5] uses a very expensive platform to get the job done.

The processing power also becomes an issue when designing this kind of systems. Usually, a system that relays on external processing power does not have problems regarding this variable, since weight is reduced and the higher power consumption is on the external system. But that means that the system itself it is not self-dependent, relaying on an external system so it can localize itself and proceed with its autonomous behaviour. In the other hand, adding higher processing units to the MAV itself may be the best solution, but then it must be found a balance between the processing power, the weight of the processing unit and the power consumption.

Embedded platforms capable of supporting Linux type of OS are, right now, a great success among the creative and researching community. Their high processing capabilities, low weight and footprint and low power consumption allows great usage on robotics projects, especially when weight and flexibility of the system are essential for its proper work, which is the case of MAVs.

For understanding how this can be achieved, localization and mapping must be divided into two different problems and then combined in a structured way.

The problem of localization can be solved by visual odometry if complemented with a proper correction [10, 28, 45, 48], given that it accumulates error. In case of ground robots, this odometry can be measured by the use of the odometry of the wheels. In the case of aerial robots, since they do not have wheels, a different type of odometry must be used. Since cameras can be used as sensors, visual odometry is the common approach for robot localization, since it allows the egomotion estimate using as input single like in [10, 21, 28] or multiple cameras like in [20, 53].

Mapping, on the other hand, is a different issue. Many robotics applications require a three-dimensional map of the environment that surrounds them created from the data recorded by sensors. But, to create and maintain this amount of data in an efficient manner is not a trivial operation, so a good method of management of the map should ensure the ability to model free space, occupied and not yet explored, quick and easy creation and update of it, representation of the information in a probabilistic way, efficiency and multi-resolution.

The combination of map with 3D info of the environment and the localization process given by visual odometry or SLAM give a full 6D estimation of the robot pose and motion, which can be forward for to a motion planning framework that allows a structured navigation of the vehicle in the environment.

1.4 Thesis Outline

So to understand the work done in this project, the following lines give a short description of what is addressed in each chapter.

In Chapter 2 are reviewed the basic concepts to quadcopters and its physics model and functioning, followed by a review methods of SLAM (graph-based SLAM, visual odometry, feature handling, bundle adjustment and graph optimization, pose estimation and sensor fusion), optical flow, mapping, navigation and path planning.

Chapter 3 is where the development part is stated. It presents the vision-based localization, mapping and navigation modules used and the conceptual solutions for the initial problems given. It presents an overview of the used software and hardware, which includes the quadrotor platform and configuration used, the external and inboard sensors and the integrated software framework used to build the system. Also, it is explained the module to communicate between the FCU and the computer.

The forth chapter focus on testing the localization and mapping modules on different test conditions and presenting its results.

The fifth and last chapter presents the conclusions and proposals for future work.

Chapter 2

Theoretical Background

This chapter presents the theoretical background behind the frameworks used on the project. It starts with an overview over the vehicle mathematical model, then follows with the explanation of some of the algorithms and techniques used for localization, mapping and that may be used for navigation.

2.1 Vehicle model

This subchapter regards an explanation of the vehicle being used, the coordinate frames used and the kinematics and dynamics mathematical models.

2.1.1 Introduction

A MAV, or more specifically, a quadrotor MAV, is a rotary wing aircraft with four motors/propellers located at the ends of a cross structure (Figure 2.1), with the same control capabilities as helicopters.

Figure 2.1: Quadrotor MAV used in this project

In contrast to helicopters, they stand out by virtue of their much simpler and thereby massively less sensitive mechanics. There are four motors, which are rigidly connected with two right and two

left-rotating propellers. They also differ from other multirotor UAV given that they are more flexible in manoeuvring than any of the others: tricopters are flexible, but their control is harder, and cannot carry much of a payload. Hexarotors and beyond can carry a larger payload, but are less flexible on movement.

The flight behaviour of a quadrotor is determined by the rotation speeds of each of the four motors, as they vary in concert, or in opposition with each other, which define their attitude and steering mechanism.

Doing its mathematical model allows defining a good description of the behaviour of a system, which can be used to predict the position and orientation of the quadrotor. The last can further be used to develop a control strategy, whereby manipulating the speeds of individual motors results in achieving the desired motion.

Defining a full mathematical model requires the definition of the vehicle kinematics, which regards the aspects of motion (position and orientation), providing a relation between vehicle's position and velocity, and the vehicle dynamics, which regards the application of forces that influence accelerations and, consequently, the vehicle motion.

2.1.2 Coordinates Reference Frames

Before moving, it is essential to it to specify the adopted coordinate systems and frames of reference and also how transformations between the different coordinate systems can be made.

There are three main frames to be considered [43]:

1. The **inertial frame**, which regards an earth-fixed coordinate system with the origin located on the ground, which usually is the local origin of the system when it starts, or can be defined by the user (using a position triplet of X,Y,Z coordinates or GPS coordinates). The representation is in Figure 2.2.

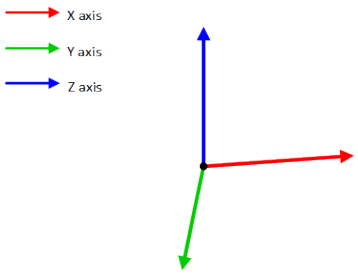


Figure 2.2: Inertial frame

2. The **body frame** has its origin located at the centre of gravity of the vehicle and as an alignment that considers the X-axis in the direction of the first motor and the Y-axis along the direction of the second motor. The Z-axis is where the cross product $Z^B = X^B \times Y^B$ happens. The representation is in Figure 2.3.

3. The **vehicle frame** is the inertial frame but with the origin on the centre of mass of the vehicle, but can have two possible configurations, depending if we consider a "X" type of motion or a "Plus" $(+)$ type. The "Plus" configuration has the X and Y of the vehicle frame aligned with the X and Y of the body frame, while with the "X" configuration, there is a yaw rotation, where the X-axis of the vehicle points in the direction of the midline between the two front motors. The representation is in Figure 2.4.

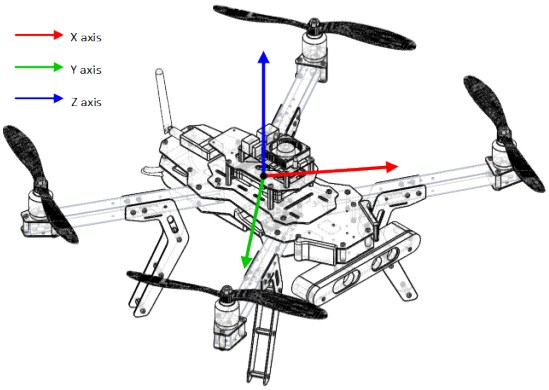


Figure 2.3: Body frame

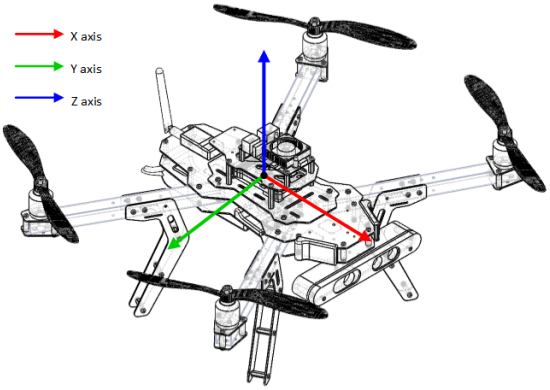


Figure 2.4: Vehicle frame

2.1.3 Kinematics

Let $\rho^I = [x^I, y^I, z^I]^T$ and $\Theta_I^T = [\phi^I, \theta^I, \psi^I]^T$ be the position in Cartesian coordinates and orientation (roll, pitch and yaw, which are the rotations around x, y and z-axis) of the vehicle relative to the inertial frame, respectively. Consider pose χ^I as $= (\rho^I, \Theta^I)$, i.e the combination of the position and orientation.

While the position of the vehicle is defined in the inertial frame and the velocity of the vehicle is defined on the vehicle frame, which it will be considered equal to the body-frame for simplicity, a rotation matrix must be defined so a frame transaction must occur. Then, considering $\nu^V = (\Upsilon, \omega)$, where $\Upsilon^V = [v_x, v_y, v_z]^T$ is the linear velocity and $\omega^V = [r_r, p_r, y_r]^T$ the angular rates of roll, pitch and yaw, relative to the vehicle frame, we get the following rotation matrix for the three Euler angles [35, 37]:

$$R(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, R(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}, R(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}, \quad (2.1)$$

which combination,

$$R_\Theta = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}, \tag{2.2}$$

corresponds to the final rotation matrix:

$$R_\Theta = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi s_\phi & s_\psi s_\phi + c_\psi s_\theta c_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & s_\psi s_\theta s_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\theta & c_\theta c_\phi \end{bmatrix}, \tag{2.3}$$

where $c_\alpha \equiv cos\alpha$ and $s_\alpha \equiv sin\alpha$. Given this rotation matrix, it's now possible to get a relation between the position vector ρ^I and the linear velocity vector Υ^V with,

$$\dot{\rho^I} = R_\Theta \Upsilon^V \Leftrightarrow \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = R_\Theta \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}. \tag{2.4}$$

And given the same process, it is done the same thing to the angular position and velocities, which gives the relation:

$$\omega^V = T_\Theta^{-1} \dot{\Theta}^I \Leftrightarrow \dot{\Theta}^I = T_\Theta \omega^V, \tag{2.5}$$

where T_Θ is a transformation matrix. This one is calculated through:

$$\begin{bmatrix} r_r \\ p_r \\ y_r \end{bmatrix} = T_\Theta^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + {R_\phi}^{-1} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + {R_\phi}^{-1} {R_\theta}^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}, \tag{2.6}$$

which, after inverting, we will get the following transformation matrix [12]:

$$T_\Theta = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\theta & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi c_\theta \end{bmatrix}, \tag{2.7}$$

where $t_\alpha \equiv tan\alpha$.

2.1.4 Dynamics

Considering the angular velocities of the four motors as a vector $\omega = [\Omega_1, \Omega_2, \Omega_3, \Omega_4]^T$ [15]. The four types of movement control will be described as a U vector, where $U_1 \equiv altitude$, $U_2 \equiv roll$, $U_3 \equiv pitch$ and $U_4 \equiv yaw$. So to be able to get the corresponding attitude change, the input for each motor must be changed accordingly. For this particular case, it will be considered a "cross" (X) type of motion to the quadrotor, since the used quadrotor in the project was designed for this type of motion, which means

the first motor Ω_1 is the top left motor and the rest is numbered in a clockwise way.

So, if one wants to change its altitude $U_1[N]$, all four motors must change their angular velocity at the same time, as we can see in Figure 2.5.

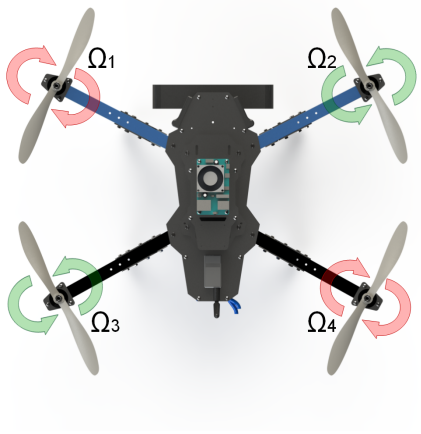
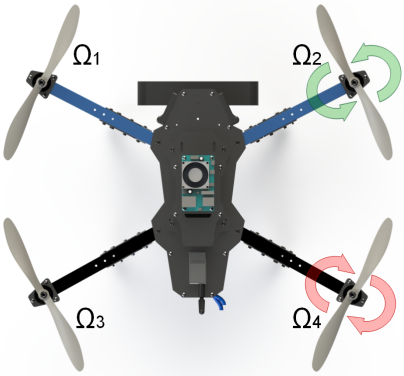
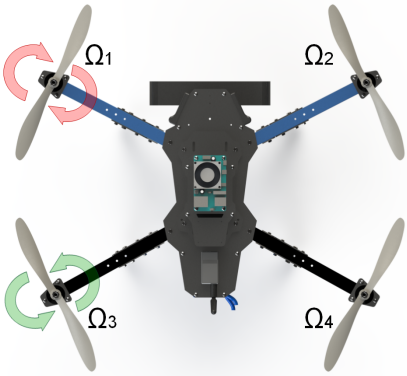


Figure 2.5: Altitude increase

The rolling movement $U_2[N.m]$ is obtained by increasing/decreasing the pairs Ω_1, Ω_4 and Ω_2, Ω_3, which depends if one wants to bend right (Fig. 2.6(a)) or left (Fig. 2.6(b)) respectively, which will change the ϕ angle.



(a) Move vehicle left

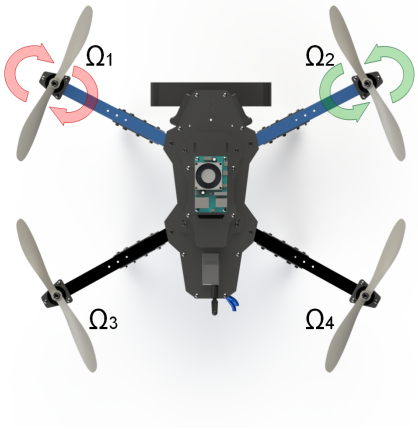


(b) Move vehicle right

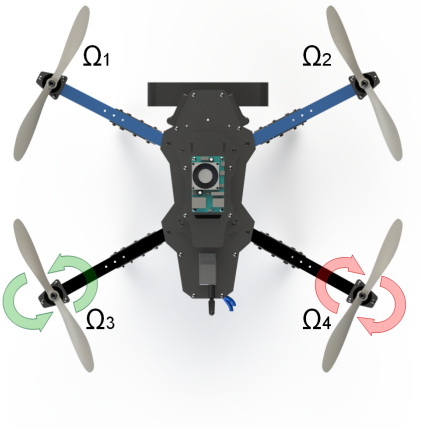
Figure 2.6: Vehicle roll movement

The pitching movement $U_3[N.m]$ is obtained by increasing/decreasing the pairs Ω_1, Ω_2 and Ω_3, Ω_4, which depends if one wants to bend backward (Fig. 2.7(a)) or forward (Fig. 2.7(b)) respectively, which will change the θ angle.

The yaw movement $U_4[N.m]$ is obtained by increasing/decreasing the pairs Ω_1, Ω_3 and Ω_2, Ω_4, which depends if one wants to turn right (Fig. 2.8(a)) or turn left (Fig. 2.8(b)), respectively, changing the ψ angle.

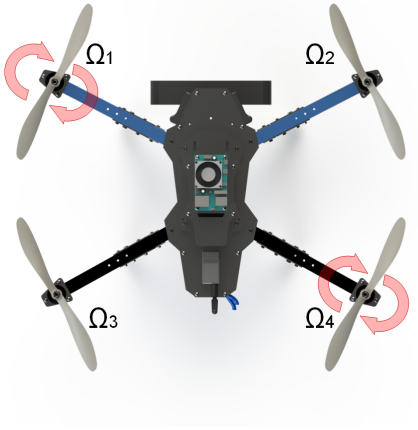


(a) Move vehicle backward

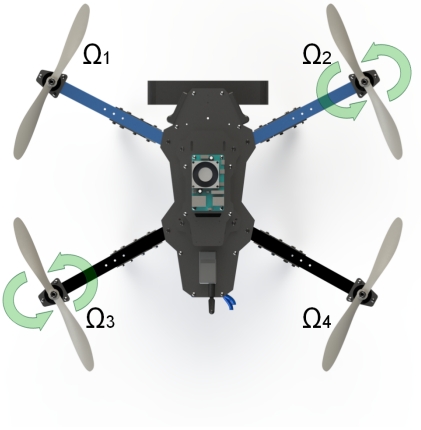


(b) Move vehicle forward

Figure 2.7: Vehicle pitch movement



(a) Rotate vehicle right



(b) Rotate vehicle left

Figure 2.8: Vehicle yaw movement

To consider forces and moments, the Laws of Newton must be applied [7, 12]. Considering Newton's second law, applied to the translational motion,

$$F^V = m(\dot{\Upsilon}^V + \omega^V \times \Upsilon^V), \tag{2.8}$$

and to rotational motion,

$$\tau^V = I\dot{\omega}^V + \omega^V \times (I\omega^V), \tag{2.9}$$

with m equals the vehicle mass in kilograms, $\dot{\Upsilon}^V$ the linear acceleration in meters per second squared, $\dot{\omega}^V$ the angular acceleration in radians per second squared, I a diagonal matrix which is the inertia matrix of the system, in Newton's-meter-second-squared. The inertia matrix is defined as follows, since

it is assumed a symetric configuration on the vehicle X and Y axis:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{2.10}$$

Just to complete the equations 2.8 and 2.9, it's defined the system entrance U^V vector and added the gravity force. That will result in a new set of equations, after some empirical deductions:

$$F^V = R_\Theta^T. \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} = \begin{bmatrix} mgs_\theta \\ -mgc_\theta s_\phi \\ -mgc_\theta c_\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} = \begin{bmatrix} mgs_\theta \\ -mgc_\theta s_\phi \\ -mgc_\theta c_\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix}, \tag{2.11}$$

$$\tau^V = \begin{bmatrix} \tau_x^V \\ \tau_y^V \\ \tau_z^V \end{bmatrix} = \begin{bmatrix} U_3 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} bl(\Omega_1^2 + \Omega_4^2 - \Omega_2^2 - \Omega_3^2) \\ bl(\Omega_3^2 + \Omega_4^2 - \Omega_1^2 - \Omega_2^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix}. \tag{2.12}$$

The combination of the kinematics equations 2.4 and 2.5 with 2.11 and 2.12 gives the complete mathematical model of the quadrotor:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = R_\Theta \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}, \tag{2.13}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = T_\Theta \begin{bmatrix} p_r \\ r_r \\ y_r \end{bmatrix}, \tag{2.14}$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} y_r v_y - r_r v_z \\ r_r v_z - y_r v_x \\ p_r v_x - r_r v_y \end{bmatrix} + g \begin{bmatrix} s_\theta \\ -c_\theta s_\phi \\ -c_\theta c_\phi \end{bmatrix} + 1/m \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix}, \tag{2.15}$$

$$\begin{bmatrix} \dot{r}_r \\ \dot{p}_r \\ \dot{y}_r \end{bmatrix} = \begin{bmatrix} \frac{I_{yy} - Izz}{I_{zz}} p_r y_r \\ \frac{I_{zz} - Ixx}{I_{yy}} r_r y_r \\ \frac{I_{xx} - Iyy}{I_{zz}} r_r p_r \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{zz}} U_2 \\ \frac{1}{I_{yy}} U_3 \\ \frac{1}{I_{zz}} U_4 \end{bmatrix}, \tag{2.16}$$

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ bl(\Omega_1^2 + \Omega_4^2 - \Omega_2^2 - \Omega_3^2) \\ bl(\Omega_3^2 + \Omega_4^2 - \Omega_1^2 - \Omega_2^2) \\ d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{bmatrix}, \tag{2.17}$$

where g is the gravity acceleration in $[m.s^{-2}]$, b is the propulsion coefficient in $[Kg.m.rad^{-2}]$, d is the binary coeficient of the propellers in $[Kg.m^2.rad^{-2}]$ and l is the distance from the centre of the quadrotor to the propellers in meters [12].

2.2 Localization

In the next sections, it is covered important aspects of pose estimation for robots, including the different concepts involved on visual odometry and visual graph-based SLAM algorithms, what is optical flow and how it aids on robot localization, and finally how can data from different sensor sources be merged into sensor fusion algorithms so that vehicle pose and motion can be predicted.

2.2.1 Introduction

In order to localize itself, a robot has retrieve both relative and absolute data measurements that provide feedback about the robot actions and the situation of the environment that surrounds it. The main issue usually falls on how the robot deals with noisy and unpredictable data that can distort the way the world is presented and how the vehicle is behaving.

Vehicle localization issue is a key problem when making autonomous robots, since it can be difficult for a robot to determine what to do next if it does not know where it is. There are many types of technologies and sensor sources available for robot localization, including GPS, odometry, active and passive beacons, sonar, among others, but improving one of them may imply more costs and additional processing power. That is why adding different cheap sensor sources and fuse them in some filter is the best and the usually adopted solution for localization and, consequently, for navigation.

Regarding the measurements, there are many types of sensors that provide positioning and distance data for the robot know how the environment is composed, is form and distances, the presence of obstacles, etc. All that data must be used in a way that the robot can plan, for example, its movements given the presence of dynamic entities and the dimensions of a given area. Also, besides its position, the robot must be aware of its attitude and motion, so a proper sensor measurement that, after its processing in a filter, can give a velocity estimation of the vehicle movement, is indeed welcoming. In this last case, optical flow algorithms are a way of determining the motion of the vehicle given image processing of camera streams.

In the last paragraph, the word “filter” was also used as a way of processing data. When we discuss filters in terms of robot localization we are considering sensor data filters that allow the fusion of different sensor sources to estimate the pose and motion of a robot in a given environment. The use of filters allows the improvement of the estimations besides giving a continuous estimation even if a sensor source fails. It also deals very well with the noise of the sensors by interpolating the data of different sensor sources on a probabilistic mathematical model.

2.2.2 Visual Odometry alone vs SLAM

Visual odometry is a type of method available in mobile robots that allows the pose and attitude estimation of the vehicle based on the data retrieved from cameras.

In the other hand, SLAM, or simultaneous localization and mapping, regards the pose estimation of the robot given the global estimate in relation with a map, which implies a incrementally map reconstruc-

tion and positioning relative to that map.

Visual odometry mainly aims at recovering the path incrementally, pose after pose, and potentially optimizing only over the last n poses of the path, estimates local motion and generally has unbounded global drift, which differs from SLAM, which in general wants to obtain a globally consistent estimate of the robot path [51]. That implies keeping a track of a map of the environment because it is needed to realize when the robot returns to a previously visited area, which is a process called loop closure. In any case, a SLAM algorithm requires the use of feature extractor and matching algorithms from image streams coming from vision sensors (considering visual SLAM algorithms), which are also present on the visual odometry algorithms.

In order to understand how each type of pose estimation methods work and have to offer, in the next sections, some basic concepts and techniques will be presented, mainly regarding the techniques used and tested on the project developed for this thesis.

Graph-based SLAM

If the autonomous MAV wants to able to navigate in previously unknown environments, it has to perform SLAM, which is an acronym for simultaneous localization and mapping. SLAM has been one of the major fields of research in mobile robotics during the past decades and it gives the ability to the robot to understand and perceive the unknown environment while moving through it and, at the same time, incrementally build a map of what has been explored, on which simultaneously determines its localization.

The solution of the SLAM problem has been one of the notable successes of the robotics community given the last researches of the robotics fields. It has been formulated and solved as a theoretical problem in a number of different ways and has also been implemented in a number of different research areas and applications as indoor robots, outdoor, air and underwater systems and, as is, it can be considered a solved problem.

However, some issues remain in practically implement a more general SLAM solutions and that can use perceptually rich maps as part of the algorithm itself. For this last issue, RGB-D cameras can be used. They provide both colour images and per-pixel depth estimates, which richness of data and the recent development of low-cost sensors have been combined on an excellent opportunity for mobile robotics research [20], which also include MAV's.

The three main approaches to the SLAM problem consider: firstly, a filter approach solution, which use a Kalman Filter to estimate the robot pose; secondly, also a filter solution, but using the so called Monte Carlo solution, or Particle Filter; the third approach, which is the one that is going to be consider in this project given the used algorithms, is the graph-based SLAM solution.

The graph-based SLAM [22] describes the robot localization as a graph, where every node corresponds to a robot position and to a sensor measurement and the edges between two nodes are considered data-dependent spatial constraints between the two nodes. This constraints are obtained from observations of the environment or from movement actions carried out by the robot.

The construction of such graph allows that a map can be computed by finding the spatial config-

uration of the nodes that is considered, by the several iterations, mostly consistent with the raw measurements being modelled by the edges, with this last ones being seen as “virtual measurements”. The modulation of the graph is consistent with a probabilistic model, where the edges have the probability of a given pose of the robot given the relative measurements of the nodes with which it is connecting.

A graph-based SLAM is usually divided into three main modules, each one with its proper function: the frontend framework, the backend framework and the map representation.

In the case of RGBD SLAM[1], or in the case of CCNY RGBD Tools[2], which is very similar, the structure of the graph-based SLAM architecture is based on the schematic of Figure 2.1 [14]. CCNY RGBD though lacks a backend framework, as explained in 3.4.1.

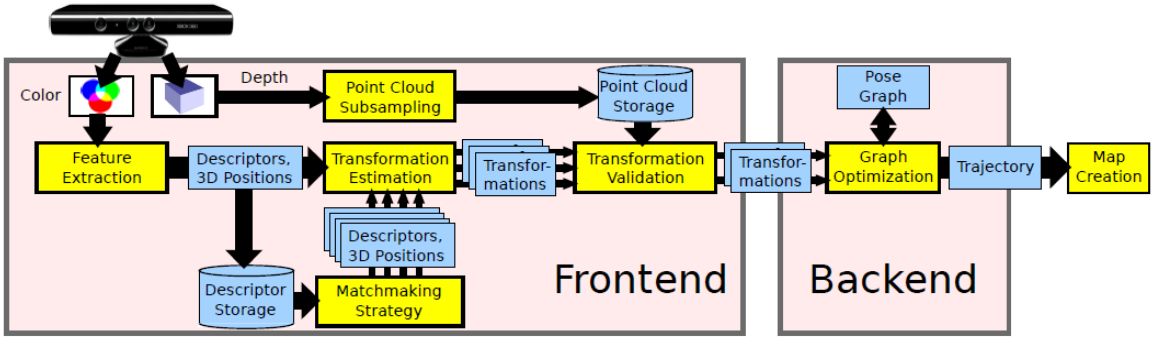


Figure 2.9: Graph-based SLAM architecture

As said, there are three main components of the framework. The frontend framework includes all the sensor data processing that allow the generation of specific geometry relations between the data being received, which includes feature detection, extraction and matching. This also includes a way of dealing with motion between two observations, which requires a visual odometry algorithm to deal with it.

In the case of the algorithms that uses RGB-D data, as in the case of the SLAM frameworks used on this project, that means having a way of determine landmarks by extracting a high-dimensional descriptor vector from the image data and store it together with their location relative to the observation pose. The depth info is then used as a way of validate the transformations between the camera link and the 3D descriptors being retrieved by the feature extraction algorithms being used.

To deal with uncertainties introduced by sensor measurements, it is used a graph that represents the geometric relations and their uncertainties, which can then be optimized in the backend using graph optimization techniques, allowing to obtain a maximum likelihood solution for the represented robot trajectory.

Monocular visual odometry

Visual odometry is implemented in algorithms nowadays that allow the use of different kinds of image processing techniques, depending on the current used sensor and what type of features can be used, to allow the robot to localize itself on the environment. The idea is to compute and retrieve the egomotion

[1] F. Endres. RGBDSLAM v2. `http://felixendres.github.io/rgbdslam_v2`. Online.
[2] ROS Wiki. CCNY RGBD Tools. `http://wiki.ros.org/ccny_rgbd_tools`. Online.

based on the features being retrieved and matched, and the relation between the same features from one image to another.

In specific to a MAV system like this, it is recommended the use of an algorithm that is able to be relatively robust but that can assure processing at high rates, which most of the current online image processing techniques for motion perception and localization are not, given the higher processing required so they can work, which usually means using an external processing unit for the data stream or the use of high power consumer COM.

There exists two types of visual odometry algorithms based on the type/number of cameras being used: the monocular visual odometry, which can use a typical camera with just one oculus to process the egomotion, or stereo visual odometry, which uses a special camera with two or more oculus or, instead, multiple cameras with image merging.

The approach used on this project, and since it is specifically used and RGB-D type of camera, uses a monocular visual odometry algorithm that adds depth data to improve the pose estimation. One of the methods used on it is explained in 2.2.4. section.

2.2.3 Optical Flow

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera[3], resulting on a 2D vector field where each vector is a displacement vector showing the movement of points from a frame to the other. This vector is a result of the calculation of partial derivatives with respect to the spatial and temporal coordinates.

The used methods allow the estimation of motion as either instantaneous image velocities or discrete image displacements in a sequence of ordered images. That is why it is very useful on robot appliances since it allows the egomotion estimation of the robot given the estimated velocity of pixels from a frame to another in a image stream.

The most known methods to give the solutions to the optical flow equations are the Lucas-Kanade method [42], which obtains the solution of the optical flow equations by assuming a constant value of flow for each neighbours of a certain pixel and calculating the neighbour flows iteratively by a least squares approximation, and the Horn-Schunck method [42], which applies a constant value of smoothness in all the pixels, represented by a energy function, so to minimize distortions and choose the solutions for the flow equations which give more ”smoothness”.

Metric velocity calculation

For consideration of what is used in this project, it is going to be explained how the optical flow module being used gives an estimate of 3D velocity of the camera frame relative to the image plane, based on the work developed in [26].

[3]OpenCV 3.0 dev doc. Optical flow. `http://docs.opencv.org/trunk/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html`. Online.

Assuming

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{2.18}$$

as a point in the camera reference frame, the Z-axis being the optical axis, f the focal length and the centre of the projection as the origin of the frame. The pixel coordinates projected in the image plane are represented by

$$p = f\frac{P}{Z}. \tag{2.19}$$

Given that the focal length gives the distance between the origin and the image plane, that results on a coordinate

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.20}$$

which is constant. The relative motion can then be computed with the equation

$$V = -T - \omega \times P, \tag{2.21}$$

with ω being the angular velocity and T the motion translational component. Applying the derivative with respect to time will give a relation to the velocity of P (in camera frame) and velocity of p (in image plane). The result of that relation is then applied and the result is the motion field equations:

$$v_x = \frac{T_z x - T_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy - \omega_y x^2}{f}, \tag{2.22}$$

$$v_y = \frac{T_z y - T_y f}{Z} - \omega_x f + \omega_z x + \frac{\omega_x y^2 - \omega_y xy}{f}. \tag{2.23}$$

Since the motion field is composed by a translational part plus a rotational part, but given that rotational parts are Z-independent, the translational component can be calculated using the focal length f and the distance to plane Z, assuming that the rotational velocity is zero or it is compensated by a calculation of a gyroscope measurement:

$$v_{trans} = v\frac{Z}{f}. \tag{2.24}$$

In the case of a constant distance to the plane, it is considered

$$v_x = \frac{-T_x f}{Z} - \omega_y f + \omega_z y, \tag{2.25}$$

$$v_y = \frac{-T_y f}{Z} - \omega_x f + \omega_z x, \tag{2.26}$$

where terms divided by the focal length are ignored given that the order of magnitude is smaller compared with the other terms. Having the angular rates given by a gyroscope measurements, the previous velocities can be compensated, assuming the considered ignored terms.

An idea of how this calculations apply to a MAV movement is described on Figure 2.10[4].

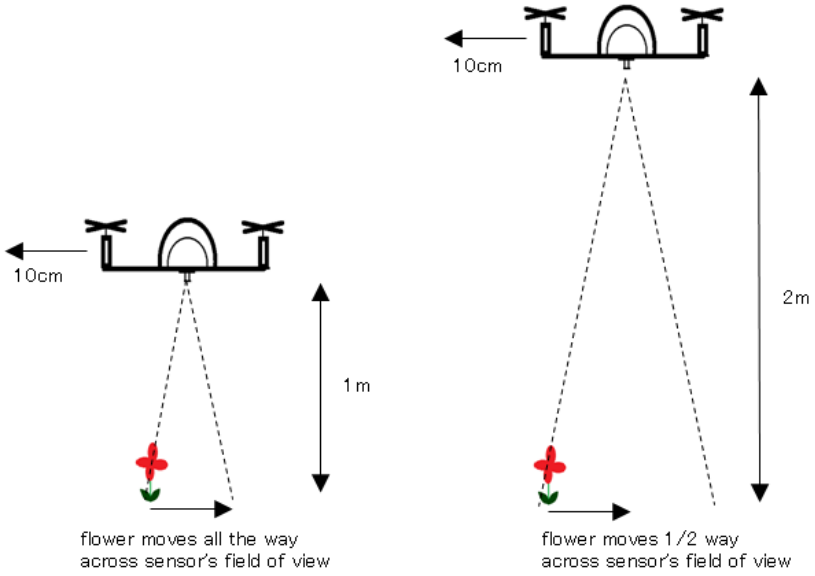


Figure 2.10: Concept applied to quadrotor movement

2.2.4 Feature handling

Feature handling is an essential part of image processing which deals with the detection and representation of interesting parts of an image or a stream of images into usable compact data vectors. In the case of vision-based localization systems, it is essential to have techniques that allow detection, extraction and matching of the computed features, so to allow the robot to localize and estimate its motion given the data being processed by its camera sensors.

There are various methods that can be used for this purposes, but the next section will cover the one used in feature extraction and matching on the SLAM frameworks used on this thesis: SURF [6]. In the case of the visual odometry framework, it is used the Kanade-Lucas-Tomasi [50] for pose estimation, which is allied to the optical flow techniques used for feature tracking but applied to detection and tracking of Harris corners[5].

SURF

SURF [6], or Speeded-Up Robust Features, is a method that allows feature detection and extraction. It is based on the descriptors of the images and it is invariant relative to rotation and scale. It proceeds to the detection of the features by firstly, calculating the integral images by

$$\boldsymbol{I}_{\Sigma}(x,y) = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} I(i,j), \tag{2.27}$$

[4]APM Wiki. Optical flow sensor. `http://copter.ardupilot.com/wiki/optical-flow-sensor`. Online.
[5]OpenCV 3.0 dev doc. Harris corners. `http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html`. Online.

and then applying the previous equation to compute the Hessian Matrix, the convolution over a second order Gaussian derivative [58], for each it calculates the determinant:

$$H(\boldsymbol{X}, \sigma) = \begin{bmatrix} L_(xx)(\boldsymbol{X}, \sigma) & L_(xy)(\boldsymbol{X}, \sigma) \\ L_(yx)(\boldsymbol{X}, \sigma) & L_(yy)(\boldsymbol{X}, \sigma) \end{bmatrix}. \tag{2.28}$$

After that, it uses non-maximum suppression for choosing the local maximums. All the descriptors have a certain intensity content which allows them to be distinguishable. They are computed considering the invariance on rotation, which is obtained by the Haar wavelet on the X and Y directions, and then, after being computed, they are added in the feature vector properly normalized, which adds the invariance on scale.

For feature matching of SURF image descriptors, it can be used some sort of minimization of Euclidian distances algorithm, which minimize the distances between each feature. This algorithm can be FLANN [40], brute-force algorithms, among others.

For both of the SLAM frameworks being tested on this thesis, it is used SURF for feature extraction and matching based on brute-force.

KLT feature tracker

The basic idea behind KLT, Kanade-Lucas-Tomasi feature tracker [33, 42, 50] is given a sequence of two grey scaled images, where each pixel has a grey scale value, find a certain value u where the grey scale in the two images is the same, which is represented by the sum of the optical flow of the feature being analysed and the coordinates of the first point.

For that, it is applied a matching function which corresponds to a window where to track those features from a frame to another. The idea is then to find that certain value u that minimizes the matching function.

In the case of the KLT tracker being used on the visual odometry framework, the selected features to track are Harris corners. Figure 2.11 represents an example of some feature points being tracked along a sequence of images [50].

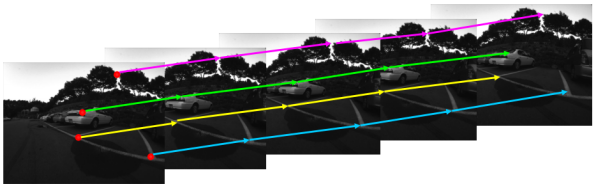


Figure 2.11: Feature tracking example

2.2.5 Bundle adjustment and graph optimization

The idea of bundle adjustment [3] is to find 3D point positions and camera parameters that minimize the reprojection error, formulated as a non-linear least squares problem, given a set of image features and

its relations. It is considered the final step of a optimization/reconstruction problem, and it is very useful for correcting estimations from visual odometry algorithms.

Also, SLAM problems, as in the case of graph-based SLAM, can be optimized using techniques that allow the optimization with the minimization of a non-linear error function that can be represented as a graph.

When the relations between features is a graph, bundle adjustment problem can be treated as a graph optimization problem.

iSAM

iSAM is an approach to SLAM based of fast incremental matrix factorization [39]. Since it is applicable to graph problems, it can also be used in graph optimization problems besides SLAM, such as bundle adjustment problems.

The idea of iSAM is to be used as a graph optimization solution for graph-based problems. It reuses the previously calculated components of the squared root factor of the squared root information matrix and performs only calculations over new entries of the matrix, which are affected directly by new measurements.

It is also very useful on cases of trajectory loops, where it is used periodic variable reordering of the squared root information matrix so to avoid fill-in of the matrix, which would result on a slowdown of the incremental factor of this type of optimization. Another feature that helps in this last case is use back-substitution as a way of recovering the current estimate.

Gauss-Newton optimization

Considering a minimization of an error function like the following [22]:

$$F(X) = \sum_{<i,j>\in C} e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}) \tag{2.29}$$

where X is a vector of parameters that represent the sensor poses, z_{ij} the mean and Ω_{ij} the information matrix of a constraint between the x_i and x_j computed poses, and $e(x_i, x_j, z_{ij})$ the vector error function that computes the difference between the expected observations and the real observation gathered by the robot, i.e. measures how well the poses satisfy the constraint z_{ij}.

To solve the minimization equation $X = argmin_x F(X)$, the Gauss-Newton optimization model can be applied to find the numerical solution of it. First step is to assume a good initial guess to the robot pose and then apply an approximation to the first order Taylor expansion of this guess. So considering $\breve{X}$ as the initial guess,

$$e_{ij}(\breve{X}_i + \Delta x_i, \breve{X}_j + \Delta X_j) = e_{ij}(\breve{X} + \Delta X) \tag{2.30}$$

$$\simeq e_{ij} J_{ij} \Delta X \tag{2.31}$$

where J_{ij} is the Jacobian of $e_{ij}(X)$ computed in $\breve{X}$ and e_{ij} is, by definition, equal to $e_{ij(X)}$. If we apply

the error F_{ij}, we get:

$$F_{ij}(\breve{X}) + \Delta X) = c_{ij} + 2b_{ij}\Delta X + \Delta X^T H_{ij} \Delta X, \tag{2.32}$$

where $c_{ij} \equiv e_{ij}^T \Omega_{ij} e_{ij}$,$b_{ij} \equiv e_{ij}^T \Omega_{ij} J_{ij}$ and $J_{ij} \equiv .J_{ij}^T \Omega_{ij} J_{ij}$. After a local approximation, where it is considered $c = \sum c_{ij}$, $b = \sum b_{ij}$ and $H = \sum H_{ij}$, we obtain,

$$F_{ij}(\breve{X}) + \Delta X) = c + 2b^T \Delta X + \Delta X^T H \Delta X, \tag{2.33}$$

which can be minimized in ΔX, result of a linear system of the of form $ax = b$,

$$H\Delta X^* = -b. \tag{2.34}$$

The resulting H matrix is the sparse information matrix of the system. The solution of the linear system can then be given by

$$X^* = \breve{X} + \Delta \breve{X}^*. \tag{2.35}$$

The Gauss-Newton optimization is then applied, iterating on Eq. 2.33, giving the solution in 2.34 and using 2.35 as an update step, where the previous solution is used as the initial guess of the robot pose.

2.2.6 Pose estimation

The pose estimation is the main core of the vision-based localization frameworks, and allows the estimation of both position and orientation of the vehicles, given the camera pose. For what visual pose estimation matters, there are numerous ways of getting the orientation of the visual stream, and those include evaluating the features of the image (or depth data, in case of having this source) and compute the pose of the vehicle relative to that data.

The estimated pose, allied to a proper selection of features and keypoints of the images, also allows to determine the motion of the system, if it is known the motion model of the vehicle given the data being processed.

For the SLAM algorithms being tested in this thesis, there are two main techniques being used for pose estimation: GICP [2] and RANSAC [18].

GICP vs RANSAC

GICP [2], or Generalized Iterative Closest Point, is a generalized model of ICP, which computes the differences, point-to-point, between two point clouds, by applying the a probabilistic model based on Gaussian distributions and using maximum-likelihood estimator to get the transformation between each point cloud. So, it is an algorithm that generally computes poses using the depth data only.

RANSAC [18], or Random Sample Consensus, can be applied to any kind of data, and it is based on the concept of picking a model sample and fitting it in the recovered data. That model fitting is applied in a way that the chosen subsection, after some iterations, minimizes a given cost function. So, it is a more robust algorithm that can compute poses using any kind of data, besides depth data as GICP.

2.2.7 Sensor fusion and model-estimation

The common type of sensor fusion for pose estimation are the Kalman Filters, which combine the information of different uncertain sources to obtain the values of variables of interest, which in the case of localization regard the pose and attitude of the vehicle, and the uncertainty related to these.

Other sensor fusion algorithms relay on the so called Complementary Filters, which turn to be a steady-state Kalman filters [25]. They are way simpler to implement, as they do not consider any statistical description of noise, but only the frequency which the signal arrives. They are way lighter in terms of processing power needed also, but when dealing with many sensor sensors, become extremely difficult to tweak, given that they have to deal with the rate and the accuracy of many sensor sources at the same time, which is not as optimal or generalized as the Kalman Filters.

Extended Kalman Filter

The EKF is a widely used Bayesian filtering tool for obtaining suboptimal solutions to nonlinear filtering problems [21]. EKF employs instantaneous linearization at each time step to approximate the nonlinearities, where the propagation of the error covariance matrix is assumed to be linear.

The most common form of structure of implies mixed continuous-discrete type of filter. Assuming x as the state of the system, u as the input of the system, y the output, z the sampled output, w the additive process noise, v additive measurement noise and the following generic representation of the system dynamics [21]:

$$\dot{x}(t) = f(x(t), u(t)) + Fw(t), \quad x(t_0) = x_0, \tag{2.36}$$

$$y(t) = g(x(t), u(t)), \tag{2.37}$$

$$z(k) = y(k) + Gv(k). \tag{2.38}$$

where,

- k represents the discrete sampling time when a measurement is available, considering that different sensors may update at different rates.
- f and g are differentiable functions.
- the matrices F and G determine the variance of the process and measurement noise which are assumed to be Gaussian.

The EKF algorithm includes a prediction and a correction step, where in the prediction step, the nonlinear process model is integrated forward from the last state estimate $\hat{x}(k-1)$ to yield the predicted state $\tilde{x}(k)$ until a measurement is available at time k, which means that is considered a transition from a continuous to a discrete time system.

The error covariance matrix P is also propagated to yield the predicted error covariance matrix $\tilde{P}$ by linearizing the process model using the Jacobian matrix $A = [\frac{\delta f}{\delta x}]_{x=\hat{x}(k)}$ around the instantaneous state estimate from the previous step.

This results in the approximation for the error covariance,

$$\dot{\tilde{P}} \approx AP + PA^T + Q, \tag{2.39}$$

which is also integrated forward from $P(k-1)$ to yield $\tilde{P}(k-1)$, having $Q = FF^T$.

The following step, which is the correction step, uses a received sensor measurement on a nonlinear measurement model $g(\tilde{x}, u)$. The predicted system output $\tilde{y}(k)$ is given by the predicted state estimate $\tilde{x}(k)$ and the measurement model, which is computed by the Jacobian matrix $C(k) = [\frac{\delta g}{\delta x}]_{x=\hat{x}(k)}$ around the predicted state $\tilde{x}(k)$, resulting on the correction equations [24]

$$K(k) = \tilde{P}C^T(k)[C(k)\tilde{P}(k)C^T(k) + R]^{-1}, \tag{2.40}$$

$$\hat{x}(k) = \tilde{x}(k) + K(k)[z(k) - \tilde{y}(k)], \tag{2.41}$$

$$P(k) = [I - K(k)C(k)]\tilde{P}(k), \tag{2.42}$$

where $R = GG^T$ and G is such that R is positive definite.

Complementary filter

Complementary filters are capable of managing both high-pass and low-pass filters simultaneously. The low pass filters high frequency signals (such as the accelerometer in the case of vibration) and high pass filters low frequency signals (such as the drift of the gyroscope). By combining these filters, one can get a good signal, without the general more complicated implementation of the Kalman filter.

The complementary filters can have different orders of magnitude, depending on the order of the transfer functions applied at the filters. For example, a first-order complementary filter can be built the following way: considering x and y the noisy measurements of a sensor source and $\hat{z}$ the estimated signal produced by the filter. If the noise in y is mostly from high-frequency and x from low-frequency, then a filter $G(s)$ can be applied to y and it works as a low-pass filter, while its complement $[1 - G(s)]$ is applied to x as a high-pass filter (Fig. 2.12) [25].

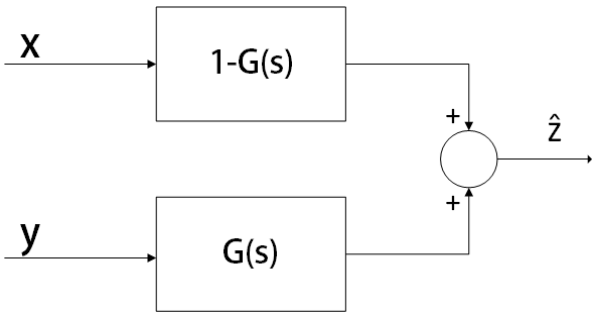


Figure 2.12: A simple complementary filter

The same filter can be applied in another way, where the filter only is applied to noise. If $x = z + n_1$ and $x = z + n_2$, where n is the noise, the input of $G(s)$ will be $n_2 - n_1$ (Fig. 2.13).

The common inertial navigation systems have accelerations measurements combined with position

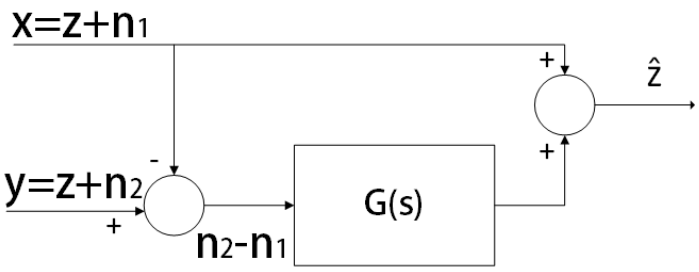
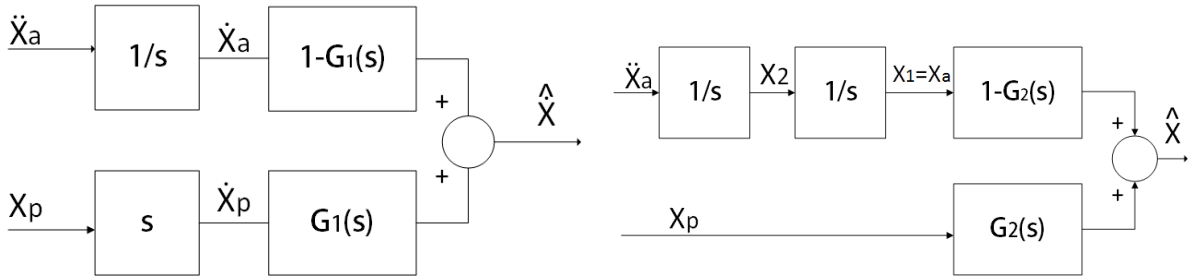


Figure 2.13: Complementary filter applied to noise

measurements. For aircraft systems, the most common position source is the GPS, which is combined with the accelerometer measurements in the filter, which allow the estimation of both position and velocity. An example of this is the following filter (Fig. 2.14).



(a) Filter for velocity estimation (b) Filter for position estimation

Figure 2.14: Inertial navigation complementary filter

Fig. 2.14 (a) represents the filter needed estimate velocity. $\ddot{x}_a$ is the linear acceleration measurement coming from the accelerometer sensor while x_p is the GPS position measurement. The integration $\frac{1}{s}$ attenuates the high-frequency noise in the acceleration measurement while s is a gain for the position measurement. As it is the velocity that is pretended to be estimated, it is required a second-order filter so that the transfer function allows attenuation at the high frequencies but also have a unity gain in low frequencies. The result is a second-order filter:

$$G_1(s) = \frac{b}{s^2 + as + b}, \tag{2.43}$$

$$1 - G_1(s) = \frac{s^2 + as}{s^2 + as + b}, \tag{2.44}$$

where parameters a and b define natural frequency and damping factor for the filter. In the case of the position estimate - Fig. 2.14 (b) - , it is applied double integration to the acceleration measurement for obtaining a position and the GPS has unity gain. Another second-order transfer function is required:

$$G_2(s) = \frac{as + b}{s^2 + as + b}, \tag{2.45}$$

$$1 - G_2(s) = \frac{s^2}{s^2 + as + b}, \tag{2.46}$$

The combination of the previous filters results in a second-order complementary filter as follows (Fig. 2.15):

$$\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} (x_p - \hat{x}_1) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ddot{x}_a \tag{2.47}$$

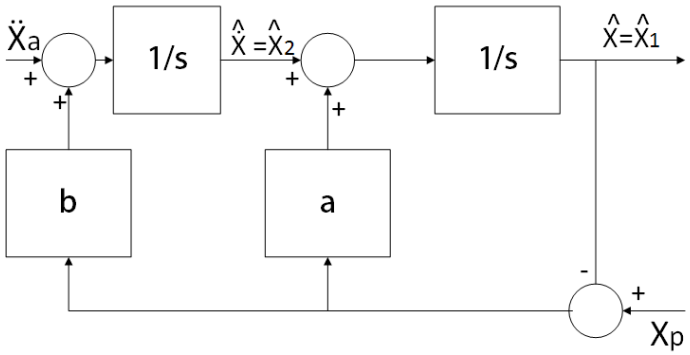


Figure 2.15: Second-Order Complementary Filter for Inertial Navigation

The position estimate is differenced with the position measurement to produce an error signal which is fed back to produce corrections in the estimates [25].

2.3 Mapping

The following sections cover an explanation of the importance of mapping in the navigation tasks, the different ways of storing a map of the environment given as source of data a point cloud, and it is presented the well known and widely used state-of-the-art method for 3D mapping: Octomap.

2.3.1 Introduction

Many robotics applications require a three-dimensional map of the environment that surrounds, which is created from the data recorded by sensors, but create and maintain this amount of data in an efficient manner is not a trivial operation though.

A good method of management of the map should ensure the ability to model free space, occupied and not yet explored. It also should be fast and simple when creating and updating. For example, a second storage technique could be more or less complex in processing the input data.

Also, there should be a representation of probabilistic information, because the data to create the map typically comes from 3D sensors of various kinds united by the presence of more or less important measurement noise and, therefore, false data. A probabilistic approach to the management of the map allows you to merge many uncertain steps to obtain a degree, in other words, a point referred to as busy or free, using a probabilistic estimate on the latest measurements.

Efficiency on mapping methods is also required, both in terms of access time to obtain information on a single point, but also in terms of memory occupied to represent it. For that, the ability to get multi-

resolution can be handy to have a map that can display space for different resolutions according to the needs of the moment and the efficiency required.

The following sections will explain the source of data being used and, from the different structures of data that can be used to create and maintain a map, why is the Octomap [1] approach one most used and how does it work.

2.3.2 Source of data

The input data can come from a variety of sources, like lasers, sonars, cameras, etc. For this particular thesis, it is taken into account only sensors that generate a 3D point cloud. These represent the portion of the world observed by an array of points where each is characterized by a numerical value that indicates the distance from the sensor to the occupied space. The position in space is then given by this value and the position within the occupation matrix.

A fork of this concept is RGB-D, where each pixel is associated with four numerical values, which are the three colours (RGB) and depth (D) for the points distance from the observer, which can give the distance to the surroundings of the robot where the RGB-D camera is installed.

In both cases, what is obtained is a representation of the observed objects and surroundings by means of thousands of points positioned in space, where in the case RGB-D, each point is also associated with a colour. These representations are considered the standard for reconstruction applications of 3D environments since they allow to combine, in a simple and computationally efficient way, information regarding the appearance of the observed scene and its three-dimensional structure.

The resolution of the input data is often greater and more than needed compared to the typical needs of a robotic system, so it is therefore a common practice to apply a discretization of a point cloud by the use of voxels: cubes of fixed size associated to a single point, with a numerical value representation of the portion of space that is being measured. Intuitively, the cloud of points in the input is divided into equivalent volumes, each of them associated with a centre and a value (busy or free) based on the metric chosen to filter the input points.

2.3.3 Map creation and data structures

The data from RGB-D sensors are a sequence of point clouds related to each other only by a temporal order, like a three-dimensional video of what the sensor has observed, and do not create a three-dimensional model of the observed object without further processing.

This is because point clouds lacks information regarding where was the sensor within the space while acquired that image. The knowledge obtained of three-dimensional environment is then limited to the position of the objects observed compared to the reference system that has, as its origin, the sensor frame. Without absolute information with respect to a fixed reference frame, the system is not able to determine the relative position of the various point clouds in space and, thus, obtain a map.

Therefore, as it was explained in the previous section, a localization method is need, so to keep track of the movement in order to position each cloud in the place in which it was acquired. Knowing how

the sensor is moved in space is indeed necessary to correctly position the point cloud in an absolute reference system and, therefore, obtain a consistent representation of the observed world.

The map creation is an important step, but storing this map and get is representation is evenly important. How to keep the map in memory so that it is efficient to use and takes up very little resources is the main problem that arises, and for which some solutions have already been researched.

One of them is the Point Cloud (Fig 2.16)[6] itself, where there is no application of any type of processing or discretization and the data is saved directly as it comes from the sensors. That data is a series of coordinates (X, Y, Z) which indicate occupied points in space.

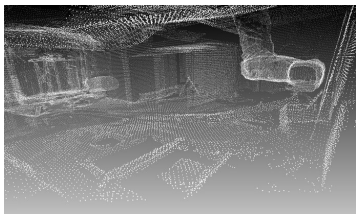


Figure 2.16: Raw point cloud

From a computational point of view this is the most efficient method because there is no transformation or processing to be performed, and then the data is saved as is. The problem with this method is that there is no explicit representation of the free or not explored space and it also does not give a way of handling the noise inevitably present in the unfiltered data. Also it is memory expensive, as the modern RGB-D sensors are able to generate clouds of the order of 10,000 points per frame working at frequencies of 30 Hz. So directly store these data without any type of discretization is neither useful nor efficient.

Another approach is the use of voxel grids (Fig 2.17)[7], where it is applied a discretization of the space through the use of voxels - cubes of a certain volume.

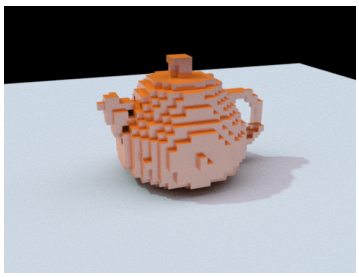


Figure 2.17: A voxel grid generated from an object observation

The environment is represented by a 3D grid of cubes, each representing the entire space inside by a single value: (busy / free / not explored). This method improves efficiency in the consumption of memory, but not the speed of query, since there is still no well-defined hierarchical structure that binds the blocks between them. The grid size also must be known in advance to be able to divide the points in the voxels.

[6] UndergroundCity3D. 3d point cloud. `http://www.undergroundcity3d.com/uvod/forum/3d-point-cloud`. Online.
[7] Creative Crash. Maya - voxel grid generator. `http://www.creativecrash.com/maya/script/voxel-grid-generator`. Online.

There is also the elevation map (Figure 2.18) [1], which uses a 2D grid to model space. In this method, for each cell of the grid, it is saved the maximum elevation reached by the obstacles in its interior and that is why it is often referred to as ”2.5D grid”.

Figure 2.18: Elevation map of an observed tree

This representation is considered to be optimal in terms of storage efficiency, but has the great defect of not giving a volumetric representation of space, but only a discretization on a plane. Can, though, be considered an ideal representation method for robots that just have to navigate on one floor.

An evolution of the previous method is the multi-level elevation grid (Figure 2.19) [1], where it also exploits a 2D grid but for each cell it is saved all occupied voxels that are in the portion of space.

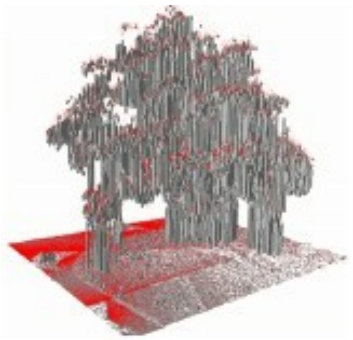


Figure 2.19: Elevation grid of an observed tree

In this method, even though there is a volume rendering, it is not possible to distinguish between free space and occupied/not explored space (although it would be possible to work around this by saving even a list of free voxel for each cell). Also, each map update requires computationally heavy processes since there is no kind of explicit hierarchy between voxels.

All the previous methods are possible representation methods, but have their issues on identification of free space and in the computational requirements for running them because there is no hierarchy between the elements that represent the volume of space. But there is a method that is considered the state-of-the-art most optimal method for 3D mapping representation, which is Octomap [1].

This method, Octomap, recursively decomposes the space into cubes using what is called an octree. This last one gives the possibility of dividing the observable environment in eight volumes, typically voxels of the same size. This allows a discretised representation of an environment, on each occupied space is represented with cubes. This cubes can have a colour associated with the height they occupy

in the local reference frame (Fig. 2.20)[8]. This technique allows dividing the space up to the desired

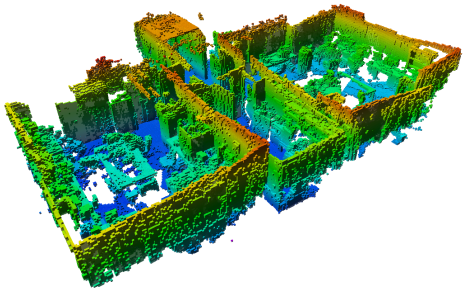


Figure 2.20: Octmap representation of a building

accuracy, where at each step of the recursion it is created smaller voxels to represent the space. The recursion stops when the size volume reaches the minimum size established a priori. The discretized world can then be represented by a tree structure where each node has eight children and represents a portion of space with a well determined resolution, given by the dimension of the cube side (Fig 2.21) [1]. This approach is optimal since it allows to leave parts of the map at different resolutions, or even not

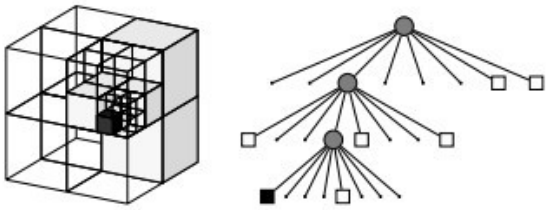


Figure 2.21: Octree decomposition in multiple cubes

initialized, without affecting the overall quality of the representation, since it is a hierarchical structure.

The tree structure allows minimizing the number of information saved for each node (the position in space is obtainable, starting from the root node), to speed up the query of the map and to save information in case that particular volume is free, busy or unexplored. This information is given by a probabilistic model, which gives the probability of each leaf node of the tree. The probability of occupation $P(n|z_{1:t})$ given a sensor measurement $z_{1:t}$ is given by

$$P(n|z_{1:t}) = [1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t} - 1)} \frac{P(n)}{1 - P(n)}]^{-1}, \tag{2.48}$$

where z_t is the current sensor measurement, $P(n)$ is the a priori probability, $P(n|z_t)$ the probability of voxel n being occupied given measurement z_t and $P(n|z_{1:t} - 1)$ the previous estimate. Since it is assumed that the probability $P(n)$ is 0.5, i.e. a voxel can be occupied or not with the same probability, and using the log-odds notation, the previous math can be simplified to

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t), \tag{2.49}$$

[8]ROS Wiki. CCNY RGBD Tools. `http://wiki.ros.org/ccny_rgbd_tools`. Online.

with

$$L(n) = log[\frac{P(n)}{1 - P(n)}]. \tag{2.50}$$

This last equation needs to be modified when we consider the 3D map to be used in navigation: a threshold can be used to establish if a voxel is occupied or not, which requires adaptability of the algorithm to situations where the environment itself is not static. And for that, an upper and lower bounds of occupancy estimates can be considered, which leads to

$$L(n|z_{1:t}) = max(min(L(n|z_{1:t-1}) + L(n|z_t), l_{max}), l_{min}) \tag{2.51}$$

where l_{max} and l_{min} are the upper and lower bound respectively.

Furthermore, depending on the depth to which to trim the tree, it is also possible to obtain maps at different resolutions. That multi-resolution can be obtained by aggregation of the probabilities of the inner nodes children. This can be given by an average occupancy

$$\bar{l}(n) = \frac{1}{8}\sum_{i=1}^{8} L(n_i), \tag{2.52}$$

or the maximum occupancy

$$\hat{l}(n) = max_i L(n_i), \tag{2.53}$$

where the last one can be a suitable strategy to robot navigation since it is assumed that every part of a volume that was measured as occupied leads to a volume be considered as occupied, which is useful on cases where we want to plan paths in an environment with many obstacles.

So we can see that the update at each new input is always performed at the level of the leaf nodes by means of a probabilistic function, which allows to have a robust implementation even in the presence of noisy data coming from the sensors. From the old value of employment associated with the node and the new one coming from the sensors by applying a function defined a priori, the new value is obtained. That way, the node is considered busy if the result of this operation exceeds a certain threshold. Otherwise it is considered free. If the node has not yet been initialized, then it is considered unexplored.

2.4 Navigation and Path Planning

The next sections make an approach to the navigation issue. It is stated a difference between the lower control loops, which include the PID control, and the higher control loops, which include a description of several types of path planning solutions for the navigation problem.

2.4.1 Introduction

Navigation can be described as the process of determining a suitable and safe path between a starting and a goal point for a robot travelling between them. This process is aided by the localization process,

which localizes the robot in an environment, and by motion planning techniques, which guide the robot through the space between the starting point and the goal point.

The performance of a good navigation framework depends of an accurate robot localization in the environment but also of an efficient path planning method that can guide the robot.

On indoor navigation systems, one can find three main groups [16]: map-based navigation systems, map-building-based navigation systems and mapless navigation systems.

The first one includes systems that have pre-processed maps of the environment. In visual-based navigation systems, this mean that the localization problem can be solved using landmarks, natural or artificial, and therefore, a path planning algorithm can be applied so that the robot can navigate.

The map-building-based, which is the type that is being considered for this work, regard systems that are able to construct their map while they navigate, which means that an exploration behaviour can be used so that the robot can plan its trajectories.

The last type, which is the mapless navigation system, do not need to know a map to navigate, because the localization and movement depends on the observation of specific elements of the environment, like the floor, walls, markers, and others. Optical flow only based navigation systems are systems of this type, cause estimates the motion of the robot based on the motion of an object in a sequence of images.

Also, there must be considered two layers of control abstraction: one lower layer, which directly controls the actuators for motion of the robots. This layer includes the PID controllers for motor control and feedback. Also, it is needed one higher layer, which takes into account higher processing loops for position estimation, mapping, user-machine interface and path planning.

If the system has a map of the surrounding environment, it can use it to plan the best route free of obstacles to reach a goal. To exploit the map and generate a path, the system must necessarily know other pieces of information: a geometric description of the space occupied by the robot, in order to correctly verify any collisions, the starting position within the map and the model description of the robot, which includes the kinematic and dynamic models, physical model and motion constraints.

2.4.2 Lower-control loops - Position and Attitude Control

The requirements of an autonomous system include some kind of control loop that must be implemented so direct control of the system behaviour can be achieved. As path planning solutions are higher-control loops that just send control commands as a vector of coordinates, being those position, velocity, acceleration or even attitude, in the system itself there must exist a lower control loop that controls the velocity of the motors directly.

For that, a simple PID controller can be used so to control the attitude of the MAV by having as input the goal state plus the feedback of the loop. In the case of a controlling system that has as inputs the desired position, the PID controller must have two reset feedbacks, one that is internal and only depends on the feedback of the direct control of the attitude of the vehicle, and an external one that is used to correct position.

But for a correct implementation of a control loop that can actually assume both attitude and position control, it must be understood how one controller and the other works. The following sections will discretize the difference between the two and how both can be implemented.

Attitude controller

An attitude controller stabilizes the orientation of the aircraft in the 3D space, which means it tries to push the movement around each of the 3 axis, X, Y and Z towards an input setpoint, which can be given by a remote control, or in case of autonomous behaviour, an external high-control loop based on a planned path.

Since the attitude control interacts directly with the angular and thrust variables of the MAV, which have direct impact on the angular velocity of each of the motors, it can be considered an inner-reset PID control loop, where the feedback is given by the current angular and thrust values of the quadrotor. The control loop can be checked in figure 2.22, where the f index refers to the values of feedback and r is the reference values, i.e. the desired values.

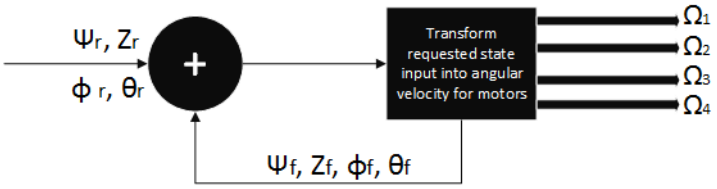


Figure 2.22: Attitude controller

The attitude controller implements four SISO - single input, single output - control loops, one four each variable (roll, pitch, yaw and thrust), where each one is composed by a simple PID controller (Fig 2.23):

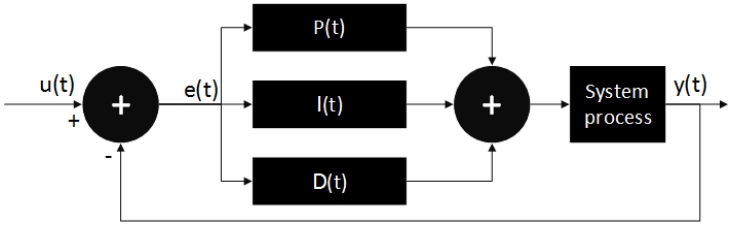


Figure 2.23: PID controller

with,

$$P(t) = K_p e(t), \tag{2.54}$$

$$I(t) = K_i \int_0^t e(\tau)\mathrm{d}\tau, \tag{2.55}$$

$$D(t) = K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t), \tag{2.56}$$

which results in,

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t), \tag{2.57}$$

where K_p is the proportional gain, K_i the integral gain and K_d the derivative gain.

Position controller

A position controller aims at making the aircraft fly towards a specific point in the world. In the case of a quadrotor, the movement of the vehicle along the axis is controlled by adjusting the thrust force vector of the vehicle. The thrust vector always points along the negative Z direction of the body frame, as upwards in a normal hovering position of the vehicle. When tilting the quadrotor around the roll or pitch axis, the orientation of the thrust vector can be changed. Therefore, to control position (via the force vector) the attitude also has to be controlled.

The same thing can be done when, instead of feeding a force vector, position or velocity vectors are used, which can be transcribed by a mathematical transformation using the third Newton's law. So to get an acceleration, and then the force, position has to be double derivate and velocity derivate once.

The fact that position is controlled by changing the attitude leads to a control setup where, as already described, position is controlled in an outer loop and attitude in an inner loop.

That results on the following control loop (Fig 2.23):

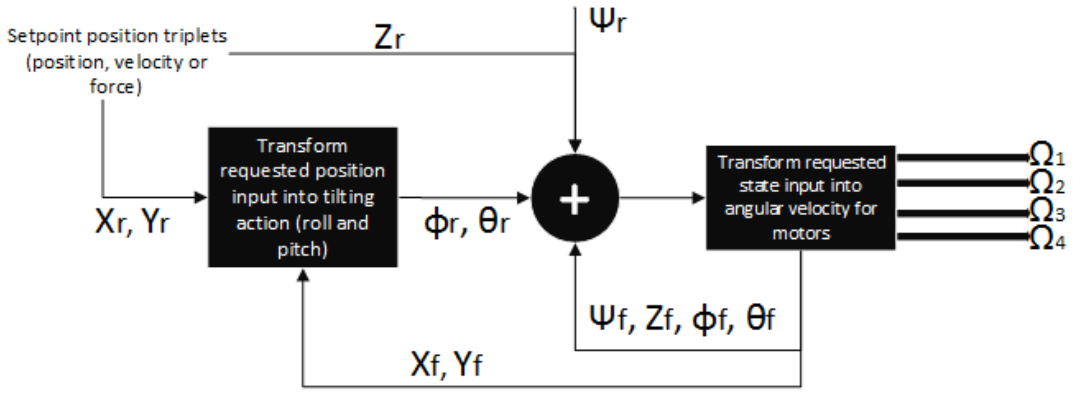


Figure 2.24: Full loop control scheme

2.4.3 Higher-control loops - 3D motion planning solutions

The trajectories generated by the algorithms of motion planning may or may not take into account the kinematic limits of the robot itself, which usually result in path solutions that are purely geometrical. Those cases usually require a step of refinement to determine the best trajectory, and have as a result a series of commands sent to the robot so to forward it to the goal - that is the case of geometric path planning. The other case, which is the control-oriented planning, already take into account the dynamics of the robot and so produce commands that are considered feasible to reach the goal.

So that the robot is controlled in a way that it avoids colliding into objects and obstacles that limit the navigable world, all the planning area must be converted into a mathematical model that results on geometric shapes that define the world. Then, if one has a geometric map with a set of all points that make part of that space, one can consider two complementary subsets: the points where the obstacles

are and the points of free space. But, for a proper planning, it must also be considered a set of locations and robot configurations that may be realized on the free space. So, given all those elements, the starting point and the goal point, the idea of both geometric path planning and control-oriented planning is to calculate an near optimal geometric connection between map points that have in consideration the given constraints of the robot and of the planning scene.

Geometric Path Planning

This kind of solutions only takes into account the trajectory from start to goal, and does not care about any information regarding commands, configuration constraints or other given to the robot. The actual control is simple: it assumes a description of the world based on geometric relations like graphs or trees and calculates a near optimal path between the given starting and goal points. That calculation is a series of iterations that try to refine the trajectory with each update on robot localization or map changes.

The following algorithms are examples of proved solutions to path planning based only on scene geometry:

- **RRT** [31], or Rapidly-Exploring Random Trees, implements an incremental construction of a search path based on search trees that explore a given state space uniformly, where unexplored space is sampled as points and represented as the leafs on the tree, which dynamically expands until it reaches the goal node. Then, it refines the path given the constructed trees. The search is done considering both algebraic constraints (obstacles) and differential constraints (dynamic of the model). Figure 2.25 [31] shows an example of a tree being constructed along free space.

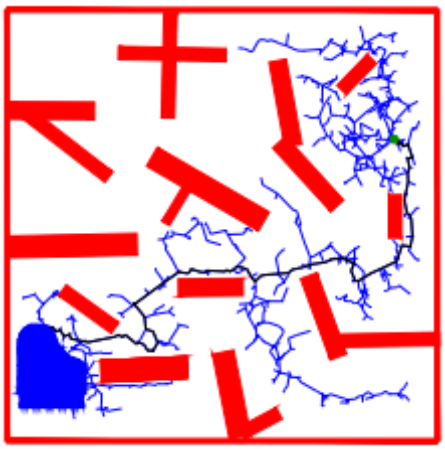


Figure 2.25: RRT tree representation

- **PRM** [27], or Probabilistic Roadmap, is a method that uses graphs to search for an optimal path. The idea is simple: first it constructs a roadmap based on a probabilistic model and stores it in a graph where the nodes are collision-free configurations and the edges are the probable feasible paths between each node; then, knowing the starting and goal nodes, it calculates the path that

joins those two nodes. Figure 2.26 [27] shows an example of a graph constructed along free space.

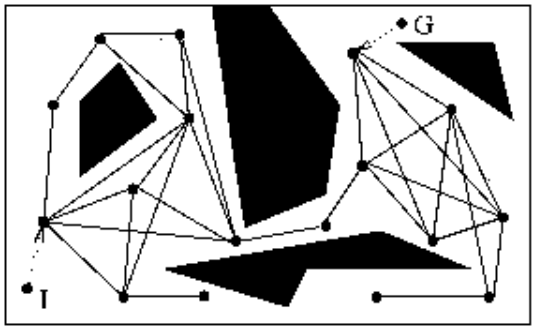


Figure 2.26: PRM graph representation

Control-oriented planning

This kind of solutions, more than the geometric approaches, also take into account kinematic and dynamic constraints of the robot. So, besides the path planning, it is issued a control planning based on the controls required to implement a certain trajectory, given the constraints presented. But, this kind of solutions also require a higher computational process, since it is also necessary to model the dynamic characteristics of the system.

A plausible solution that uses this kind of approach is the **KPIECE** [9] algorithm, which stands for Kinodynamic Planning by Interior-Exterior Cell Exploration.

The algorithm itself was designed to handle complex dynamic models, and its action is based on the construction of a tree of motions in the state space of the robot, where it discretizes the inputs with a minimum unit of time and obtains a tree with the root in the starting configuration, having the branches saving the commands needed to move from one node to the next. This procedure is then repeated for each new node until reaching the configuration of the goal.

All the planning done with KPIECE is done on a state space structured by a multi-level and multi-resolution grid cell that keeps track of the space already explored, where the focus is not on the geometry, but in the internal states of the grid (Figure 2.27) [9].

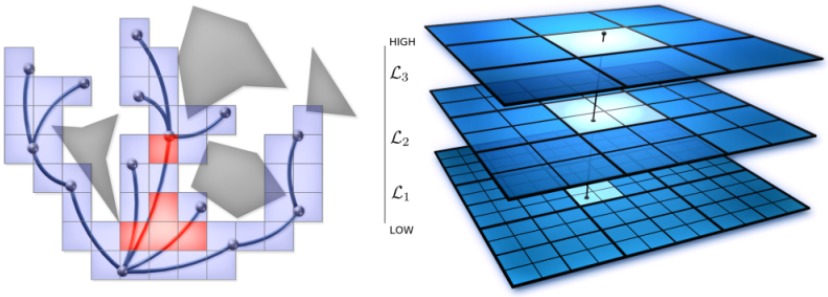


Figure 2.27: KPIECE multi-level grid

Chapter 3

Implementation and Project Setup

The next subchapters will cover the project setup, which include an overview of the several hardware and software components that compose the used system architecture. It will start with a diagram of the architecture, so to present an overall view of the system architecture, and then explanation of the several modules and its relations, both in software and in the hardware parts.

3.1 Project setup overview

The system architecture chosen is the result of a careful selection of components and an integration that had to take into account different architectures, both hardware and software, to be implemented in the best conditions and in order to be tested. The principal aim of the project is preparing a platform that integrates hardware and open-source software components that can be used as a system capable of performing autonomous flight indoors. Priority was given to how communication between robot and computer, i.e., between the off-board computer and the flight controller unit is done, and to the study of modules that allow the estimation of the position and orientation of the MAV and the mapping of space around the vehicle. These are essential for the work of the existing motion planning framework in the off-board computer so it can successfully plan the motion of the MAV in confined spaces.

Figure 3.1 is a architecture diagram that summarily represents an overview of the different components that compose the system. Note that this diagram is a very simple representation of the working system, as a most complete and deep review of the different processes and different parts of each hardware and software architectures will require more than one page to be represented.

The system architecture can be divided in two main parts: the off-board computer system hardware and software part, and the flight controller unit software implementation, inner hardware and the rest of the quadrotor MAV default and added hardware part.

The off-board computer is a higher computational unit which regard the higher computational processes of a navigation system. It runs a Linux operating system with a running ROS[1], Robot Operating System, which is used as the main framework to integrate all the different software running on the

[1] ROS. Robot operating system. `http://www.ros.org/`. Online.

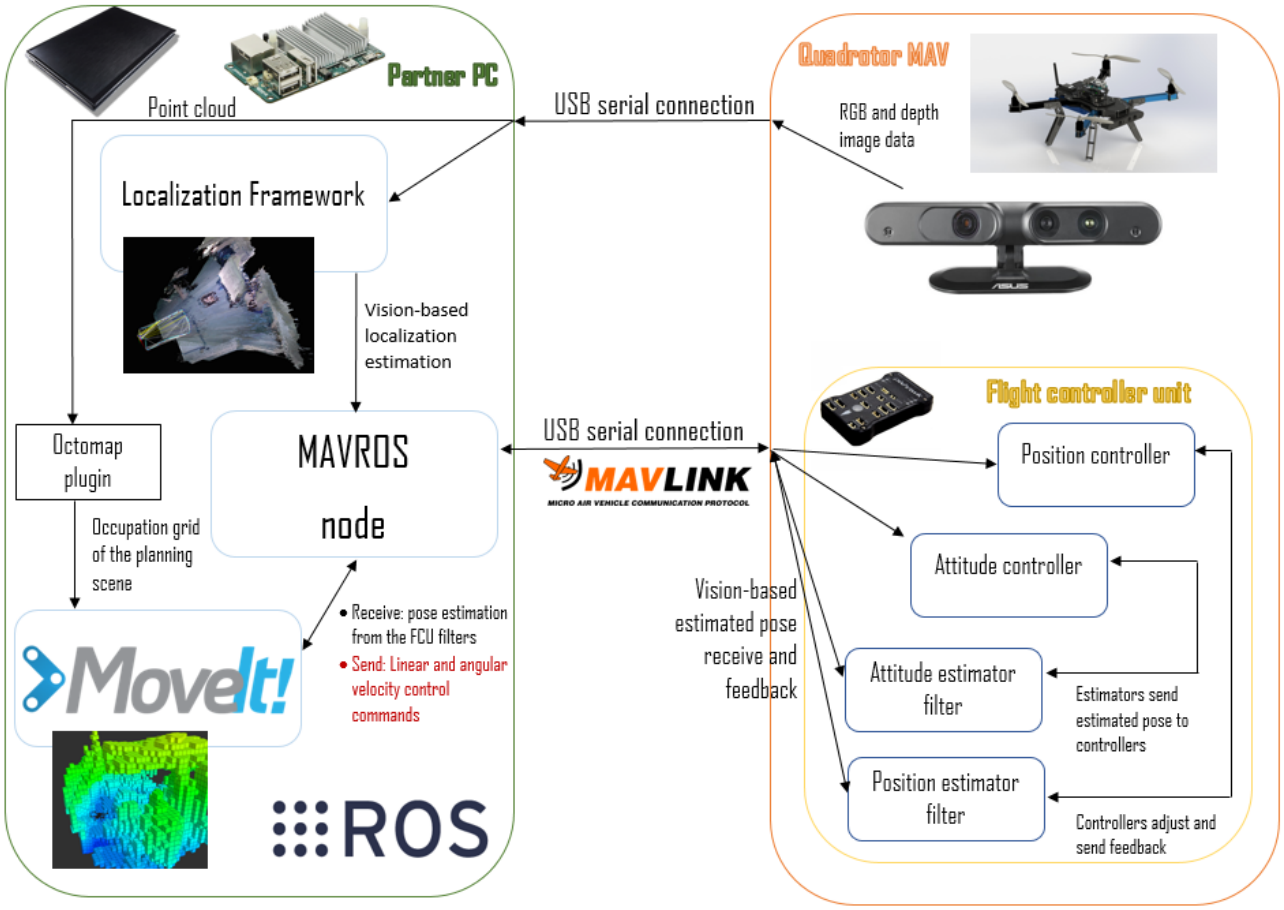


Figure 3.1: Project system architecture

off-board computer, and includes:

- a module to estimate the localization of the MAV;
- a module for representing the surrounding scene map and to properly operate the navigation tasks of the MAV;
- a module that allows the communication between the off-board computer and the FCU on-board the MAV.

The second part includes the used hardware and software on the MAV. A note to the fact that off-board computer, in the case of a lower footprint processing system, can be put on-board the MAV, but for consideration and better understanding of the architecture, will be considered off-board the MAV. The quadrotor MAV includes the supporting frame, the motors, the ESC's, the radio telemetry module, the battery and the flight control unit as the default components that compose the vehicle. Besides that, it was added a RGB-D camera for depth sensing and RGB image data retrieval, and an Optical Flow module with sonar, so to estimate linear horizontal velocities and distance to the ground plane.

The main component running on the MAV, which controls the behaviour of the quadrotor, is the FCU. This piece of hardware is responsible for position and attitude control of the MAV, besides adding some more layers of software that allow many kind of enhancements of the flight experience. It also includes

filtering for the estimation of position and attitude of the quadrotor, which are essential for the correct functioning of the system.

The different capabilities of each component of the system allows an integration between them which leads to more complex applications of the flying system besides manual radio control from the user or the autonomous flight in outdoor areas, tasks for which the default FCU implementation comes already optimized, or at least, ready to use and tweak. So that the default system is able to be used in autonomous flights in indoor environments, some adjustments have to be done in order that the on-board system could receive external estimates of its position and orientation from vision-based localization modules running on off-board computers. Also, for future navigation capabilities, the system was prepared to receive off-board controlling commands to be received by the position and attitude controllers in order to autonomously navigate with the quadrotor.

So, in a simple way, the overall system behaviour can be described in the following way:

- ROS supports the Localization and the Motion Planning modules. It also supports a communication module that codes and decodes messages between ROS and the communication protocol being used on the serial connection between the off-board computer and the FCU;
- The localization module receives depth and RGB data from the MAV on-board RGB-D camera, processes that data in feature extraction, visual odometry and keyframe mapping algorithms and provides the estimated pose and motion of the quadrotor in certain rates;
- The motion planning module receives the processed point cloud coming from the OpenNI2 wrapper running on ROS, which is the driver that allows the data conversion to formats that are readable by the ROS applications, and converts it to an occupation grid that maps the surrounding scene using a plugin. It also allows, using other software modules, to plan the motion of the MAV and send control commands to it.
- The estimated pose is sent to the decoder module on ROS, which transforms the data into a usable message to be read by the FCU. The same thing happens with controlling commands coming from the motion planning module;
- The estimated pose being sent in the serial data stream is received and decoded on the FCU and forwarded to the position and attitude estimator filters to be fused in those and to retrieve a filtered position and attitude to be used internally by the position controllers or to be sent as a feedback to the motion planning module.
- The position and attitude controllers have different command sources, which are used depending on the controlling mode being used at the moment by the FCU. If the FCU is in a position control mode, which is an inner control loop of the FCU, the controller ignores external commands (which as the estimation messages, are also decoded on the FCU after being received in the serial stream) and only uses inner loop that tries to hold the current position and attitude of the vehicle. If the control mode is set to off-board control, all inner control commands are ignored and the external commands are assumed. It is important to notice that the actual position and attitude is fed

into the controllers so they can act depending on the current state of the vehicle. More information about the flight modes and the navigation state machine running on the FCU can be consulted in [2].

It should be noted that in the presented architedture diagram is that the part referent to sending the linear and angular velocity commands from the planner to the FCU is with font coloured red. The reason is very simple: the implemented controller loop on the motion framework part is the same as the used in a previous work, which is going to be referenced in 3.4.2. The controller loop worked for the previous work in a simulation environment. Also, the controller messages are already received by the FCU. That means it can already be controlled autonomously by the motion planner. But the response from the quadrotor motors to velocity increase has ”peaks” and is unstable, after verified in some tests with the propellers out of the MAV. So this part was left in stand-by and it will be referenced as feature work, so to test and tweak the velocity control in the FCU side.

As to understand how this interactions between the different system parts occur and how the frameworks work, the following sections regard the used hardware and the used software architectures, covering the specs and the operating mode of each one.

3.2 Vehicle Setup

This section covers the used hardware and its specs. Notice that the off-board computers, in this case, are considered as belonging to the vehicle configuration, since the main goal is to work towards a total autonomous system.

3.2.1 Quadrotor Specs

The used vehicle is a 3DR RTF X4 quadrotor[3] (Figure 3.2), which features a Pixhawk Autopilot System[4], an advanced flight controller unit with a full range of autonomous flight modes, including waypoint navigation, loiter, circle, and return to launch, geofencing and robust failsafe to ensure the safe operations and built-in advanced processor and sensor technology from ST Microelectronics and a NuttX real-time operating system.

It uses a 3DR Power Module with XT60 connector, with a max input voltage of 18V, having a maximum current sensing of 60A with a Pixhawk.

It also has four 880 Kv brushless motors - Kv as the number of revolutions per minute that the motor turns when one Volt is applied with no load attached to the motor - and four 10x4.7 slow-fly APC propellers (two clock-wise and two counter-clock-wise), a 4S 5800 mAh battery pack and a 433Mhz 3DR Radio Telemetry system.

[2] PX4 Firmware Wiki. Flight modes. `http://pixhawk.org/users/system_modes`. Online.
[3] Autonomous Avionics Store. 3D Robotics RTF X4 Quadcopter. `http://autonomousavionics.com/products/3d-robotics-x4-quadcopter`. Online.
[4] Pixhawk. Autopilot system. `http://pixhawk.org/`. Online.

In addiction to the default vehicle hardware, it was added a customized propeller protection, which was designed in SolidWorks and 3D printed with the purpose of protecting, not just the vehicle, but the surroundings in case of control mistakes or failures.

Figure 3.2: Customized quadorotor used on this project

3.2.2 RGB-D Camera

The used vision system is composed by a RGB-D Camera and an Optical Flow module.

The first one is an Asus Xtion Pro Live[5] (Figure 3.3). It is an RGB-D type camera which offers a more convenient, flexible, lighter and smaller solution than the most common Microsoft XBox Kinect. Even the new Kinect For Windows, or Kinect V2, does not offer a good solution compared to the standard technology of Primesense (which is embedded in the camera), since the use of TOF, even though it is not sensitive to IR light, has smaller range compared to the Primesense sensors.

Figure 3.3: Asus Xtion Pro Live

Also the Asus camera models have the distinct advantage in the fact that they are USB-only powered, are smaller and easier to mount than the Kinect. The only disadvantage is that there is a need to pay attention to how you support the camera when mounting it on a vehicle, since the mounting tab on the Asus will break in high-stress or high-vibration environments if it is not properly supported, even though it is still easier than mounting a Kinect.

The other characteristics that are needed to take into account in this comparison are the RGB image quality, which is way better on the Asus camera, and the possibility on this one to double the frame rate

[5]ASUSTek Computer Inc. ASUS Multimedia. Asus xtion pro live. `http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO_LIVE`. Online.

by decreasing the depth image quality.

The specifications of the Asus camera are the following:

- A power consumption of approximately 2.5 W.
- A usage distance between 0.8 m and 3.5 m, which can be increased, but will mean a reduction of the accuracy of the farthest points being detected and processed.
- It has a RGB and a Depth sensor combined with a pair stereo microphones, which are supported by the OpenNI driver but not yet supported by the OpenNI wrapper package in the ROS framework.
- As said, there is the possibility of adjusting the quality and the framerate of the depth sensor; the two options are the VGA (640x480) at 30 fps and the QVGA (320x240) at 60 fps.
- Also in the RGB sensor, the image quality is adjustable for SXGA (1280*1024) or VGA only (640x480).
- The essential feature for MAV applications is its size and weight: 18 x 3.5 x 5 cm and 230 g, respectively.

A thing of notice in this kind of sensors, which turn out to be a problem if used for robotic navigation cases, are the proximity dead zone, i.e. this sensors just have a usage distance after 80 cm, which means that for obstacle avoidance applications, it will require the use of extra range sensors like IR or ultrasonic range finders. Another way would be the use of the discontinued Primesense Carmine 1.09 cameras, which have an operation range between the 35 cm and 1.4 m, which will mean a reduce of maximum range but a better proximity range. Still, this approach cannot be consider as optimal cause there is still a dead zone of 35 cm which has to be covered and it will mean a reduction on the sensing and mapping processing, resulting in a slower navigation processing.

Also to refer that despite the fact that Apple has acquired Primesense company and that this last one stopped manufacturing their sensors and cameras, the company developers stated that they will continue to support the OpenNI drivers, since there is still a huge community using their cameras. The most probable thing to happen is the community advance to the use of sensors like the new Structure Sensor, which is an Occipital product created by the ex-Primesense developers.

3.2.3 Optical Flow and Sonar

The optical flow kit is a commercial PX4Flow kit[6] (Figure 3.4), which is an optical flow smart camera with a native resolution of 752x480 pixels and calculates optical flow on a 4x binned and cropped area at 400 Hz, due to its superior light sensitivity with 24x2 μm super-pixels, using a 168 MHz Cortex M4F ARM CPU.

It works indoors and in low outdoor light conditions without the need for an illumination LED and it can be reprogrammed to do any other basic low-level computer vision task.

[6] PX4Flow. Kit. `http://pixhawk.org/modules/px4flow`. Online.

(a) PX4Flow top

(b) PX4Flow bottom

Figure 3.4: PX4Flow kit

Due to its on-board 16 bit gyroscope up to 2000°/s and 780 Hz update rate, with default high precision-mode at 500°/s, and the correction algorithms inside, there is no need to external processing of correction filters for the estimated velocities, which means that the sensor can be placed anywhere in the vehicle, with the right orientation off course, which is camera facing downwards and the camera X-axes pointing forward.

The kit also includes a Ultrasonic Range Finder, a Maxbotix HRLV-MaxSonar-EZ4[7] (Figure 3.5), which features millimetre resolution, a maximum range of 5000 mm and a dead zone from 0 to 300 mm. It operates at 42 kHz, with a reading rate of 10 Hz. This sonar is incorporated in the PX4Flow kit so it can give ground distance estimates to correct the current altitude estimation.

Figure 3.5: HRLV-EZ Ultrasonic Range Finder

3.2.4 Off-board computers

On this project, two main off-board computers are used and compared.

The first one is a laptop (Figure 3.6) with a Intel Core i7-4800MQ quadcore processor running at 2.7Ghz (4 cores with hyperthreading technology), 16GB of RAM and a dedicated Nvidia Geforce 780M GPU. It runs an Ubuntu 12.04 virtual machine with ROS Hydro and OpenNI2 drivers installed.

[7] Maxbotix. Hrlv-maxsonar-ez4. `http://www.maxbotix.com/Ultrasonic_Sensors/MB1043.htm`. Online.

Figure 3.6: Used laptop

The second one is a single-board computer ODROID-U3[8] (Figure 3.7), from Hardkernel Co., Ltd., with a 1.7GHz Quad-Core Samsung Exynos 4412 ARM processor and 2GByte RAM. It runs an Ubuntu Linaro 12.11 distro, which is specific to ARM platforms, with ROS Hydro and OpenNI2 drivers also installed. This platform, given its low processing capabilities compared to a standard x86 computer, is just used to test the localization modules, and no motion planning functionality is used.

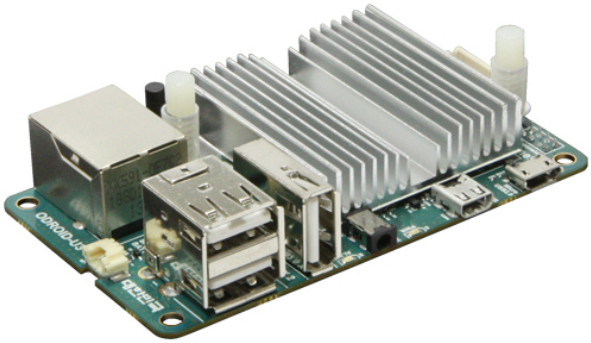


Figure 3.7: Odroid U3 board

A great specification of this piece of hardware is its size and weight: 83 by 48 mm a 48g including heat sink, which in the case of this project it was opted to use an active heat sync given the processing power being used.

After some testing, it was proved that setting up a distributed computing environment with ROS, which means that nodes can be distributed across multiple computers to share the workload and also share the data gathered in one system to the other systems, using a WiFi connection, was not feasible. The reason is that the amount of data that is shared from the OpenNI driver, which processes the data coming from the RGB-D camera, that then shared across WiFi connection is very high and overloads the connection, in a way that the connection cannot be considered reliable and working at acceptable rates.

An example of this was using the Odroid-U3 with OpenNI2 drivers, running an openni node for receiving and processing the Asus Xtion Pro Live camera data, which also includes point clouds. A connection

[8]Hardkernel. Odroid-u3. `http://hardkernel.com/main/products/prdt_info.php`. Online.

was established between the Odroid, which was running the ROS core and the laptop, which is sharing the same node data as the Odroid. The connection was both tested using Ethernet connection and a Wifi connection to a network router. To test that the depth and image data was being received on the laptop, Rviz was used to visualize the image and the point cloud data. The result was an extremely slow data sharing between the Odroid and the laptop, which allowed to state a simple conclusion: sending depth data from the Odroid to the laptop so online mapping can be executed is not feasible.

So it was opted to setup the above test bench in two ways: for the laptop, it was prepared two 5 meters USB 2.0 cables, so a connection between the FCU and the laptop, and the camera and the laptop, could be established, where all the all the vision algorithms plus the mapping and navigation module runs on the laptop; in the case of the Odroid setup itself, the tests was done with the camera and the FCU connected to the Odroid just to evaluate the localization algorithms, as stated.

3.3 FCU implementation

This section covers a summarily description of the important firmware modules implemented on the FCU that are important in what the thesis work matters or that have suffered modifications so support the application pretended for the thesis. The Pixhawk FCU running the PX4 firmware, which is the one being used in this project, turns out to be a very complex implementation, which includes a middleware, with a Linux type of OS for embedded systems, and a flight stack, which is the one that suffered some modifications and of interest to the tests being held on this thesis.

3.3.1 Attitude and position controllers

The attitude controller is based on the attitude controller explained in 2.4.2. That means it implements a PID controller for each of the behaviours that is pretended to be controlled, being it pitching, rolling, yawing or change the height with thrust. So, in its main essence, it is a inner-reset PID control loop.

The position controller also acts in a similar way to what was explained in the 2.4.2., meaning that it is an outer-reset PID control loop that receives position setpoints, having those origin on a waypoint path list defined by a ground station or in an off-board control unit, and activates the controlling sequence in a way that influences the attitude controller to move the vehicle to the desired location. A thing of notice is that the received setpoints can be of velocity or acceleration types, but the current FCU firmware implementation allows the conversion to position setpoints.

Figure 3.8[9] represents how the control loop is implemented. The reference being equal to zero means that the main objective of the controller is to bring the distance of the next setpoint or waypoint near zero.

[9] PX4 Firmware Wiki. Multirotor position control (developer page). `http://pixhawk.com/dev/multirotor_pos_control`. Online.

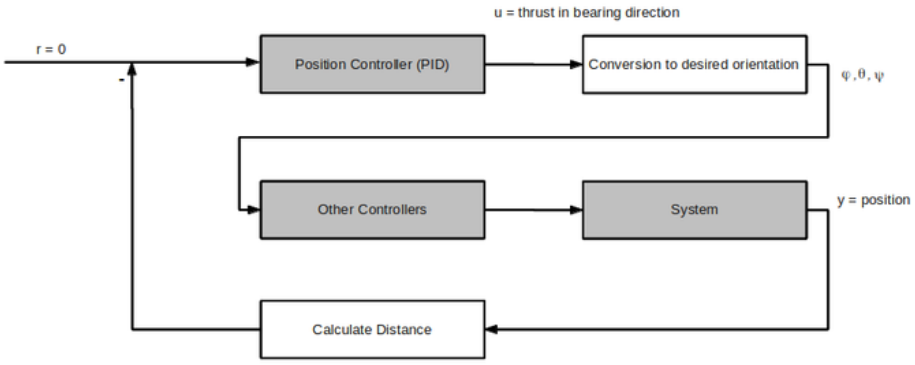


Figure 3.8: FCU position controller flowchart

3.3.2 Attitude and position estimators

The attitude estimator module on the FCU firmware implements an Extended Kalman Filter. It estimates the attitude having as source the IMU data, which includes the gyroscope angles, angular accelerations given by the accelerometers and the magnetometer heading.

So to accept the orientation given by the vision-based localization module, a modification was done in the filter: every time a valid estimation arrives to the FCU, the yaw heading, which was using as primary source the magnetometer heading, starts to use the estimated yaw orientation given by the visual estimation. This adaptation have to be done, since using body-frame orientation together with ground-frame position will result in a wrong position estimation if the first vision estimate was not received when the yaw was aligned with value zero of heading.

The position estimator implements a third-order complementary filter [41] which allows the fusion of multiple position and velocity sources so to estimate the current vehicle position. The default implementation gives the possibility of filter simultaneously GPS position and velocity, with covariance estimated error, Optical Flow linear horizontal velocity, barometer height, sonar and linear accelerations given by the accelerometer.

A thing of notice in the position estimator is that every sensor source as its own weight on the estimation, which means that the estimation depends, not only on the values being received, but in the weight that the sensor has in the filter. Other aspects that must take into account about this filter include:

- The optical flow module sensor source, more specifically, the PX4Flow module, includes a quality of measurement reading, which is taken into account on the filter. A value below the threshold means that the sensor readings will not be used on the estimator. In the other hand, if the value is above the threshold, the estimated velocity is taken into account and its ”weight” on the estimation is balanced with the ”weight” of the GPS velocity readings, which quality also depends on the calculated covariance from EPH and EPV values. The higher the quality from one reading to another, the higher the weight on the filter.

- The sonar readings are used as an offset correction to the barometer altitude readings, or in case of existing, the GPS altitude readings. That is given to the fact that the sonar readings are noisy

and depend on the quality variable of the optical flow module, since it is attached to this. So the sonar does not give the absolute altitude, but a correction between the absolute altitude sensor and the ground plane, so to give an accurate estimate of the ground plane.

- The filter includes timeouts for each sensor. Those depend on the rates on which the sensor readings or the estimations are being received and also on the quality of the measurements: a value below the threshold means no reading or no sensor data received.
- Each of the sensor or estimation sources are corrected against the accelerometer readings.

So to also include the vision estimate measurements, it was added a piece code similar to the one implemented to receive the GPS data. This allows to integrate the vision position estimate as an absolute source of position, which is corrected and filtered according to the present sensor readings and the respective weights on the filter.

3.4 ROS implementation

The following sections cover the used software modules on ROS so to estimate the localization of the MAV, to handle the mapping and navigation tasks and to handle the communication between the FCU and the off-board computer.

3.4.1 Vision-based localization estimation packages

All the used localization estimation packages relay on the use of the capabilities of a RGB-D type camera, given that is the main technology being study and used in this project given the fact that it allows egomotion estimation at the same that depth data is received, which can be used to both enhance the egomotion estimate but also to create an occupation grid map in real time.

The use of RGB-D type cameras provide dense and high frequency depth information at a low price, size and weight. Also, the depth sensor projects structured light in the infrared spectrum, which is perceived by an infrared camera with a small baseline. This though appears as a problem: structured light sensors are sensitive to IR light, which means they are generally not applicable in direct sunlight. This last issue was confirmed while doing outdoor experiments for this project.

So that the localization modules, as in the case of RGBD SLAM, could run on the Odroid, a code porting was required, since it was originally implemented to run on x86/64 platforms.

RGBD SLAM

RGBD SLAM[10] [14] is an implementation of a 6D visual graph-based SLAM framework. Its structure was presented in the 2.2.2, so it includes a frontend framework for feature extraction and matching, visual odometry and keyframe selection, a backend which includes the graph optimization, and a mapping framework.

[10] F. Endres. RGBDSLAM v2.http://felixendres.github.io/rgbdslam_v2. Online.

It allows the use of a GUI where the user can check the camera raw RGB and depth images being displayed and the processed feature and coloured point cloud map being processed. It also allows to change many kinds of algorithm parameters I/O data settings (as define the camera input topics, point cloud outputs, among others), transform information (which is useful in the case of robot localization), visual features (which include defining the features extractors, as SURF [6], ORB [44], SIFT [32] and SIFTGPU - an implementation of SIFT for GPUs), algorithm and visualization settings. Figure 3.9 represents an example of the GUI processing the data from the camera data and representing the path a map on the interface.

The processed map can also be saved as a raw point cloud or in the Octomap format, using the Octomap server topic. This allows the use of the generated map in an offline situation, where, for example, is pretended to do navigation of a robot on a known environment.

Figure 3.9: RGBDSLAM GUI

DEMO RGBD

DEMO RGBD[11] [54] is the implementation package of Depth Enhanced Monocular Odometry (Demo), which is a monocular visual odometry method assisted by depth maps and that applies the principles of feature tracking extraction, visual odometry and bundle adjustment to estimate motion a pose of a given selected frame.

The three major threads running in parallel include the Kanade-Lucas-Tomasi (KLT) feature tracker method, the visual odometry thread that computes frame to frame motion using the tracked features, which in the case of this last ones being associated with depth info, allows solving the 6DOF motion, while the features without depth help solve orientation. The third process includes the bundle adjustment to refine the estimated motion using Incremental Smoothing and Mapping (iSAM) open source library.

Figure 3.10 shows an example of the framework running and presenting the estimated frame motion and pose given the computed clouds of the environment in *Rviz* ROS app.

[11] ROS Wiki. DEMO RGBD. `http://wiki.ros.org/demo_rgbd`. Online.

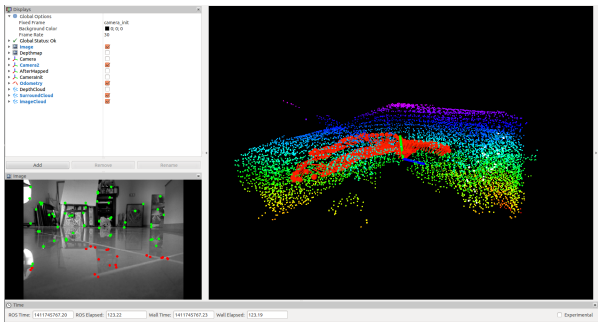


Figure 3.10: DEMO RGBD transformations and mapping on Rviz

CCNY RGBD Tools

The CCNY RGBD framework[12] is similar to the RGBD SLAM framework implementation. The main differences relay on the fact that this package does not directly include a graph optimization as in the case of RGBD SLAM, does not include a GUI, which means that the parameters must be set in the launch files, and does not add the possibility of feature extraction and matching processing using the GPU, as in the case of SIFTGPU implementation on RGBD SLAM.

Besides that, it includes the normal workflow of a visual graph-based SLAM algorithm: feature extraction algorithms as GFT (Good Features to Track), ORB or SURF, feature matching applications based on the feature extraction being used, which include ”brute-force” methods, a visual odometry algorithm to compute the egomotion from sparse features, and a keyframe mapper for RGB-D keyframes, where offline graph-based SLAM is done. Figure 3.11 is an example of the keyframes being mapped and the path of the camera frame being represented on *RViz* application.

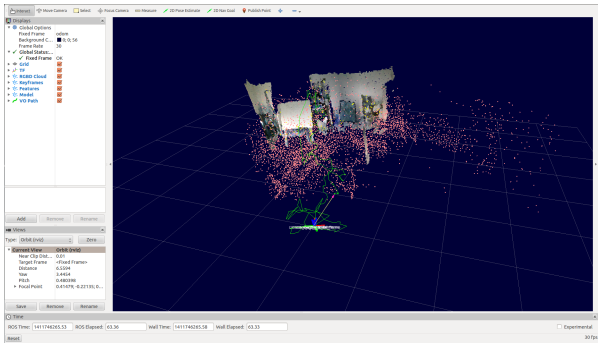


Figure 3.11: CCNY RGBD transformations and mapping on Rviz

3.4.2 MoveIt! - motion planning framework

Moveit! is a software framework developed within ROS, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation[13], and created with the idea of offering a simple and immediate solution for goal oriented actions, like trajectories to avoid obstacles within an environment, manipulation tasks, among others.

[12] ROS Wiki. CCNY RGBD Tools. `http://wiki.ros.org/ccny_rgbd_tools`. Online.
[13] ROS. Moveit!. `http://moveit.ros.org/`. Online.

One of the main advantages of this framework is that it makes available within the framework multiple external libraries that implement state of the art algorithms to solve problems of path planning and mapping, coordinating the operation by masking the complexity of the system and promoting a easier and on-the-go solution, so that the developer does not have to worry about the general issues which have already been studied and solved, but then focus on the unique characteristics of the system he is designing and the goals he wants to achieve.

The result of merging all the possible external and internal modules is the framework architecture flowchart of Figure 3.12[14]. In it can checked that there is a component of mapping, which constructs the planning scene where the robot will act. Also there is a component of user interface, which allows the access to the framework using a code API or a GUI. All of this are inputs for a main process called *move_group*, which function will be explained in the next section. There are also modules which are used to plan the motion of the robot based on a collision checking library and motion planning library, which together will help the *move_group* to actuate on the robot motion controllers.

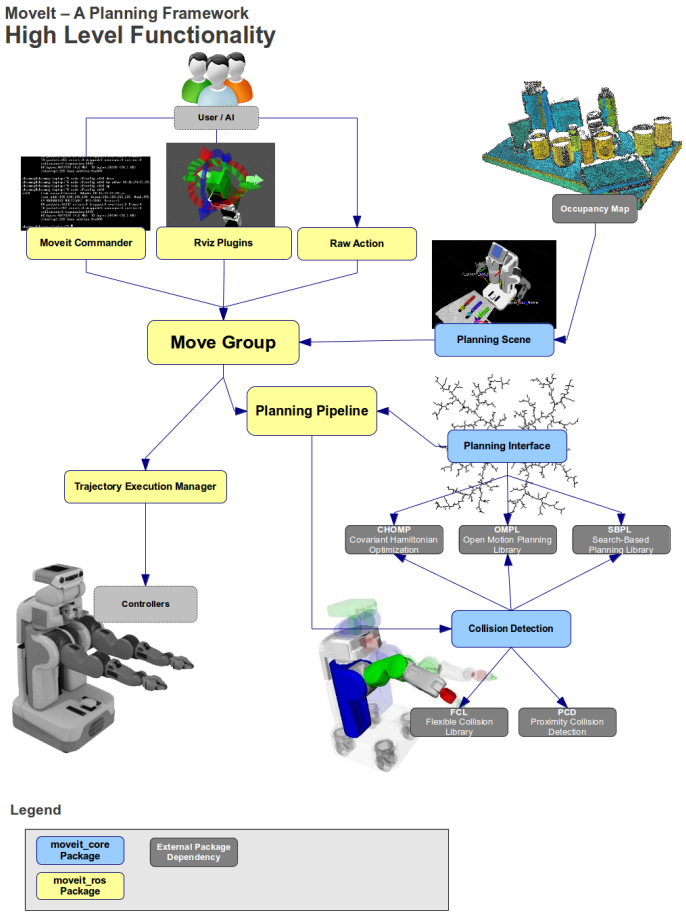


Figure 3.12: MoveIt! framework high-level architecture

The framework was developed in a modular way so it can be easy to expand by replacing some parts with other custom to be able to easily adapt to user needs. Besides that, all the configuration is done by external files (*.yaml* files in most part), generated largely automatically by a setup configuration interface

[14]ROS. Moveit!. `http://moveit.ros.org/`. Online.

where a description of the robot is given.

The main process - *move_group*

All the system architecture works around a primary ROS node called *move_group* which has the task of coordinating the operation of all the libraries involved and handle user requests. The system architecture can be checked in figure 3.13[15].

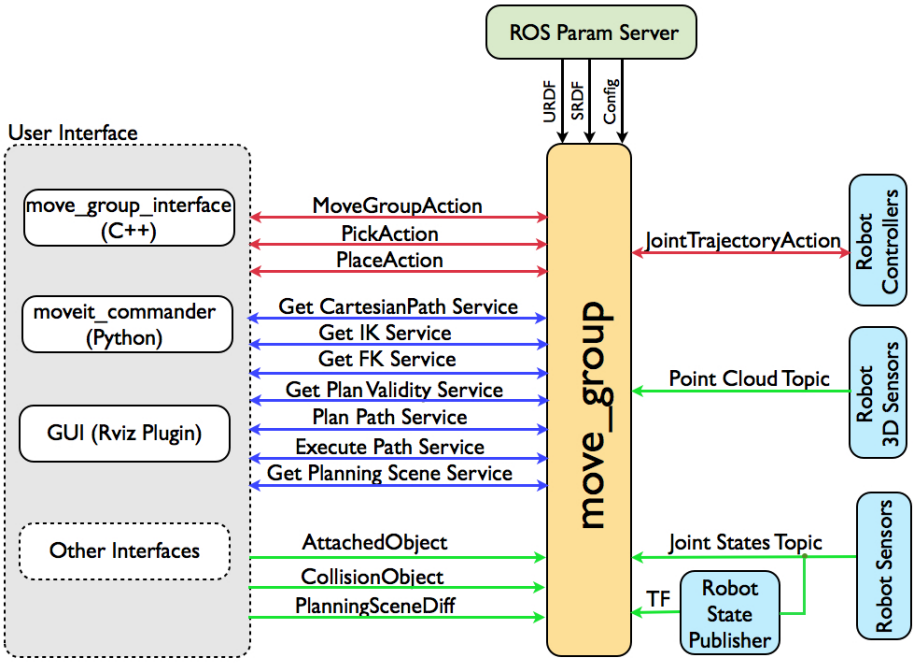


Figure 3.13: System architecture

By using ROS parameters, it requires that a description of the robot relative to its joint, links, sizes and sensor plugins in what is called a URDF[16]. This description is given by ROS *robot_description* parameter.

Besides the previous description, *move_group* also needs a somewhat similar description called SRDF[17], which is a representation of semantic information about robots that includes other information that is not included on the URDF itself, giving a more user readable description of the interactions between the different parts of the robot.

move_group will also take into account other configurations specific to MoveIt!, which include joint limits, kinematics, motion planning, perception and other information. This info, as said, is automatically generated in configuration files for these components by the MoveIt! setup assistant and stored in the configuration directory of the corresponding MoveIt! config package for the robot.

For this specific project, also a process that publishes the odometry of the robot is needed. In the implementation, a transformation[18] between the local origin of the environment and the MAV. The pose

[15]ROS. Moveit!. `http://moveit.ros.org/`. Online.
[16]ROS Wiki. URDF. `http://wiki.ros.org/urdf`. Online.
[17]ROS Wiki. SRDF. `http://wiki.ros.org/srdf`. Online.
[18]ROS Wiki. TF. `http://wiki.ros.org/tf`. Online.

of the robot, as explained in the project setup overview, comes from the estimator running on-board of the MAV, and it is used in a way that each link shown is the transformation between the reference system centred in the MAV links, which are mainly the camera link and the MAV FCU, and the local origin reference system, which is absolute and fixed. This way it is possible to derive the position of the components of the MAV in space and bring to an absolute reference system any relative measurement, which include, not only the pose of the MAV, but also the position of the different points of the processed point cloud.

The planning scene construction - Octomap

Since the objective of this project is a possible real time application of map-based navigation, it is necessary a process that can publish a point cloud that comes from the camera sensor so there can be a live reconstruction of the environment where the robot is actuating.

For that, MoveIt! uses an implementation of the Octomap framework, that implements a representation of the environment map using octrees and exploits probabilistic functions to update the map construction from point clouds and odometry of the robot.

As previous explained in subchapter 2.3, the framework is able to guarantee a very detailed map of the points observed without an high memory usage, offering the ability to model the occupied, the free space and the space not yet explored. The efficient management of the memory is guaranteed not only by the definition the map chosen by the user but also, as seen, from the tree structure of the data: if the children of a node all have the same value you can delete them and leave only the parent node to describe that portion of space. That way, the quantity of information to be stored is the minimum and the subsequent queries on the map are speeded.

Queries are also constructed based on the hierarchical structure in which the data is stored. As an example, if a minimum resolution such as a cube of 5 cm of side is considered, only 16 levels of tree are considered to represent an area enclosed in a cube with 327 m of side, which is more than enough for the majority of applications.

MoveIt! integrates the Octomap framework so to obtain high performance. Because of that, the resolution of the map is relatively large and does not consider any other type of data besides the point cloud. For example, RGB-D data could be considered in a way that we could obtain a coloured map, where each voxel as a colour assigned by the RGB sensor.

Given the modularity composition of MoveIt!, the user is able to specify the type of sensing plugin that is being used, so that, for example, it can use a depth image or a point cloud (Figure 3.14) representation of the environment, depending on the type of Octomap updater sensor plugin that is being used.

The motion planning process

Not being the main priority of this thesis, it is important to point out some specificities on how the navigation stack of MoveIt! works, as it was already tested in previous work.

For the navigation and planning issues, MoveIt! integrate a communication with motion planners from

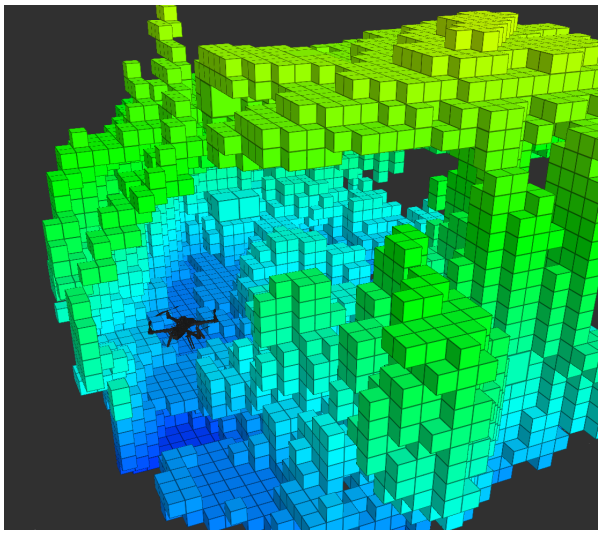


Figure 3.14: Moveit! mapping capabilities being shown on Rviz

multiple libraries using a plugin interface, which proves the extensibility and modularity of this framework. This kind of integration is possible with the use of a ROS Action or service offered by the *move_group* node.

When the user issues a planning request, the motion planner will have in consideration several factors in the planning of the motion of the robot. It will not only have in consideration the path to the described goal or end-effector considered, but it will also consider the kinematics constraints of the robot, like position, orientation, visibility, and in the meantime, check for collisions using FCL library for the purpose.

After computing the path, the *move_group* node will issue the motion, not only considering the path that it has to take based on the result of the planning process, but also having in consideration the velocity and acceleration constraints that have to be considered to the joint.

In some specific situations, MoveIt! allows the use of pre and post-specific planning request adapters, which include starting state constraints, planning area boundaries and time parameterization.

A planning pipeline flow is show on Figure 3.15[19].

For the planning process, OMPL library is integrated. OMPL[20] is an open source library that implements many state-of-the-art algorithms used for motion planning. The library was created for use in both research environment and robotics industry, combining modularity with high performance by simplification of the library components and making them completely free from each other, which make it easily extensible and efficient.

What the library offers is simply a series of optimized implementations of the main path planning algorithms. This includes algorithms as RRT, PRM, KPIECE, among others. While it is not explicitly represented the geometry of the robot or the environment when navigating, this kind of algorithms can also be used in other fields beyond robotic applications.

The idea of functioning of this library is very simple: given a set of elements and a validation function

[19]ROS. Moveit!. `http://moveit.ros.org/`. Online.
[20]OMPL. docs. `http://ompl.kavrakilab.org/`. Online.

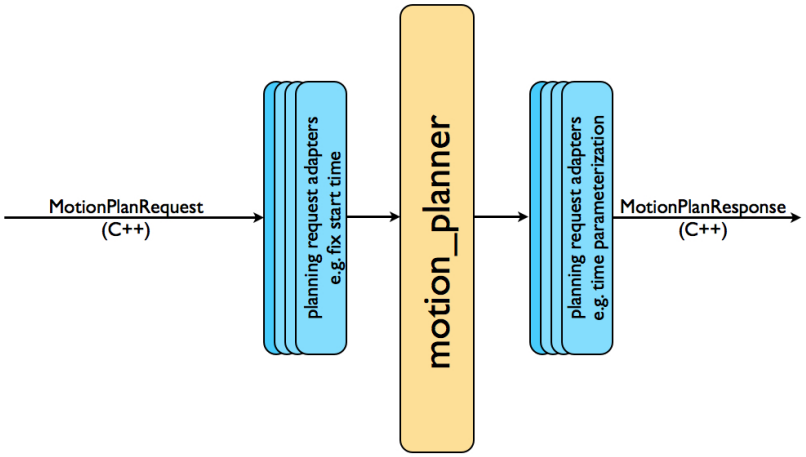


Figure 3.15: Planning Pipeline

of those elements, the chosen algorithm to act tries to find a path between two points belonging to that set of elements.

For the specifications of a MAV, considering a goal and a starting state, and since the user has the possibility of using whatever algorithm the OMPL library offers using the intuitive interface of Moveit!, given that the configuration space that the robot can occupy is 3 dimensional and using a validation function that will verify the absence of collisions with the world and respect the constraints of self-collision and the ones given by the user, the motion planner tries to find the best path between the starting and the goal point.

So that the path between two neighbouring nodes of the path can be verified, a special component, the Motion Validator, checks the validation based on the interpolation of a discrete number of intermediate states between the two path nodes to be connected, which allows that this library can work with both the geometric planning and for control-based planning by simply changing the set from which the sampling takes place and the function that checks the validity of the states.

The used setup

For the thesis project, and given the previous work already developed in the Autonomous System course during the 2013/14 semester [17], supported with what was implemented in [52], it is considered a similar configuration to the one used on that project, on which is used a simulation that includes the use of the simulated quadrotor from *Hector_quadrotor* package[21] in a Gazebo[22] simulation environment.

The only differences between the application on the project simulation and the application on the real system is in the sensor and the estimated localization sources. The localization being fed to MoveIt! module in the simulation case relay on a simulated ground truth system plugin that is offered by the

[21] ROS Wiki. Hector quadrotor. `http://wiki.ros.org/hector_quadrotor`. Online.
[22] Gazebo Project. Gazebo sim. `http://gazebosim.org/`. Online.

Gazebo environment itself, which in simple description, means it is not a pose estimation but the true pose. In the case of the sensor source, the robot description of the *hector_quadrotor* model uses a plugin that simulates an RGB-D sensor, which allows to use a simulated camera with as the source of the point cloud being used in Octomap and being fed to MoveIt!

In the real system, as already shown in the beginning of this chapter, the estimated localization of the quadrotor comes from the sensor fusion framework on the flight controller and the sensor source for mapping is an Asus Xtion Pro Live.

In both cases, the configuration follows and includes this steps:

- The creation of an URDF robot description using the setup assistant, which automatically creates all the files needed to run the application. This description includes a series of links, that is, static elements, connected by a joint, movable joints of the robot, for each of them are described dimensions and degrees of freedom in addition to a geometric description and viewing given by the 3D mesh. For the case of the of simulation work, *hector_quadrotor* URDF description file was used, while for the real system, and after the creation of a 3D model of the quadrotor (Figure 3.16), a new description file is used, which is similar to the *hector_quadrotor* one.

  In this process, it is also specified the planning groups on which MoveIt! will act, i.e. the sets of links and joints for which the system will schedule trajectories. For the case of the MAV, the considered joint is a fluctuating joint, since it is above the ground without any kind of physical link attaching the robot to it.

  A planning group that includes the *camera_link* and the *base_link* (which, in this case, represents the vehicle/body frame of the MAV) is created.

  Also in this process, dynamics and kinematics constraints are added. The first include limiting the velocity of the motions to 1 radians per second for angular rates and 1 meter per second on linear velocity, due to the limitations of a system like this to respond in a safe a reliable way to the motion commands that receives. The second one include not allowing the planning of a 3D configuration that surpasses more/less than +/-15 degrees of pitch or roll. The reasons are easy to understand: firstly, the used rates for velocity do not allow any kind of acrobatic manoeuvre; secondly, it is impossible to use acrobatic control in a real system which relays on an on-board vision positioning estimate system that is not fast or accurate enough for computing good estimations when the vehicle does an aggressive manoeuvre.

  An example of the MoveIt! setup assistant window can be seen in Figure 3.16.

- The configuration files previously produced in the last process are tweaked by the user so it can include the pointers to the created controller that issues the trajectories and to the sensor sources plugin that acquire data of the environment.

  It was considered the use of the point cloud occupancy map updater plugin for generating the map, limiting the range of processing data for 5 meters and using a 10 cm resolution for the occupancy grid. In the case of the controller, it was pointed and used the controller specified in the next point.

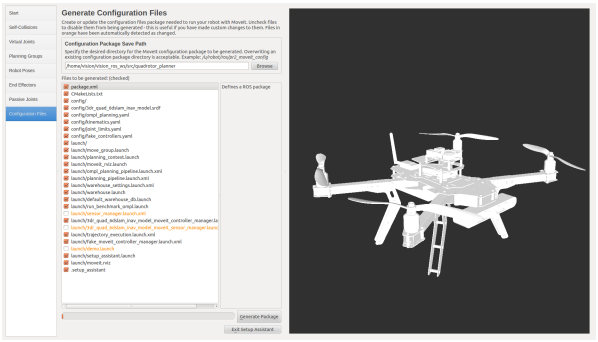


Figure 3.16: MoveIt! Setup Assistance with MAV created model

- The used controller on this project is a simple controller that implements a ROS Action. It includes a modification of default *SimpleControllerManager* and an *ActionController*, which is an independent node that uses *MultiDofFollowJointTrajectoryAction* messages, included in the *actionlib* library of ROS to issue trajectories as a goal composed by a series of points in a 3D space. Its way of working is simple: given a trajectory input by the motion planner, it converts the trajectory into velocity commands, that use the ROS *Twist* message API, to be performed by the MAV, as seen in the beginning of this chapter, in the diagram.

- The interaction with the framework is the last step and the main objective, which can be done using C++ code or using the ROS Rviz GUI.

  The use of the GUI is preferable in the cases where it is the user that issues the control over the vehicle, because the interface itself lets the user choose what type of path planning solution to use, the bounds of the planning area, the start and goal points, the pose configuration of the vehicle in a given path node, among other options, besides allowing the user to see the current map being constructed after sensor data processing and the path being followed by the vehicle.

  In the other hand, in a case where the system performs an exploratory behaviour of the environment, i.e. it is not the user that controls where the MAV has to go but it is the vehicle itself that chooses what path to follow based on an exploration control routine, a C++ type of interaction is more useful, since there is no need to access the planning conditions at motion time.

  Since for the work done in the thesis, the objective was not to implement an exploratory behaviour, it was opted to use the GUI interface to check both localization and mapping processes, without issuing any kind of navigation commands, as already stated. So, given the time bandwidth to develop the thesis, only the estimated pose of the MAV and the map of the environment was verified on the GUI, only lacking testing the velocity control on the real system. For that, as it can be checked on the next subsection regarding the ROS-Vehicle communication, all the code structure has been prepared for testing, as the velocity commands can already be sent to the real MAV system.

3.5 ROS-Vehicle Communication

3.5.1 Mavlink Protocol

MAVLink is a lightweight, header-only message marshalling library for micro air vehicles[23]. Its main purpose is to establish a standard communication protocol that can be used by all types of flight controllers on different types of vehicles. It offers the possibility of packing C-structs over serial communication channels with high efficiency and send these packets to the ground control station or any other system that implements the protocol decryption, allowing it to be the backbone for the FCU/IMU communication as well as for Linux interprocess and ground link communication.

It can simultaneously support 255 aircraft, ranging from 1 to 255, since 0 is not a valid aircraft ID. Also it just uses 8 bytes overhead per packet, including start sign and packet drop detection, can work in multiple platforms and architectures and relies on ITU X.25 checksum for packet corruption detection, which guarantees the safe functioning of the system.

Protocol specifications

Since it works with critical systems as the UAV's are, MAVLink is implemented to work in a way that guarantees the proper functioning of the aircraft and avoid critical communications failures between the aircraft and the ground station or the systems that it is connected to. Also, given the lightweight implementation, it allows fast data transfer rates, which are essential in a system that needs live data and effective fast commands as a response to the dynamic situations where it is implemented.

MAVLink implements protocol internals and a message API in serial link data streams, and so, it has to decode and encode its own messages from the byte stream. The MAVLink packet has the structure of Figure 3.17.

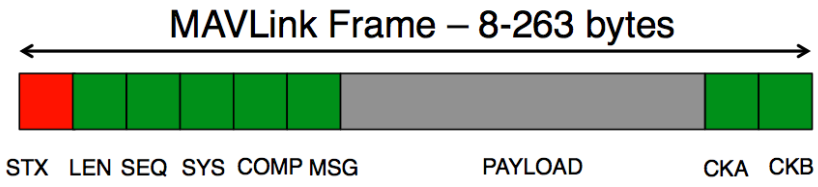


Figure 3.17: Mavlink packet

where, considering a n bytes of length depending on the type of message:

- byte 0 is the start of the packet;
- byte 1 is the length of the payload;
- byte 2 is packet sequence number, which allows to detect packet loss;
- byte 3 is the system ID of the sending system, which allows to have different MAVs on the same network;

[23] QGroundControl. Mavlink. `http://qgroundcontrol.org/mavlink/start`. Online.

- byte 4 is the component ID the sending component, which allows to differentiate different components of the same system, e.g. the IMU and the autopilot;
- byte 5 is the ID of the message;
- byte 6 (to n+6) represents the data field of the message, which depends on the message ID.
- byte n+7 (to n+8) represent the checksum field.

To detect and decode messages, MAVLink waits for the packet start sign. After that, it reads the packet length and matches the checksum after n bytes. If the checksum matches, it returns the packet and waits again for the start sign, while if the checksum does not match, it means that the some bytes are altered or lost, which result in an drop of the current message and continue the next try on the following message.

So to guarantee the safety of functioning of the system it is included, the protocol is included in the safety critical components of an unmanned air system, and for that it includes a sequence in the header which allows MAVLink to continuously provide feedback about the drop rate. That is a useful information for the aircraft or ground control station so it can take action in the case of a communication or a system failure.

The rate performance of the transmission varies between 20 and 100Hz, depending on the established baud rate link speed.

3.5.2 MAVROS - a ROS-Mavlink bridge

MAVROS[24] is MAVLink extendable communication node that establishes a bridge between MAVlink and the ROS framework. The main idea of its creation is to add the possibility of extending the possibilities of the ROS framework to whatever unmanned device which uses as main communication protocol the MAVlink protocol.

Given its modularity, flexible and steady implementation, MAVROS also allows that external links can be established with ground stations, i.e. the main node may be running and sharing the received data over TCP or UDP to other clients that want to receive and send data through the serial stream where MAVROS is connected to, from where it receives and sends MAVLink packets.

3.5.3 System architecture

The package consists of three main components (Figure 3.18):

1. ***Mavconn***, which is the application layer abstraction library which receives and sends MAVLink messages.
2. ***nodelib***, that includes the core of MAVROS and its functions and classes.
3. **Plugin library**, which is a common set of plugins that extend the functionality of ROS to the use of MAVLink messages.

[24] ROS Wiki. MAVROS. `http://wiki.ros.org/mavros`. Online.

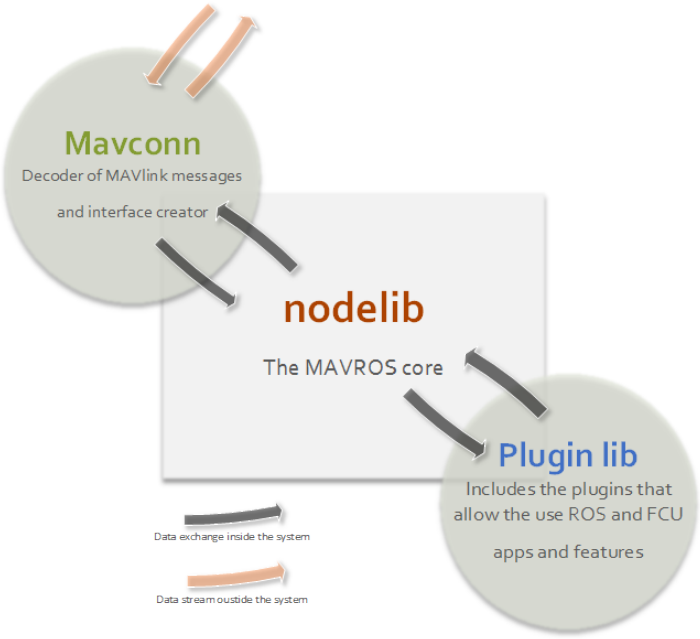


Figure 3.18: MAVROS System Architecture

3.5.4 Mavconn – the decoder between ROS and MAVLink

Mavconn may be considered the main component of the system, because without it, the reason of existence of the package itself would be lost: *Mavconn* can be included as an Application Layer filter that allows the decoding of MAVLink messages to the below layers.

While the MAVLink protocol only provide functions to stream and parse messages, there's a need of interpret those messages and pass them to the applications that need them to be used. *Mavconn* gets use of the parsing helpers included in the MAVLink protocol and does the decoding and encoding of messages, from and to a data stream respectively. Also MAVLink protocol itself does not provide functions to access the stream itself, and for that *Mavconn* uses a set of *Boost* library functions to do that kind of access.

The normal flux of data that passes through the mavconn abstraction layer includes:

1. Establishing a serial link to a data stream where MAVLink messages are being sent.
2. The messages are read using *Boost.Asio* library functions and stored in a allocated memory.
3. The messages are parsed using a MAVLink helper function that, after checking the message CRC, decomposes the message header and stores the metadata and the payload on a struct of type *mavlink_message_t*, which structure depends on the type of message that is received. That structure is identified by a configuration *XML* file which defines the type depending on the message received.

Mavconn also has a URL type of parsing which allows to define the type of interface connection being used and the characteristics of that connection, i.e. if one wants to define a serial connection, it can define that connection with an URL. This feature is included on its connection class and allows the

creation of interface abstractions with included URL and the Instance.

Mavconn is responsible also for channel allocation with recursion to MAVLink helpers, i.e. sessions of connections that allow the creation of buffer arrays of data and check the status of the MAVLink connection. For that, it creates a thread for every ”channel” or interface instance so that all message handlers can make function calls from that thread. Those threads run in parallel several ROS middleware threads launched by the MAVROS node.

3.5.5 Nodelib – MAVROS core

The data is then handled in the MAVROS main class node, *MavRos*. This is the core node of the system and it is considered the nodelib, a library of all functions and classes of the system. It is used to:

- Using URL parser of mavconn, construct a mavconn connection class, which is interpreted as an interface between the core where all the data flows and the nodes that need and use that data, which includes the ground stations and the plugins. So this can be used as an analogy to the client/server topology, where the server is *MavRos* class and the nodes are the ground stations and the plugins, which are connected with interfaces constructed depending the type of communication protocol being used, let it be TCP or UDP.
- Manage the translation between the FCU and MAVlink ROS message type (*Mavlink.msg*). This message is useful because, even though any message can be stored in *mavlink_message_t* struct, it can also be stored in *Mavlink.msg* too. This message is used by any other node that wants to have access to the FCU link and it is the type of message used by ground stations, to which point-to-point UDP bridge can be built, allowing multiple ground stations have access to the same link.
- Load plugin classes using the *pluginlib* library of ROS. The idea is loading a dynamic library and create class instances depending on the type of plugin. For that, it also uses *XML* descriptors to help finding the adequate instance to launch. That way the messages can be routed to the correct plugins that use them, or vice-versa, route the messages from the plugins to the data stream.

3.5.6 Plugin library

The plugin library is a common set of plugins shipped with MAVROS package. They are loaded by MavRos class using the *pluginlib*, as multiple classes from one dynamic library. That library consists of special tags which are defined by *PLUGINLIB_EXPORT_CLASS* macro.

Also, as seen, a special description *XML* file is used to map the library, class and base class of the plugin, allowing the creation of an instance of a class depending on the type of MAVLink message to be handled. Class aliases are used to allow blacklisting the plugins that the users do not want to use in their application. It is to note that some plugin are required by others to work properly, but for a question of simplicity, dependencies check was not added to the module.

The usually required plugins are:

- ***sys_status***, which gives update info on the connection status and stores autopilot type to UAS, which is a context storage class to identify the FCU, type of vehicle and other relevant info about the connected system. Many other plugins use this plugin also to change the connection status signal to safely restart operation.
- ***imu_pub***, which stores IMU data in UAS, so other plugins can use it without having to use a ROS subscriber.
- ***gps***, which saves GPS location and the one sigma estimated position errors, EPH and EPH. Also gives the status of satellite fixing so it can be used.

The previous plugins are plugins that are required by others to work properly. But for this project in specific, there were implemented several other plugins which are useful for three main tasks: send the localization estimation from a running algorithm in another ROS topic, receive the estimated pose from the FCU after being filtered on the sensor fusion loops, and also for sending control setpoints from the motion planner or navigation stack running on an external system to the FCU.

So, given the plugins implemented, they can be divided in two different categories: estimate plugins and setpoint plugins.

The first category include all the plugins that are used to send or receive estimated poses to or from the FCU. This includes:

- ***imu_pub***, which as said, stores IMU data in UAS, so other plugins can use it without having to use a ROS subscriber. It also provides the attitude of the vehicle where the FCU is mounted, which off course is a result of attitude estimation running on it. It can be used in conjunction with the following plugin to give the pose of the MAV to be used by the ROS framework.
- ***local_position***, used to receive the local position of the MAV, which is sent by the FCU after being estimated on the on-board sensor fusion filter. It is very useful for two purposes: it is the main source of position of the MAV and, in conjunction with the attitude, it gives the full pose of the MAV, which can be used by MoveIt! or other navigation stack to receive feedback of the pose of the vehicle. Also this feedback can be retrieved by the user using a vizualization tool like *Rviz*, the second purpose of using this plugin.
- ***global_position*** has the same purpose of the previous one, but retrieves the global filtered position coming from the FCU, i.e. the estimated GPS position with accelerometer correction. This plugin gives the possibility of convert GPS coordinates into UTM coordinates, which are linear coordinates like the local position coordinates. And, for that reason, this plugin can be, in a future work, very useful to implement a MAV system that can navigate from indoor to outdoor spaces and vice-versa, changing the source of localization dynamically.
- ***vision_pose_estimate*** provides a pose estimator to the FCU, which comes from a running pose estimator package on a off-board computer. In the case of the thesis project, this plugin is essential because it is the plugin that sends the estimated position and attitude of the vehicle to the FCU so it can be fused on the sensor fusion algorithms on-board.

- ***vision_speed_estimate***, provides a speed estimator to the FCU, which comes from a running speed estimator package. In the case of the thesis project, this plugin is not used but may be used on the future so that velocity estimations can be fused onboard the MAV to better pose and motion estimation.

- ***mocap_pose_estimate***, allows sending a pose estimator to the FCU comming from a motion capture system which is connected to ROS and publishing the pose estimations. In the case of the thesis project, this plugin is not used also but may be used on the future as a way to also test a more precise localization system or even to fuse multiple estimate sources onboard the MAV.

The second category includes the setpoint control plugins, i.e. all the plugins that allow offboard control of the MAV using setpoint vectors. This category includes:

- ***setpoint_position*** receives a position vector with x, y and z coordinates and sends it to the FCU as a setpoint to which the MAV should move to. The onboard position controller assumes that position and forwards it to the position control loop so that the system adjusts its position on a local area. Also, this control mode allows to define the angle of yaw of the MAV.

- ***setpoint_velocity*** works the same way as the previous plugin, but instead uses a velocity vector with vx, vy and vz velocities and sends it to the FCU as a setpoint of velocity at which the MAV should move. Given the Action Controller written and used on previous work, as seen in chapter 3.2.2, this is the adequate mode to use. Future work just requires testing this mode and check the behaviour of the MAV given the received velocity commands. So, it is just a small step until achievement of autonomous control. To note that also this mode allows control of yaw of the MAV based on a setpoint of rate, which is also assumed by the FCU.

- ***setpoint_accel***, after receiving a vector with linear accelerations or force values (afx, afy and afz), it sends it to the FCU as setpoint of force or linear accelerations, which basically is a mode that changes the position of the MAV using accelerations.

- ***setpoint_attitude*** directly controls the attitude of the MAV. It can control roll, pitch and yaw of the MAV with direct input of this values using a vector, which controls the angle or the rate, or using a quaternion representation. It also allows control of the thrust of the motors of the MAV to control the response of the quad to the attitude commands. It is the most straight forward control method and only influences the onboard attitude controller of the FCU.

Given that the coordinate frames used on the FCU are in NED convention, which means that X axis points forward, Y axis points right and Z axis points down, all the plugins described before have a part of code where a conversion between frames, being the vehicle/body frame or the local ground frame, is done, given that the ROS framework works using the ENU convention. Each ENU and NED systems rotate along with the earth, so they cannot be considered inertial reference frames. This conversion can be checked on Appendix A.

Chapter 4

Results

The next sections will cover the results obtained from the tests performed with the system architecture presented in the previous chapter. It is also performed an analysis of these results taking into account the characteristics and constraints presented for each of the tests, particularly with regard to the testing area, the particularities of used algorithms and features specified for the system.

4.1 Tests specificities

First, it was taken into account that, if it is necessary to test localization architectures, it is indeed needed some kind of system that can give a relatively good estimate of the vehicle position, so a comparison between the estimated pose and this estimate could be done, giving an estimation error. But, reliable systems as Vicon or Optitrack were not available, so an alternative was considered: the use of a marker-based localization system. The used package was *ar_track_alvar*[1], which is a ROS application presented as an evolution of AR Toolkit.

The usage of the marker tracking package was done the following way: it was chosen an AR marker (Appendix B - Figure B.1(b)) to identify the quadrotor, which was placed over the vehicle. Then, another marker was chosen (Appendix B - Figure B.1(a)) so to identify the local origin of the area.

4.1.1 Testing area

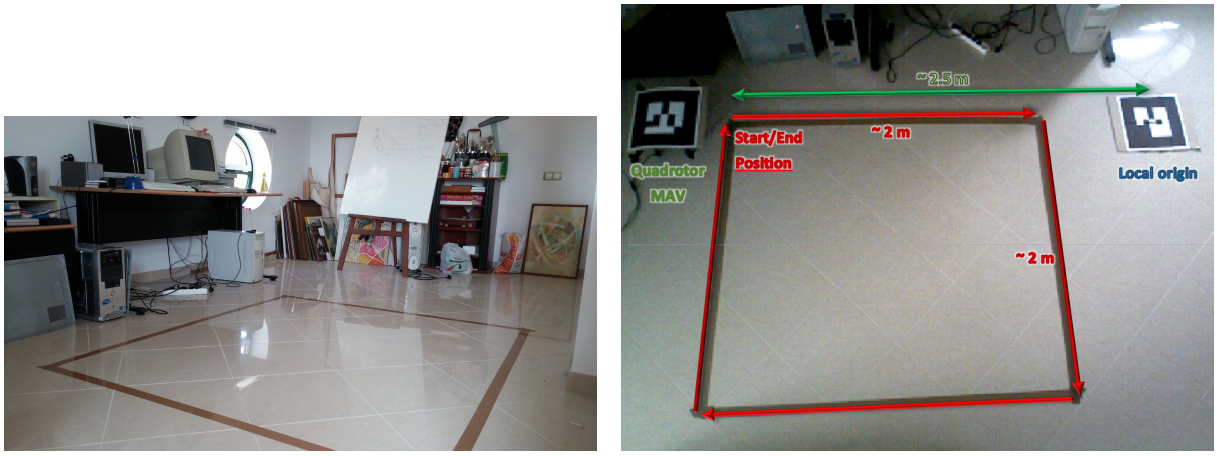
Testing area 1

The first chosen testing area was a space in an indoor area with more or less 3x5 meters. In that area, it was installed a RGB-D camera at the top (3.5 meters above the ground), pointing to the ground with a certain angle.

This area had to be confined for a reason: given the camera position, the camera vision angle is not large enough so it cover all the attic area to track markers. So, it was defined a 2x2 meters area in the centre of the focal area of the camera, so the quadrotor could be tracked inside that area. Then, it was

[1] ROS Wiki. AR Track Alvar. `http://wiki.ros.org/ar_track_alvar`. Online.

decided to define a path the MAV must follow. The path is a simple square that follows the area defined for tracking (Figure 4.1(b)).



(a) Indoor area overview

(b) Marked course seen from the installed camera

Figure 4.1: Indoor tests area

The movement starts in the top left corner, follows in front, then keeping the same yaw orientation, it keeps the movement along the edges of the delimited area, until it reaches the same spot. As it can be seen in (Figure 4.1(b)), the local origin was placed 2.5m ahead of the starting point of the quadrotor, since it is well a backlit area (there are now artificial light sources available) and it is in a place where its tracking is not lost, given that the quadrotor could cross the line of sight between the camera optics and the marker.

All the datasets were made with the MAV handheld. The reasons are the following:

- Given that the area covered by the camera for tracking the quadrotor, a not well calibrated flight, which includes the consequences of the vehicle dynamics and the ground vortex effect, can result in a long failure of the tracking system, which invalidates the test
- Since the area of testing is very short and there is the presence of an obstacle rich surrounding environment, which again, given the dynamics of the quadrotor, and given that is not easy to control, can result on damaging the quadrotor or the obstacles around.
- Since the purpose is to evaluate the performance of the localization algorithms against a ”ground truth” system and knowing the a pre-established path, handholding the quadrotor allows a better control of the movement of this and a better path structure to compare to the ground truth and the defined path on the floor.

Testing area 2

A second testing area had to be chosen. The main reason is the following: to the optical flow module to operate, given the software architecture implemented [26], it needs to have good readings. Those readings imply that the CMOS camera must get data readings from a high texturized surface so that the feature extractor algorithm running on the module can calculate the linear motion of the camera frame

relative to the surface plane. Those readings are evaluated by a ”quality” variable, which values vary between 0 and 255. The acceptable quality threshold is above the 120.

In the case of the floor on the indoor area, it was impossible to use the optical flow measurements since the floor is made of reflective tiles, without any kind of texture. So all the indoor tests were made without the optical flow module velocity measurements being fused on the position estimator of the FCU firmware.

So that the horizontal velocity measurements could also be considered in the position estimation, it was chosen as a testing area an outdoor area, where the floor is made of cobble stone, which pattern is rich enough for getting acceptable “quality” values of the measurements. The testing area (Figure 4.2) was used to evaluate the performance of a vision-based localization system using optical flow velocity calculations and a system without it.

(a) Outdoor area overview

(b) Marked course seen from the installed camera

Figure 4.2: Outdoor tests area

4.1.2 Localization modules parameters

The graph-based SLAM algorithms, as seen, give numerous possibilities of tweaking the backend and the frontend framework applications. CCNY RGBD only allows the tweaking of the frontend, since it does not have a mean of optimizing the graphs.

So, for a fair comparison between the SLAM algorithms, it was chosen the following constants and thresholds for both:

- Feature detector: SURF
- Max distance between keyframes: 15 cm
- Max angular distance between keyframes: 10
- Number of features per image: 400
- Max number of iterations: 10

Off course this parameters can be tweaked in a way they can be neat optimal for the considered testing conditions, but that was not take into account, as the main purpose is to test the difference between each type of pose estimation framework.

The main difference between the SLAM frameworks is that CCNY RGBD uses ICP for pose estimation and no post graph optimization, while RGBD SLAM uses RANSAC with transform verification by EMM - beam-based environment measurement model - and uses graph optimization with the g_2o library[2].

In the case of DEMO, there is an option of using the bundle adjustment for path correction or not. After some analysis, and given that there is a ”slimmed down” version available for embedded systems like the Odroid, it was decided to do all the tests without the bundle adjustment. The analysis made also allowed to note that the visual odometry for itself, i.e. without bundle adjustment, is capable of running slightly fast on both laptop and Odroid, which then leave to the conclusion that, given the system purposes, a bundle adjustment is not applicable.

4.1.3 FCU estimators parameters

As presented, the position estimator judges the sensor data depending on relative weights of the different sensor sources. It is known that the barometer measurements are very noise in indoor environments, so the default weight on the position estimator was reduced. The sonar weight was increased, but since this one, as already explained, only takes role on the height offset correction and it is not used as a primary altitude source, the weight increase will not do that much effect. Also, as the sonar readings are unstable, noisy and very subject to interference from highly reflective surfaces, it is comprehensible not to use the sonar as the main source of altitude.

So, considering the PX4 Firmware API, the following parameter weights were defined on the FCU: (note that the minimum weight is 0 and the maximum weight is 10):

- Vision XY weight: 7
- Vision Z weight: 4
- Sonar weight: 10
- Barometer weight: 0.01

Then, it must also be considered the weight of the optical flow measurements. As stated and verified, since the indoor testing floor area is a reflective kind of floor, without any kind of texture, the “quality” measurement variable of the sensor is always low (below the 100, in a scale between 0 and 255). So it is considered a value of 0, since the sensor cannot be used.

In the case of the outdoor testing, the case is different, given that the cobble stoned floor has enough texture to be considered feature reach, and so the quality variable increases, and the measured linear velocities are considered in the position estimator. So, for comparison between a non-velocity-fed estimator and a velocity-fed estimator, it was considered the same weights as above, plus:

[2]OpenSLAM. g2o. `http://openslam.org/g2o.html`. Online.

- Optical Flow XY weight: 0/5

Which means for the case of evaluating the system without the use of velocity estimation, it is used a 0 weight value, and a value of 5 for the opposite situation.

4.1.4 Datasets

Dataset 1: The first dataset was done on the indoor area, with the main purpose of evaluating and compare vision localization frameworks algorithms between each other and relative to the ground truth. This estimates were not fused on the FCU, so the position and attitude estimator of this one were not used for this tests.

The test was done, as stated, with the quad handheld and doing a pre-determined path in the test area: follow the area edges, performing a square shape path, without changing the orientation.

Dataset 2: The second dataset tests were made under the same conditions of the first dataset, but with a main difference: it is made a test for each of the localization frameworks having the estimated pose being fused in the FCU position and attitude estimators. The main idea is then to evaluate the performance of using the sensor fusion framework against not using it.

Dataset 3: The third dataset is made of two tests made in the outdoor area, with the main purpose, as stated, of evaluating the use of optical flow measurements in the estimation of the localization of the system.

Dataset 4: This dataset has the main purpose of evaluating the performance, in time and precision, of using the Odroid-U3 for running the ROS side framework.

Since after some tests, it was understood that the Odroid itself is not capable of handling the RGBD SLAM acceptably, it was decided just to make a comparison between running DEMO and CCNY RGBD on the Odroid and the “ground truth” system.

Dataset 5: So to evaluate the mapping framework, this dataset was made in the indoor area, having the quadrotor pointing in the same direction for each test.

Using an uninterrupted stream of camera data, the idea was to evaluate the MoveIt! mapping capabilities using the Octomap plugin. For that, it is made a comparison between three grid resolutions of the same map: one with octree edge size of 5 cm, which is the higher resolution of the three tests, 10 cm, as considered the ”medium” resolution, and 15 cm, as the lowest resolution.

All the datasets are available in *.csv* file format in [36], converted from previously created *.bag* files of the *ROSBAG* application. Table 4.1 sumarizes a description of each localization tests datasets.

4.2 Localization results

The main evaluation will be made in terms of time and localization accuracy performance, considering that the marker system is not a true reliable ”ground truth” system and also has errors associated, which are not considered for the case.

Datasets	Tests	Duration (s)	Number of samples
1	DEMO	27.49	274
	CCNY RGBD		133
	RGBDSLAM		651
2	DEMO	26.76	760
	CCNY RGBD	20.95	595
	RGBDSLAM	42.81	1215
3	DEMO (no flow)	30.12	852
	DEMO (w/ flow)	29.86	846
4	DEMO	27.49	1537
	CCNY RGBD		536

Table 4.1: Datasets description

4.2.1 Time performance

A comparison is done between the three frameworks running on the laptop and the DEMO and CCNY RGBD on the Odroid-U3, excluding the RGBD SLAM in this last one since its rate was so low that it was not possible to evaluate its performance (registered rates below 0.01 Hz with the established test conditions). It was used dataset 1, and by inclusion, dataset 3, to test the performance of the algorithms in terms of transformation publishing rate to be forwarded to the FCU. All the frameworks were launched and tested having also the MAVROS node working and publishing/subscribing to data. This node, in both laptop and Odroid-U3 tests, works between the 28 and the 30 Hz.

Laptop tests performance

The following results come from the tests were made on the laptop ROS environment. It is presented the plot curves for the working rates of the three localization frameworks, a median of the each rate in dashed lines and the difference between the three rates on a dashed red line. Results can be checked on Figure 4.3. As we can see, the DEMO framework offers the better performance in terms of working rate,

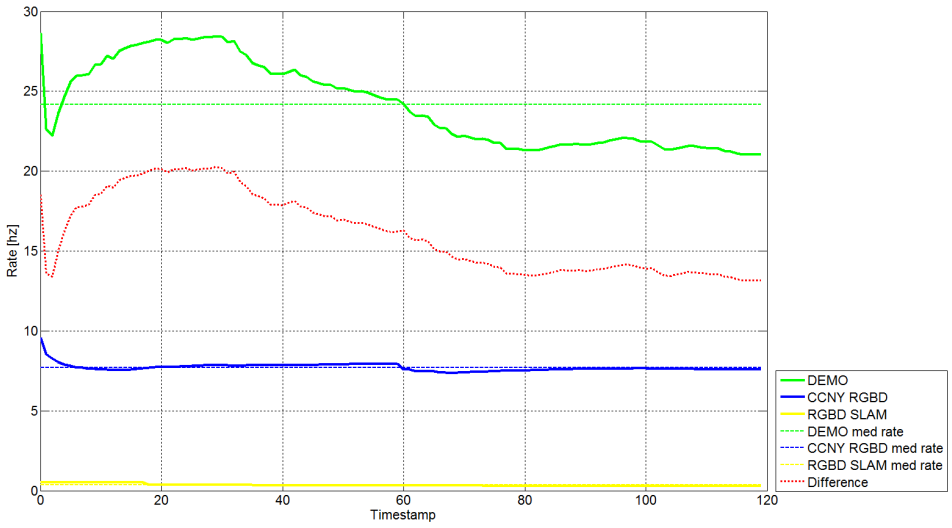


Figure 4.3: Localization frameworks time performance on the laptop

given that this framework only offers an odometry computation, different from the SLAM frameworks, which also have to compute the mapping of keyframes and loop closure. In the case of the RGBD SLAM framework, it also adds a graph optimization layer, which also reduces the working rate.

RGBD SLAM rates are related to the way the transformations between the world frame and the camera frame are being published. The code is not well optimized to give a stable high publishing of the transforms, even if in the GUI, it can be checked that the transforms appearance occur in an acceptable way. The fact that this rates are so low have a great negative impact on sending the estimated localization to the FCU, as it will be analysed on 4.2.2.

The quantified mean rate values are presented in table 4.2.

Framework	Mean Rate [hz]
DEMO	24.159
CCNY RGBD	7.708
RGBDSLAM	0.347

Table 4.2: Localization frameworks mean working rates on the laptop

Odroid-U3 tests performance

The following results come from the tests were made on the Odroid-U3 ROS environment. As it works on a ARM architecture system, it is normal to get lower rates than in the case of the laptop. The same plot structure as in the laptop case is used. The plot can be checked in Figure 4.4. As stated, it was

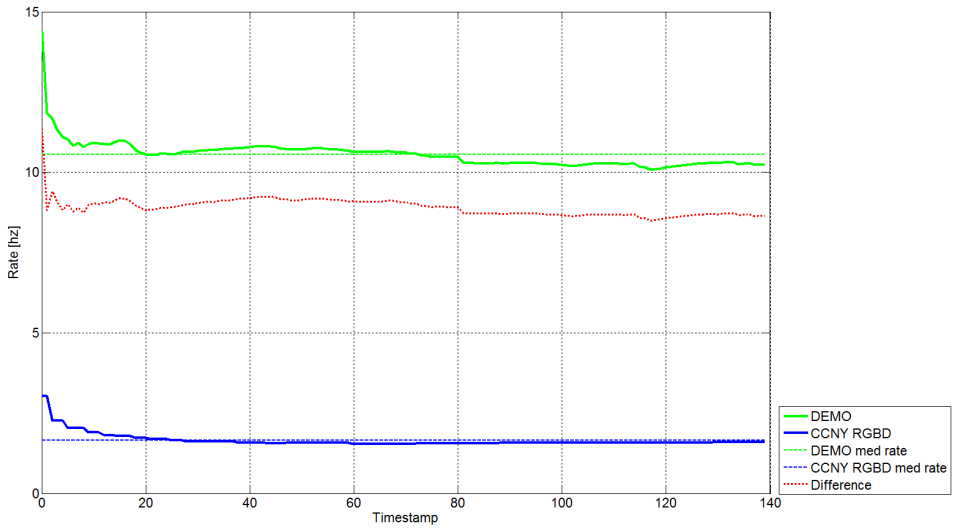


Figure 4.4: Localization frameworks time performance on the Odroid-U3

not possible to evaluate the RGBD SLAM working rate performance, as it gives too low rate values that the monitoring node running on ROS cannot estimate, as it is below the 0.01 Hz of transform publishing rate. But regarding the values of the other framework, the DEMO framework once again offers the best performance in terms of time, with a rate almost ten times faster than the CCNY SLAM framework.

The quantified mean rate values are presented in table 4.3.

Framework	Mean Rate [hz]
DEMO	10.559
CCNY RGBD	1.649
RGBDSLAM	-

Table 4.3: Localization frameworks mean working rates on the Odroid-U3

So to compare the working rates of the localization frameworks running on laptop against them running on the Odroid-U3 platform, table 4.4 is used to quantify the values. As it can be seen, there is a reduction of almost two thirds on the rate performance from the laptop to the Odroid-U3. Still though, the working rate of DEMO framework is enough for being considerable ”usable” by the FCU position estimator, which can work with sensor update rates from 4-5 Hz.

Mean Rate [hz]		
Framework	Laptop	Odroid-U3
DEMO	24.159	10.559
CCNY RGBD	7.708	1.649
RGBDSLAM	0.347	-

Table 4.4: Localization frameworks time performance evaluation on the laptop and on the Odroid-U3

4.2.2 Localization performance

All the plots represent in Appendix B are a 3D spatial representation of the path and vehicle frames recovered in the datasets. The more or less number of frames of these depends on the transforms publication rate of the respective frameworks, which off course depends on the algorithm processing times. The frames represent, not also the vehicle position, but also the orientation of it. It is identified by a RGB colour-map, meaning blue for Z, green for Y and red for X. The position given by the marker framework also uses a frame axis representation, but with a grey colour for distinction from the localization framework on the MAV.

Dataset 1

As a reminder, this dataset has the purpose of comparing the different localization framework between each other and against the marker ”ground truth” system without fusing the estimate into the FCU position estimator.

The first framework to be evaluated is DEMO. With a working rate on near 30 Hz, it is indeed fast compared to the SLAM frameworks. Being fast, also allows to recover most frame pose estimates per second, giving a more concise and easy to read path, not only to the MAV, which needs an accurate and relatively fast position updates, but also for a person to evaluate.

Figure B.2 presents the main the plot of the position estimated by DEMO.

Figure B.3 has three subplots, with different perspectives of the plot, for better interpretation.

From the analysis of the given plots, it can be stated that there is a slight error associated with the

estimated pose, mainly regarding the Z position estimate, which has an approximate offset error of 21 cm. The X has a good approximation to the real path, while Y suffered a deviation in the way back to the starting position.

The overall position and orientation mean absolute errors in all the path can be verified in Table 4.5. The calculated error for the position is in meters, while the orientation error is in degrees. The offset

	Mean absolute error
Position	[m]
X	-0.09
Y	-0.04
Z	0.21
Orientation	[deg]
Roll	-21.3
Pitch	1.0
Yaw	11.9

Table 4.5: DEMO without sensor fusion VS Ground truth mean errors

error on Z can be considered normal, as the algorithm starts the absolute 0 at the camera height. That means, given that the camera is positioned 12 cm above ground level, that means there is an offset of 12 cm, which leads to only 8 cm error, which is inside the range of values of the mean errors of X and Y.

Next to be evaluated is CCNY RGBD estimation without sensor fusion on-board the FCU. Compared to DEMO, it is slower, given that it needs a landmark matching of the different frames in order to establish a landmark graph a built a map at the same time it runs a visual odometry algorithm, while DEMO is only a visual odometry framework by itself. The 3D main plot with the comparison with ground truth is done in figure B.4. The different plot perspectives is presented in Figure B.5.

Again, the overall position and orientation mean absolute errors in all the path can be verified in Table 4.6. The calculated error for the position is in meters, while the orientation error is in degrees.

	Mean absolute error
Position	[m]
X	-0.04
Y	-0.15
Z	0.02
Orientation	[deg]
Roll	3.5
Pitch	4.8
Yaw	-70.3

Table 4.6: CCNY RGBD without sensor fusion VS Ground truth mean errors

The tabled absolute error may lead to a misjudge of the true performance of the algorithm. As the absolute error is presented as a mean of all the samples, if most of the samples are taken when the position has less error, it may lead to evaluate the algorithm as having good performance, which, considering what is seen in the plots, is not the case. For example, as most of the initial samples of both ground truth and SLAM are taken in the beginning, before the quadrotor does initiates motion in X

and Y, it lead that Z got an error of 2 cm only, when checking the plots, it can be seen an error of more or less 15 cm. Another example but were the error is oversampled is in the case of yaw. This may be related to a initial error when the SLAM algorithm started, which was then corrected when more samples were taken.

Next framework to evaluate is RGBD SLAM. Even that this optimization leads to a good pose and mapping, it does not go well on the case of transformation publishing, as it was seen in the time performance evaluation. Publishing the required transformations so that the MAV can know ”where it is” at a rate less than 1 Hz leads to an erroneous evaluation of the performance of the localization, as the number of samples are too low. The same thing does not happen when presenting the frames and path on RGBD SLAM GUI (Fig, 4.5), where the points and path are presented in a more acceptable way, which means that RGBD SLAM current code is not optimized for transformation publishing. Just to point out the issue being judged, Figure B.6 plot compares the estimated localization of RGBD SLAM against the marker-based ground truth, with its different subplots in Figure B.7.

So, the only way this framework can be comparable to other algorithms is judging the plotted ground truth against the path and presented frames locations on RGBD SLAM GUI, as in Fig. 4.5. As it is not scientifically precise, it will not be considered as reliable. So RGBD SLAM, as is, cannot be considered usable for this type of systems. Some tweaks on code though possibly may get it to work faster so it can be usable, but that can be point out for future work.

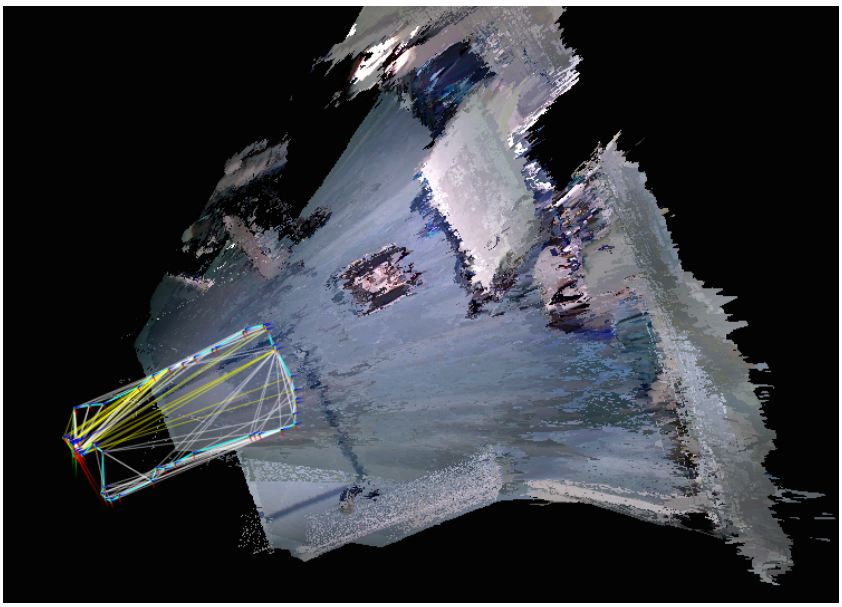


Figure 4.5: RGBD SLAM estimated path and map presented on GUI

So to conclude the comparison, table 4.7 collects all the mean absolute errors of the estimators, relative to the marker-based ground truth. RGBD SLAM has idle values, as it is not possible to compare the estimations.

	DEMO	CCNY RGBD	RGBD SLAM
Position[m]			
X	-0.04	-0.04	-
Y	-0.09	-0.15	-
Z	0.21	0.02	-
Orientation[deg]			
Roll	-21.3	3.5	-
Pitch	1.0	4.8	-
Yaw	11.9	-70.3	-

Table 4.7: Comparison between each localization framework without sensor fusion against ground truth

Dataset 2

This dataset includes, as stated, fusing other sensor sources, as the IMU. Following the evaluation structure of the first dataset, the first framework to be tested is DEMO. The way it is done is simple: Firstly, it is made a comparison of the estimated pose without sensor fusion against the estimated pose with sensor fusion; then, it is made a comparison of the estimated pose with sensor fusion against the marker-based ground truth.

Figure B.8 shows the first stated comparison. The frames in grey represent the framework without sensor fusion while the coloured ones represent the framework with post sensor fusion.

The results seem to be accurately the same, as it can be checked on table 4.8. The only difference that exists is in roll angle, where in the case of the estimated pose after sensor fusion, the roll is considered to be the correct estimated value, given the IMU fusion in the attitude estimator. The reason the estimation after sensor fusion can be considered a good estimate is that in most part of the path, the MAV stayed in a horizontal position, without tilting.

	Mean absolute error
Position	[m]
X	6.5×10^{-4}
Y	-3.5×10^{-3}
Z	3.0×10^{-3}
Orientation	[deg]
Roll	24.35
Pitch	2.61
Yaw	0.64

Table 4.8: DEMO with sensor fusion VS DEMO without sensor fusion mean errors

This last conclusion can also be confirmed in the comparison with the marker-based ground truth in Figure B.9, with respective perspectives in Figure B.10. Again, the Z offset maybe explained by the fact that the starting point of the pose estimation of the algorithm is above 20 cm above the ground. Still, there is a slight error on Y estimation and the offset on Z, when the MAV was landed, means that the sonar was not used on that time for offset correction, probably because the “quality” variable was not high enough. The resulting mean absolute error can be checked in table 4.9.

Next framework to be evaluated is CCNY RGBD. Figure B.11 shows the first stated comparison, with the mean absolute error tabled in table 4.10. Again, even though it is minimal, the fusion is essential

	Mean absolute error
Position	[m]
X	0.08
Y	0.16
Z	0.29
Orientation	[deg]
Roll	-1.58
Pitch	2.84
Yaw	-0.16

Table 4.9: DEMO with sensor fusion VS Ground truth mean errors

to mainly correct the attitude estimation, as the absolute position is given by the vision-based estimator running on the computer framework.

	Mean absolute error
Position	[m]
X	0.02
Y	-0.02
Z	0.02
Orientation	[deg]
Roll	-1.64
Pitch	-8.44
Yaw	-4.63

Table 4.10: CCNY RGBD with sensor fusion VS CCNY RGBD without sensor fusion mean errors

The next plots relate to the comparison between the ground truth and CCNY RGBD localization estimation. Figure B.12 identifies the main plot, with the subplot perspectives in Figure B.13.

The mean absolute error can be checked in table 4.11. As can be seen, CCNY RGBD has major estimation errors, and even though the estimation is fused on the FCU filters, they cannot properly correct position if there are no other position sources and vision estimate is considered to be the absolute position source. The only correction that can be done, in the case that there is no other source that can be used for position estimation (as GPS or the Optical Flow module, which is also tested in one of the datasets), is in the attitude estimation, given that the IMU gives proper readings that are used on the on-board attitude estimator.

	Mean absolute error
Position	[m]
X	-2.52
Y	0.16
Z	0.27
Orientation	[deg]
Roll	-2.26
Pitch	-5.95
Yaw	-1.10

Table 4.11: CCNY RGBD with sensor fusion VS Ground truth mean errors

The last test case should be RGBD SLAM, but as stated, the rate in what the framework works does not allow to properly send the estimates so they can be used by the FCU. The tests revealed that, as the

complementary filter is waiting for the vision estimate, it starts to mess the estimation when the received position estimates are received with a difference of 10 or more seconds between each other, which is not feasible to a system that needs fast processed estimates so to work properly. So, as a analysis to the RGBD SLAM framework in general, it cannot be considered usable to this kind of dynamic systems as is, but can at least be used to properly construct maps in offline processing environments.

As to conclude the comparison between the frameworks, table 4.12 collects all the mean absolute errors of the estimations after being filtered on the FCU, relative to the marker-based ground truth. RGBD SLAM has idle values again, as it is not possible to compare the estimations.

	DEMO	CCNY RGBD	RGBD SLAM
Position [m]			
X	0.08	-2.52	-
Y	0.16	0.16	-
Z	0.29	0.27	-
Orientation [deg]			
Roll	-1.58	-2.26	-
Pitch	2.84	-5.95	-
Yaw	-0.16	-1.10	-

Table 4.12: Comparison between each localization framework with sensor fusion against ground truth

Dataset 3

The following results come from the tests made outdoor. As the tests itself are made under different conditions from the ones made indoor, this may result in unexpected behaviours, given, for example, the light conditions where the tests were made, even though there was a worry in doing the tests in later afternoon and over building shadow, so that the IR light could not affect the measurements. Still though, this test will turn out to be somehow inconclusive regarding the benefits of using a pointing down camera for velocity measurements given the current filtering that is being used on the FCU. As said before, the tests were made using the DEMO framework, as it gives the best results of the three used frameworks.

Figure B.18 (a) and B.18 (b) respectively represent a comparison between the estimated localization using the optical flow module on the sensor fusion against the ground truth, and the estimated localization without using the optical flow module on the sensor fusion against the ground truth. Table 4.13 represent the mean absolute error of both tests. As can be seen, there is no useful conclusion that can be taken from this tests, given that both diverge from the actual measurements from the marker position. The fusion with the optical flow though seems to give even worse results, despite having a much lower X estimate error (mean absolute errors may result on a misjudge of the performance, and that is way graphical analysis is indeed necessary).

A justification for this can be that the measurements from the vision system may not be synchronized in time with the measurements from the optical flow module, and since they both concur for giving an estimate of the horizontal position, that may result in errors. Also, another justification can be that the flow “quality” was not stable, given that, despite the tests were made still on day time, it still may require proper lens calibration to get proper flow measurements at the height the tests were made.

	no optical flow	w/ optical flow
Position [m]		
X	0.50	0.08
Y	-0.28	0.32
Z	0.43	0.51
Orientation [deg]		
Roll	-3.20	-5.15
Pitch	-0.90	-1.14
Yaw	47.48	54.39

Table 4.13: Comparison between localization estimation mean absolute error with and without fusion of optical flow measurements

Dataset 4

Dataset 4 uses exactly the same data as the dataset 1, but it was divided into a separate dataset in order to establish a difference between the algorithms being run on the laptop and on the Odroid-U3. As stated, the primary objective is to evaluate the localization estimation performance of the frameworks running on a lower processing unit, compared to the common architecture of the standard computers.

The comparison follow the same test rules as the ones applied on the evaluation of dataset 1: it is used a 3D plot to compare the estimated localization given by the different frameworks versus the estimation given by the marker-based ground truth, which is considered to give the most accurate localization.

The first framework to be evaluated is DEMO. Again, it is used a 3D plot to show the different frames and the taken path. Figure B.14 shows the plot. The different plot perspectives are presented on Figure B.15.

Again, there is a slight offset on the height estimate, which is given to the stated fact that it is not added a transform correction from the camera frame, from where the algorithm starts the pose and motion estimation, and the ground basis of the local origin. Still though, there is an error offset that must be taken into account. In terms of horizontal positioning, the estimates are quite acceptable, as it can be checked on table 4.14. Also to notice that there were some errors relative to the orientation, which as seen before, can be corrected in the attitude estimator of the FCU.

	Mean absolute error
Position	[m]
X	-0.08
Y	0.03
Z	0.24
Orientation	[deg]
Roll	-20.23
Pitch	-0.18
Yaw	18.22

Table 4.14: DEMO without sensor fusion on the Odroid VS Ground truth mean errors

The second and last framework to be evaluated is CCNY RGBD. Figure B.16 shows the main plot and Figure B.17 shows the perspectives.

Again, given that the updates of the feature detector are slow, given the lower processing power, it leads to features being lost track of, which may result on a bad visual odometry estimations, as the ones that can be checked on the plot, and quantified in table 4.15. A thing to notice is that, even though the mean position errors are somehow low, the leaps on the estimations of the pose given by the visual odometry cannot are unusable by a platform as a MAV, which will for sure result on a uncontrollable behaviour of the MAV and a crash.

	Mean absolute error
Position	[m]
X	0.05
Y	-0.16
Z	0.20
Orientation	[deg]
Roll	-3.52
Pitch	4.35
Yaw	-134.82

Table 4.15: CCNY RGBD without sensor fusion on the Odroid VS Ground truth mean errors

4.3 Mapping performance

The fifth dataset, as stated, was used to validate the map creation on the Moveit! framework. For that, it is used a stream of continuous data that is loaded into the framework, in which the Octomap Moveit! plugin actuates so to transform the received point cloud into a usable occupation grid for motion planning.

So to validate the map being created, it is compared 3 different images, with 3 different resolutions for the voxels size : 5 cm (Fig. 4.7 (a)), 10 cm (Fig. 4.7 (b)) and 15 cm (Fig. 4.7 (c)), against the raw RGB image (Fig. 4.6 (a)) and the depth image (Fig. 4.6 (b)).

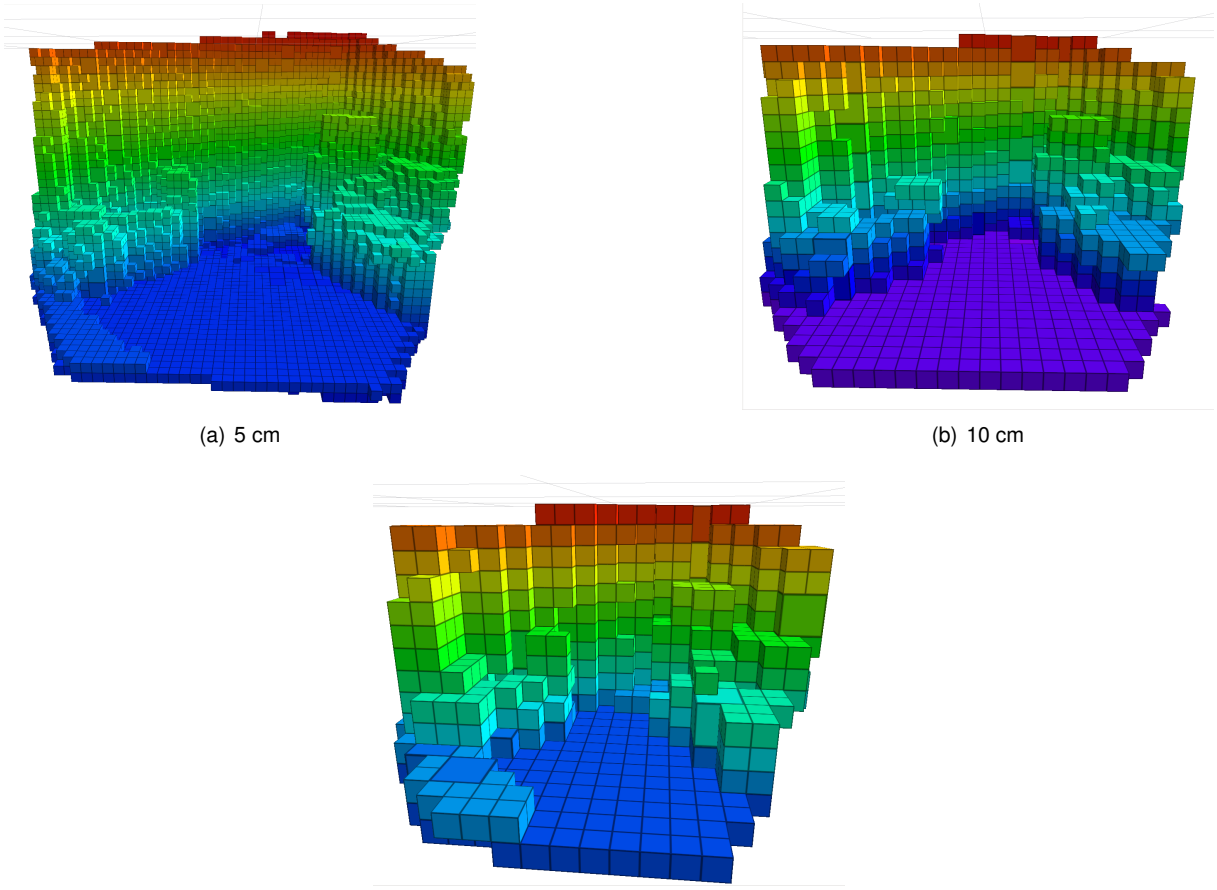
The localization framework used is DEMO, as it was confirmed by the previous results as the most accurate and faster estimator from the tested frameworks.

(a) RGB image

(b) Depth image

Figure 4.6: Original image stream

As can be seen, the map obtained in the framework is reliable in the three resolutions. The only



(a) 5 cm

(b) 10 cm

(c) 15 cm

Figure 4.7: Different occupation grid resolutions for the same image stream

handicap is related to the rate of the map and the accumulated data. Using resolutions as 5 cm or less result on a slower map update and in a need of higher processing and memory allocation for accumulate the continuous data construction.

In the other hand, lower resolutions as 10 cm or 15 cm are faster and better to store so to construct the map. 10 cm can be considered as the near optimal resolution, as its size is smaller compared to the vehicle size and the kind of manoeuvres pretended for the vehicle. 15 cm resolution is the fastest to process and store, but then may not have enough resolution for obtain a certain precision of obstacle free paths by the motion planner.

Chapter 5

Conclusions

As to conclude this work, it matters to establish some important conclusions from the framework tests done:

- From all the localization modules tests, DEMO seemed to offer the best results in terms of localization estimation and time performance with the its raw implementation, i.e. without any additional code changing or algorithm parameter adaptations as the ones done on CCNY RGBD and RGBD SLAM.
- The SLAM modules may need some code and parameter adjustments so to properly work under the conditions established of this kind of systems: estimations at high rates and precision of the estimate. Even though giving a fast estimate, CCNY RGBD, used with the established parameters, needs to be optimized so it can give a less nosier a more precise odometry estimation. In the case of RGBD SLAM, adjustments are also needed, which may also mean to deactivate the optimization step so to give faster odometry estimations.
- ARM platforms, as in the Odroid case, may act alone in the processing of the visual odometry algorithms and pose estimation as the case of DEMO, but it is not satisfatory as an integrated framework with both mapping and navigation, as extra processing power is indeed needed.
- The tests done with the localization modules do not invalidate ones as usable for navigation purposes on MAV's or validate positively them to that purpose also, as the used parameters for each platform and the test conditions may differ and may also have to be optimized. A thing of notice though is that all the trajectories were made with a controlled movement, i.e. handheld by a person. In a real test conditions, with a flying vehicle, the results may drastically change given the dynamic of the vehicle. It is important to notice that, for example, the yaw orientation of the vehicle was never changed, and in the case of changing it may alter the localization estimation.
- The current filter implementation is important for orientation correction, but not so for position, as no other localization sources are being fed to the filter. An exception goes to the optical flow module, but the current tests, as stated, were inconclusive, given the limitations of the testing area.

- The MoveIt! framework functions acceptably when receiving the localization estimate and in the processing of the planning scene map, which means that the next step would be advance to test the path planning solutions. Still though, other kind of localization modules may also be used and tested, so to establish which ones gives the best estimate, or use the ones tested on this thesis and tweak them in a way they can give better results. An important thing to notice is that the visual odometry algorithms just work acceptably in static environments, i.e. without changes in the environment caused by dynamic obstacles. That means it is required a filtering algorithm that can detect dynamic obstacles and adjust the localization given the presence of dynamic entities.
- As seen, the best results of the algorithms were obtained in the laptop environment. But that fact, off course, is a limitation to the proper work of the MAV, as it has to be tethered to the laptop and it also removes its autonomus behaviour. That means a higher processing unit is required to get all the modules working onboard the MAV.

5.1 Achievements

It was created an integrated architecture for testing localization, mapping and navigation modules, based on open-source and inexpensive hardware and available state-of-the-art frameworks. It was also possible to partially test some localization modules, under certain test conditions and algorithm parameters. The mapping capability of the framework was also tested and approved. The obtained framework is navigation ready, needing only some adjustments and testing.

5.2 Future Work

As I will continue to develop work in this area, there are already some ideas to advance into:

- Test some more visual odometry and SLAM frameworks as SVO [19] or LSD-SLAM [15], to check its usability in a system like this. Also, improve the usage of the current frameworks by adjusting its parameters and/or use the support of GPU processing.
- Use an external sensor fusion framework as ETHZ-ASL MSF [34], running on a partner computer of the MAV so to obtain faster update rates than the ones that the FCU can obtain by itself and using and allowing multi sensor fusion using an Extended Kalman Filter;
- After acquiring an NVIDIA Jetson TK1, the idea is to abandon the Odroid-U3 usage and advance to this super-computing platform, adapting the localization frameworks to use the CUDA cores on-board and get extreme accurate and fast localization estimates;
- Construct a customized MAV that is able to accommodate multiple sensors, as an RGB-D camera, a High resolution camera, a PX4Flow kit, a LIDAR-Lite, a low-cost 360 degrees RPLIDAR and GPS. The idea is to advance to a hybrid system which is able to do both indoor and outdoor navigation, as also to transpose from one to the other.

Bibliography

[1] M. Bennewitz C. Stachniss W. Burgard A. Hornung, K. M. Wurm. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomus Robots*, 34(3):189–206, 2013.

[2] D. Haehnel A. Segal and S. Thrun. Generalized-icp. *Robotics: Science and Systems*, 2009.

[3] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. Technical report, Google Inc. and University of Washington.

[4] A. Bachrach, A. de Winter, R. He, G. Hemann, S. Prentice, and N. Roy. Range - robust autonomous navigation in gps-denied environments. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1096–1097, May 2010.

[5] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy. Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments. *The International Journal of Robotics Research*, 31(11):1320–1343, September 2012.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (SURF). *Computer vision and image journal*, page 2564–2571, September 2008.

[7] T. Bresciani. Modelling, identification and control of a quadrotor helicopter. Master's thesis, Lund University, 2008.

[8] T. Carreira. Quadcopter automatic landing on a docking station. Master's thesis, IST, October 2013.

[9] I. Şucan, , and L. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. *Algorithmic Foundation of Robotics VIII, Springer Tracts in Advanced Robotics*, 57:449–464, 2010.

[10] Wu D., Johnson E., Dallaert F., Kaess M., and Chowdhary G. Autonomous flight in gps-denied environments using monocular vision and inertial sensors. *AIAA Journal of Aerospace Computing, Information, and Communication*, August 2012.

[11] R. D'Angelo and R. Leven. Design of an autonomous quadrotor uav for urban search and rescue. Master's thesis, Worchester Polytechnic Institute, April 2011.

[12] H. Dias. Controlo de formações de veículos aéreos não tripulados. Master's thesis, IST, April 2012.

[13] D. Eberli, D. Scaramuzza, S. Weiss, and R. Siegwart. Vision based position control for mavs using one single circular landmark. *Journal of Intelligent and Robotic Systems*, 61:495–512, 2011.

[14] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3d mapping with an rgb-d camera. *IEEE Transactions On Robotics*, 30(1), January 2012.

[15] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. *European Conference on Computer Vision*, 2014.

[16] A. Ortiz F. Bonin-Font and G. Oliver. Visual navigation for mobile robots: a survey. Technical report, Department of Mathematics and Computer Science, University of the Balearic Islands, Palma de Mallorca, Spain, 2005.

[17] T. Fernandes, M. Morais, N. Marques, J. Santos, and R. Pires. Quadrotor de navegação e exploração autónoma de ambientes fechados. Technical report, IST, December 2013.

[18] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[19] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. *International Conference on Robotics and Automation*, 2014.

[20] F. Fraundorfer, L. Heng, D. Honegger, G. Hee Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. Technical report, Computer Vision and Geometry Lab, ETH Zürich, Switzerland, 2012.

[21] D. Magree D. Wu A. Shein G. Chowdhary, E. Johnson. Gps-denied indoor and outdoor monocular vision aided navigation and control of unmanned aircraft. *IEEE Journal of Field Robotics*, January 2013.

[22] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. Technical report, Department of Computer Science, University of Freiburg, Freiburg, Germany, 2010.

[23] M. He, C. Ratanasawanya, M. Mehrandezh, and R. Paranjape. Uav pose estimation using posit algorithm. *International Journal of Digital Content Technology and its Applications*, 5(4), April 2011.

[24] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, page 2472–2477, 2011.

[25] Walter Higgins. A comparison of complementary and kalman filtering. *IEEE Thans. Aerospace and Electronic Systems*, AES-11(3), November 1975.

[26] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. *International Conference on Robotics and Automation (ICRA) - Karlsruhe, Germany*, 2013.

[27] L. Kavraki and M. Overmars P. Šestka, J. Latombe. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), August 1996.

[28] J. Kelly, S. Saripalli, and G. S. Sukhatme. Combined visual and inertial navigation for an unmanned aerial vehicle. Technical report, Robotic Embedded Systems Laboratory, University of Southern California, USA.

[29] C. Kerl. Odometry from rgb-d cameras for autonomous quadrocopters. Master's thesis, TUM University, Munich, Germany, November 2012.

[30] S. Ladha, D. Kumar, P. Bhalla, A. Jain, and R.K. Mittal. Use of lidar for obstacle avoidance by an autonomous aerial vehicle. *Aerial Robotics Competition Symposium*, 2011.

[31] S. M. Lavalle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. Technical report, Iowa State University, USA, and University of Tokyo, Japan, 2000.

[32] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[33] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.

[34] S. Lynen, M. Achtelik, M. Chli S. Weiss, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. *International Conference on Intelligent Robots and Systems*, 2013.

[35] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *Robotics Automation Magazine - IEEE*, 19(3):20–32, September 2012.

[36] N. Marques. Thesis datasets. `https://www.dropbox.com/sh/nup3id1tvz5gaz8/AABkvD4Gr3zqADzutTMv0ORIa?dl=0`. Online.

[37] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotor,. *Proceedings of the International Symposium on Experimental Robotics*, December 2010.

[38] Q. Michael, D. Mellinger, and V. Kumar. The grasp multiple micro-uav testbed. *Robotics and Automation Magazine*, 17(3):56–65, May 2010.

[39] M.Kaess, Ananth Ranganathan, and Frank Dellaert. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, page September, 2008.

[40] M. Muja and D. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 36(X), 2014.

[41] H. Park, W. Ra, and I. Whang. Robust complementary filter design and its application on gps/ins vertical channel design.

[42] A. G. Pinto, A. Moreira, P. G. Costa, and M. V. Correia. Revisiting lucas-kanade and horn-schunck. *Journal of Computer Engineering and Informatics*, 1(2):23–29, April 2013.

[43] S. Raza and W. Gueaieb. Intelligent flight control of an autonomous quadrotor. *Robotics - Motion Control*, 24, January 2010.

[44] E. Rublee, W. Garage, and M. Park. Orb : an efficient alternative to sift or surf. page 2564–2571, 2011.

[45] D. Scaramuzza and F. Fraundorfer. Visual odometry part i: The first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, December 2011.

[46] S. Scherer and Andreas Zell. Efficient onbard rgbd-slam for autonomous mavs. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[47] N. Serrano. Autonomous quadrotor unmanned aerial vehicle for culvert inspection. Master's thesis, MIT, USA, June 2011.

[48] A. Silva. Vision based pose computation from landmarks: an application to quadrotors. Master's thesis, IST, October 2011.

[49] M. Sobers, G. Chowdharyz, and E. N. Johnsonx. Indoor navigation for unmanned aerial vehicles. *AIAA Guidance, Navigation, and Control Conference. Chicago, IL*, pages 10 – 13, August 2009.

[50] J. Suhr. Kanade-lucas-tomasi (klt) feature tracker. Technical report, Computer Vision (EEE6503) Course , Yonsei Univ., 2009.

[51] N. Sünderhauf. *Robust Optimization for Simultaneous Localization and Mapping*. PhD thesis, Technischen Universität Chemnitz, February 2012.

[52] A. Tonioni. Study and implementation of algorithms for 3d reconstruction of the environment and the navigation of autonomous aircraft. Master's thesis, Alma Mater Studiorum - Computer Science Engineering Faculty - University of Bologne, October 2013.

[53] M. Warren and B. Upcroft. High altitude stereo visual odometry. Technical report, School of Electrical Engineering and Computer Science - Queensland University of Technology.

[54] J. Zhang, M. Kaess, and S. Singh. Real-time depth enhanced monocular odometry. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS). Chicago, IL*, September 2014.

Appendix A

Frame conversions

The NED system is preferred over ENU in many applications, because it bears a simple natural relationship to the conventional aviation body axes. But, since most of the ROS applications regard ground robots, the most common frame convention is using ENU. Also, both use a right-handed orthogonal system.

In NED, except at the poles, at each point on the earth you can define a local reference frame such that the X axis points north (forward), the Y axis points east (left), and the Z axis points down. In the other hand, ENU X axis points east (left), the Y axis points north (forward), and the Z axis points up. This convertion is only applicable in the case of the local ground frame, since in the case of the body-frame, only acceleration and angular rates matter. In that case, ENU body-frame will have negated Y and Z axis because of a 180 degrees invertion around X, which means that X axis points north (forward), the Y axis points east (left), and the Z axis points up. This conversions can be seen in Figure A.1.

In the case of body-frame coordinates, conversions are done the following way:

Body-fixed NED coordinates $\longrightarrow$ ROS ENU coordinates: $(x, y, z) \longrightarrow (x, -y, -z)$ for position and $(w, x, y, z) \longrightarrow (x, -y, -z, w)$ for orientation.

For local ground frame coordinates, as in Figure A.1, conversions are done the following way:

Local ground NED coordinates $\longrightarrow$ ROS ENU coordinates: $(x, y, z) \longrightarrow (y, x, -z)$ for position and $(w, x, y, z) \longrightarrow (y, x, -z, w)$ for orientation.

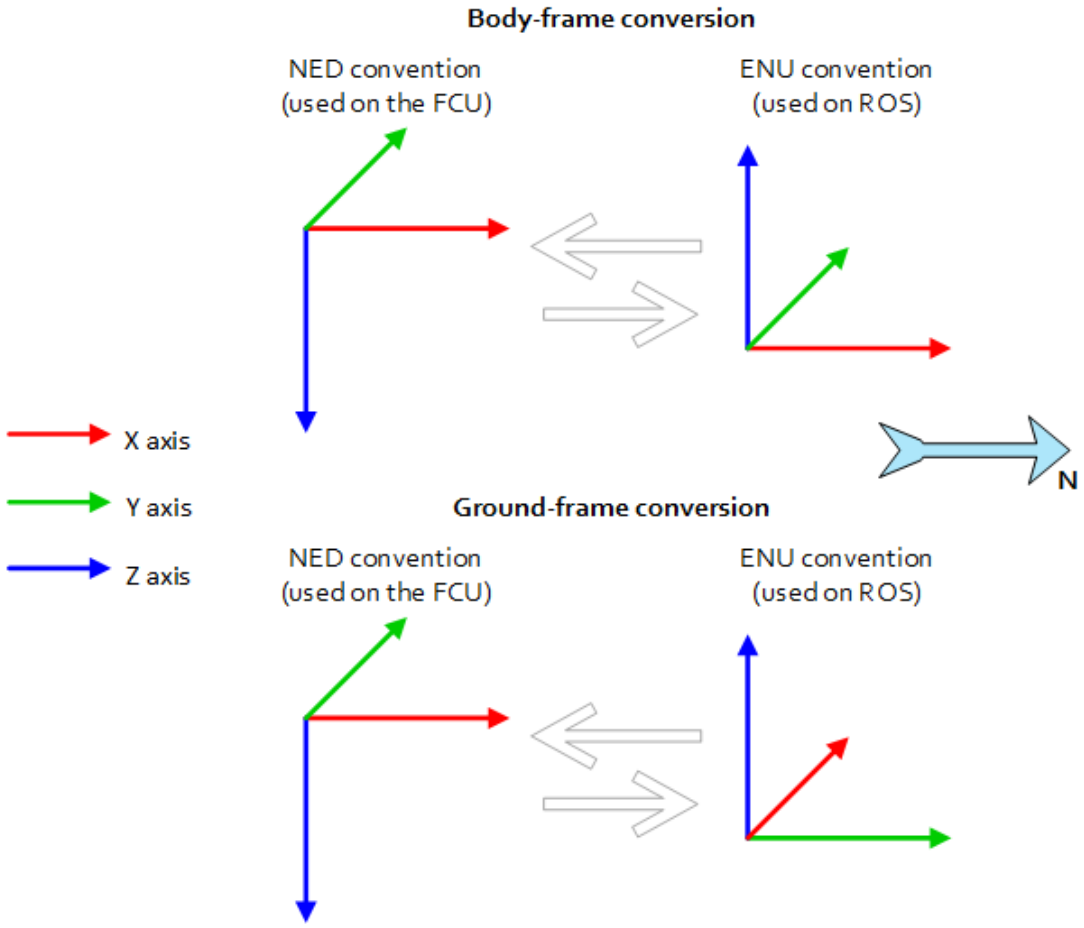
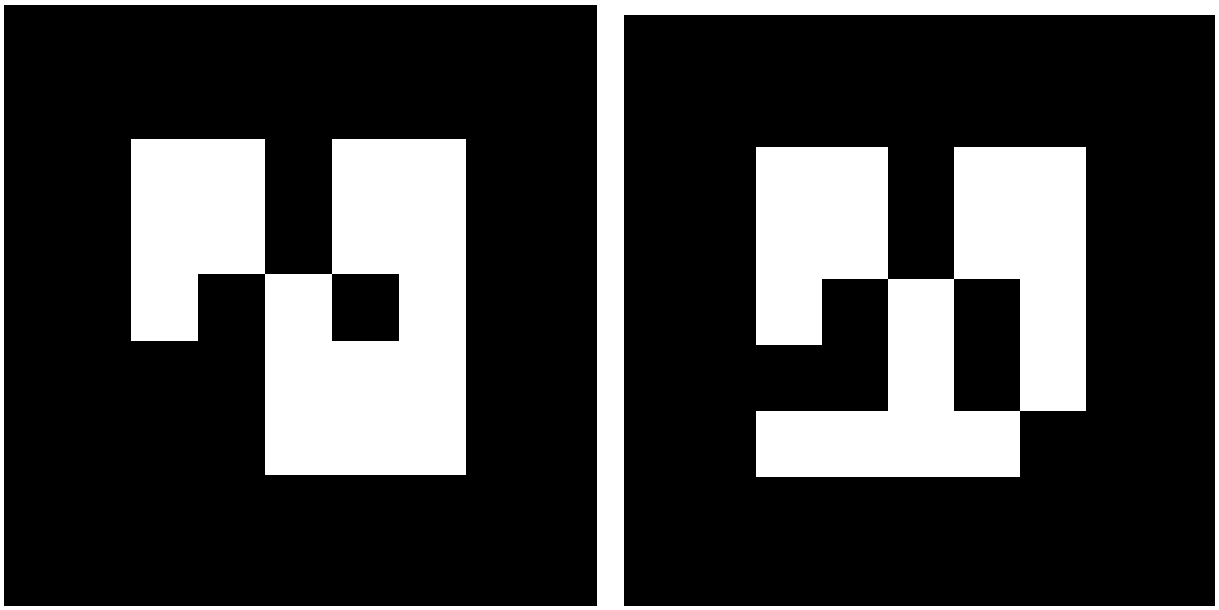


Figure A.1: Coordinate frame conversions for local ground frame

Appendix B

Results support images

B.1 Used AR markers



(a) AR Marker 6 - Local origin

(b) AR Marker 8 - Quadrotor MAV

Figure B.1: AR Markers used on the simulated Ground Truth system

B.2 Presented results plot perspectives

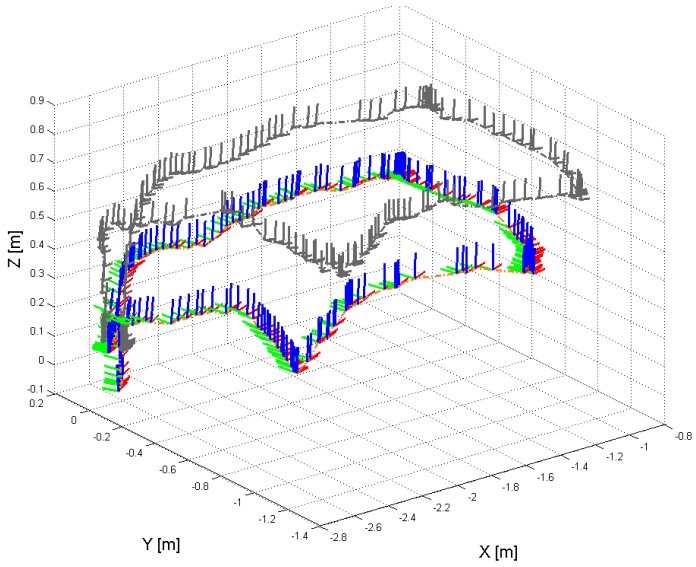
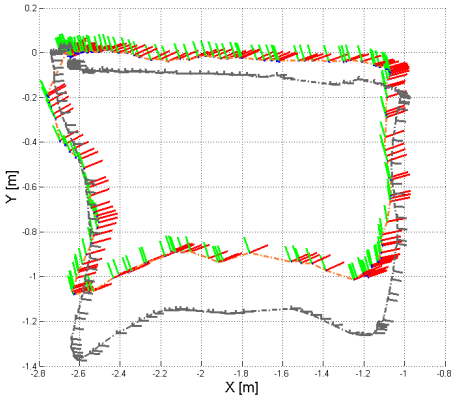
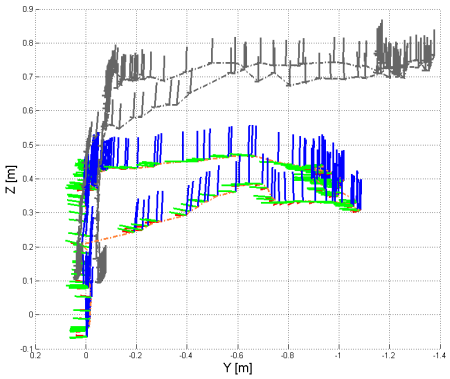


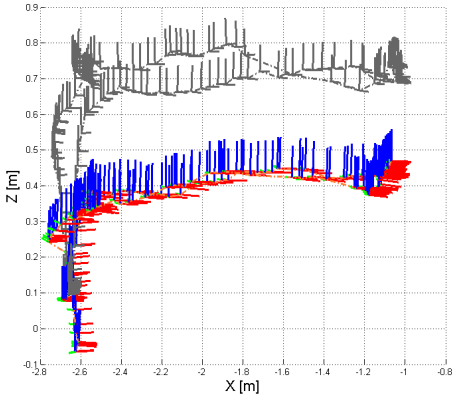
Figure B.2: DEMO estimated localization without fusion VS Ground truth



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.3: DEMO estimated localization without fusion VS Ground truth - perspectives

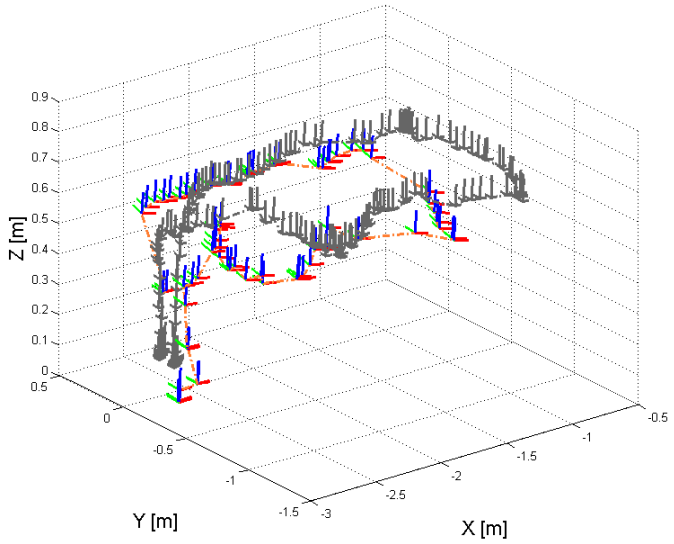
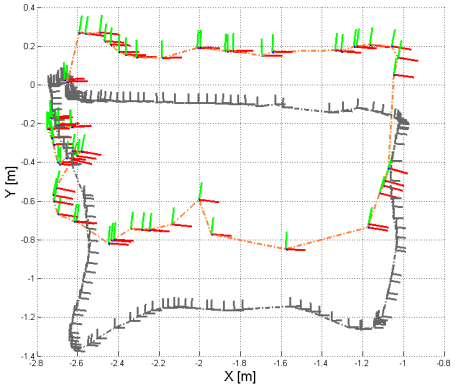
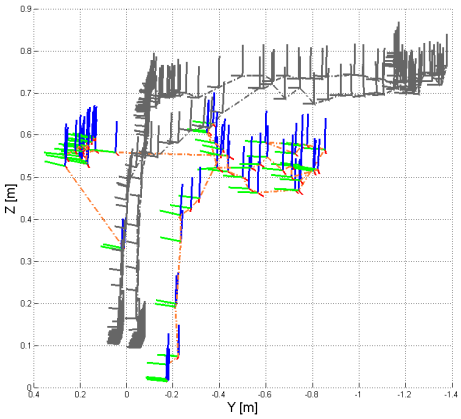


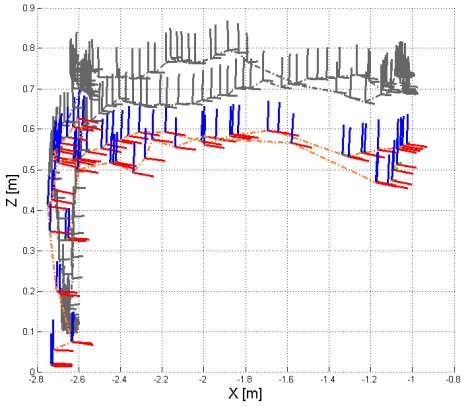
Figure B.4: CCNY estimated localization without fusion VS Ground truth



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.5: CCNY RGBD estimated localization without fusion VS Ground truth - perspectives

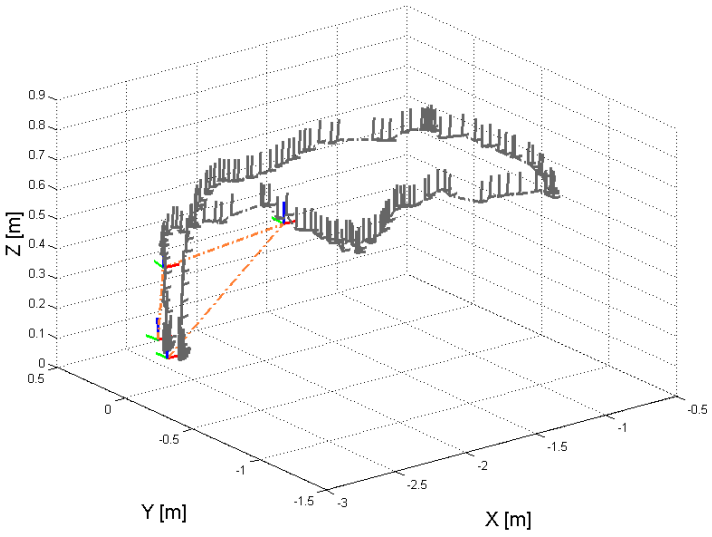
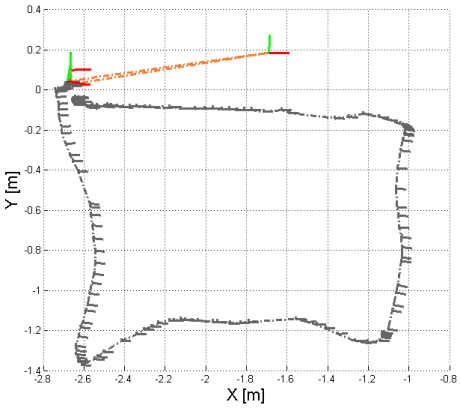
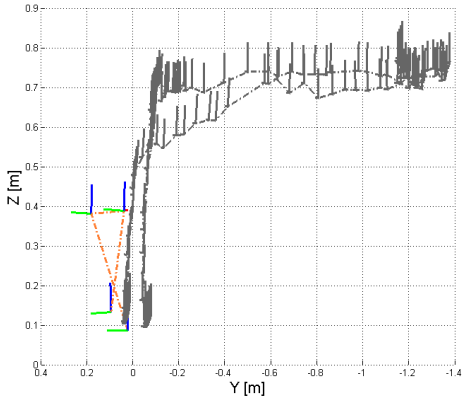


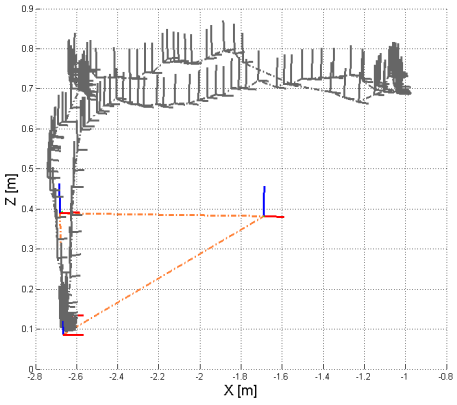
Figure B.6: RGBD SLAM estimated localization without fusion VS Ground truth



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.7: RGBD SLAM estimated localization without fusion VS Ground truth - perspectives

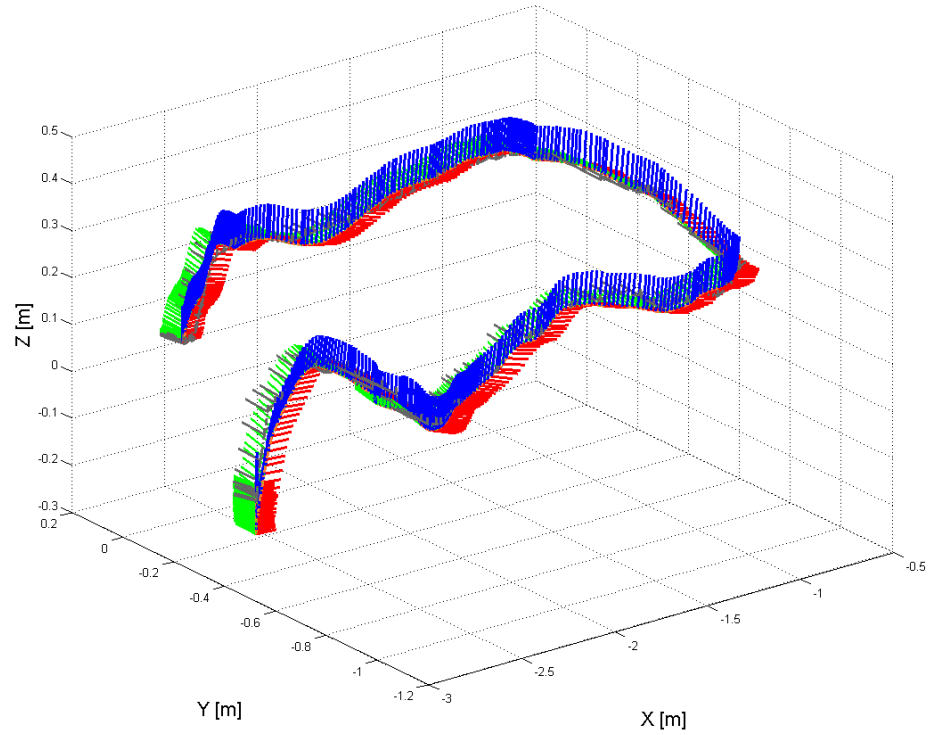


Figure B.8: DEMO estimated localization with fusion VS without fusion

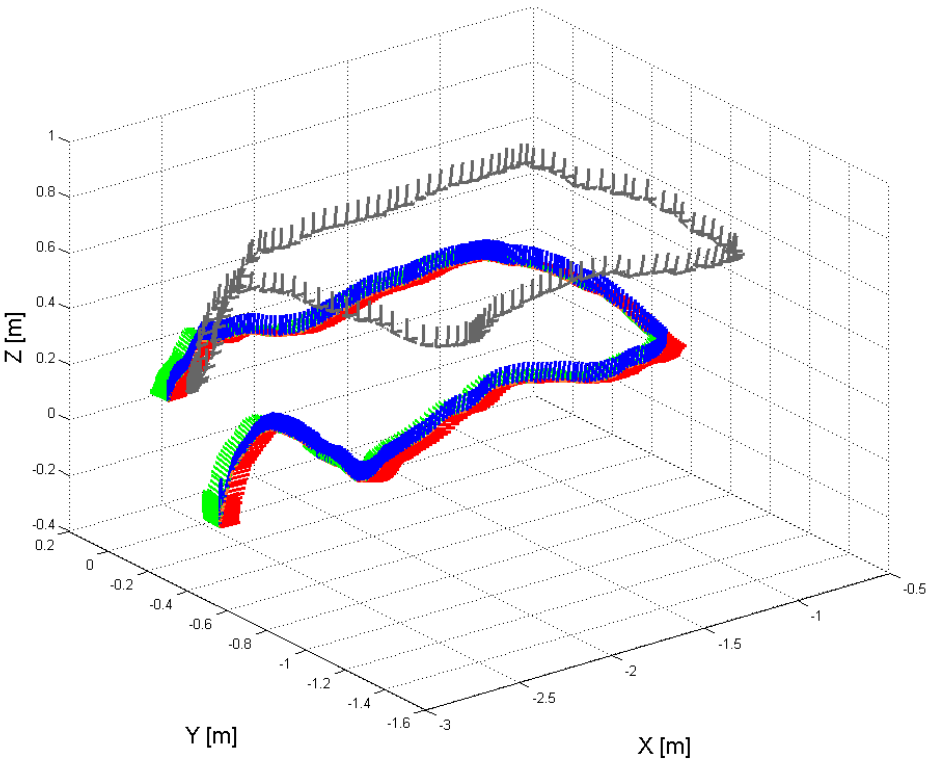
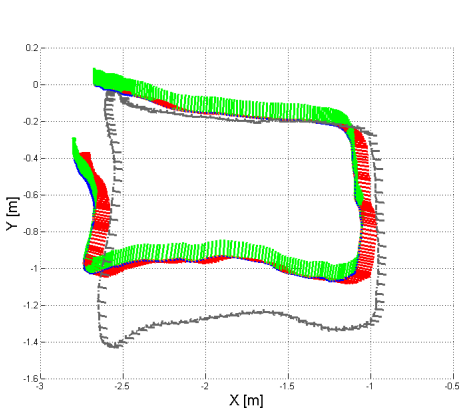
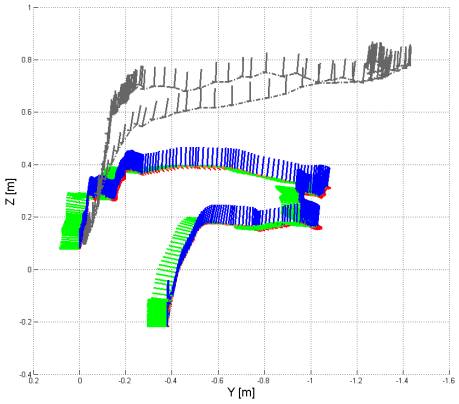


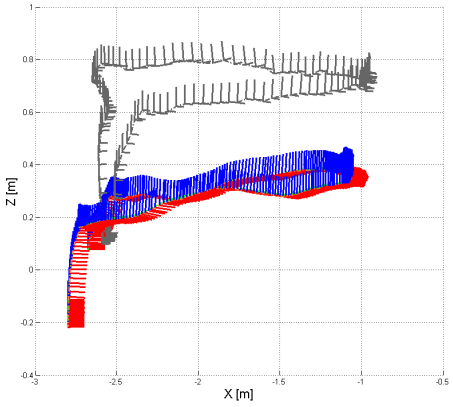
Figure B.9: DEMO estimated localization with fusion VS Ground truth



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.10: DEMO estimated localization with fusion VS Ground truth - perspectives

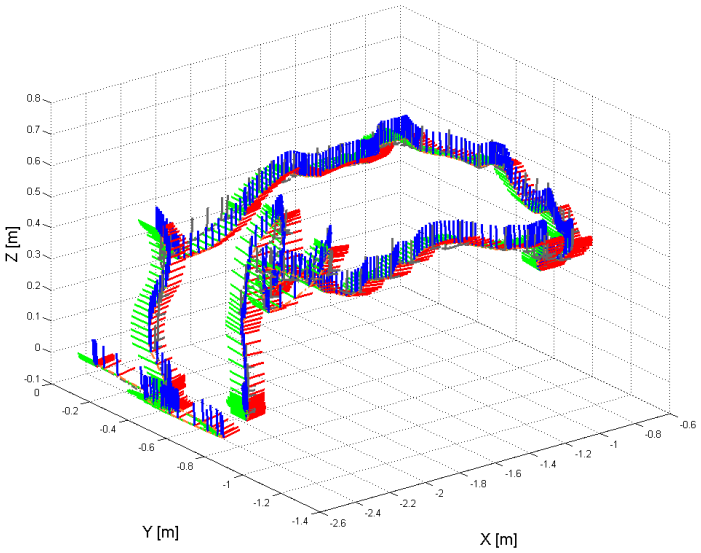


Figure B.11: CCNY RGBD estimated localization with fusion VS without fusion

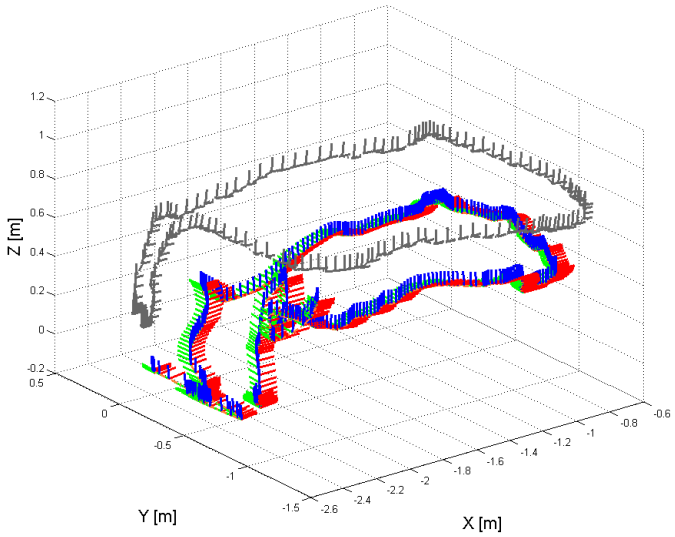
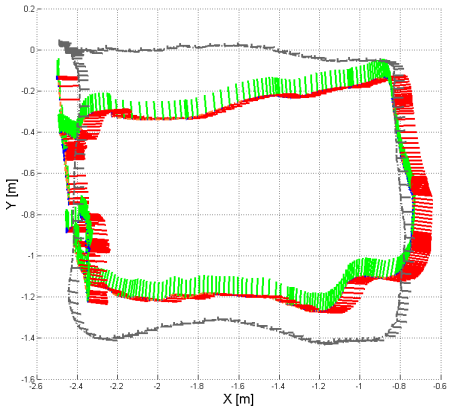
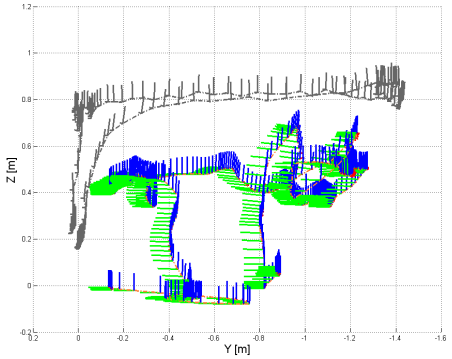


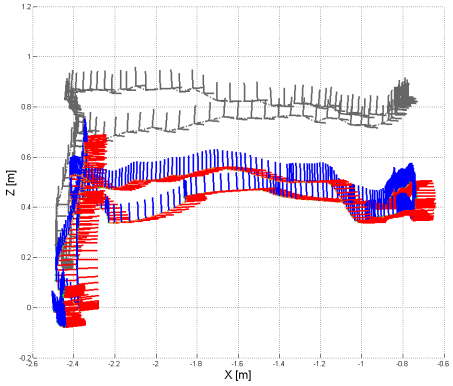
Figure B.12: CCNY RGBD estimated localization with fusion VS Ground truth



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.13: CCNY RGBD estimated localization with fusion VS Ground truth - perspectives

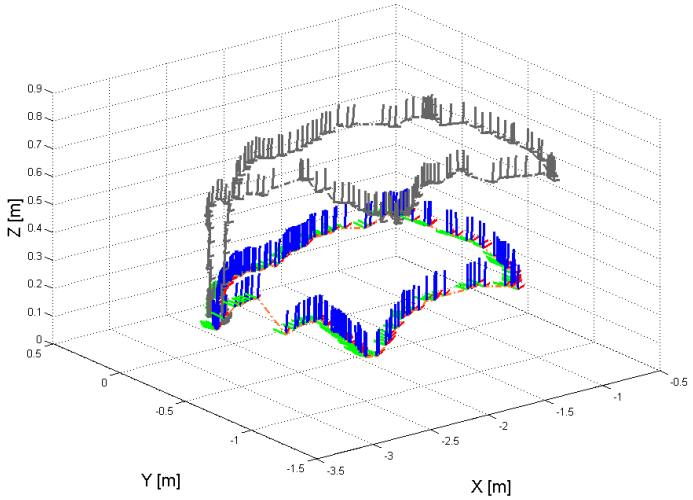
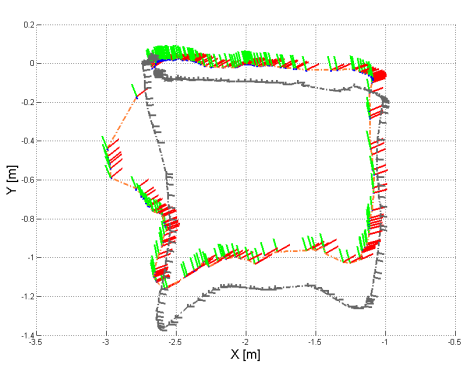
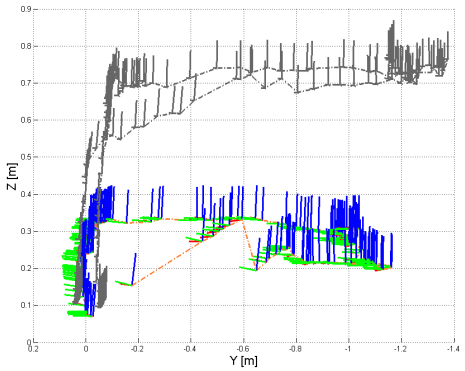


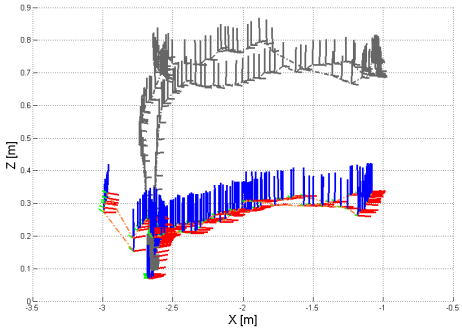
Figure B.14: DEMO estimated localization without sensor fusion on the Odroid



(a) XY Plot



(b) YZ Plot



(c) XZ Plot

Figure B.15: DEMO estimated localization without sensor fusion on the Odroid VS Ground truth - perspectives

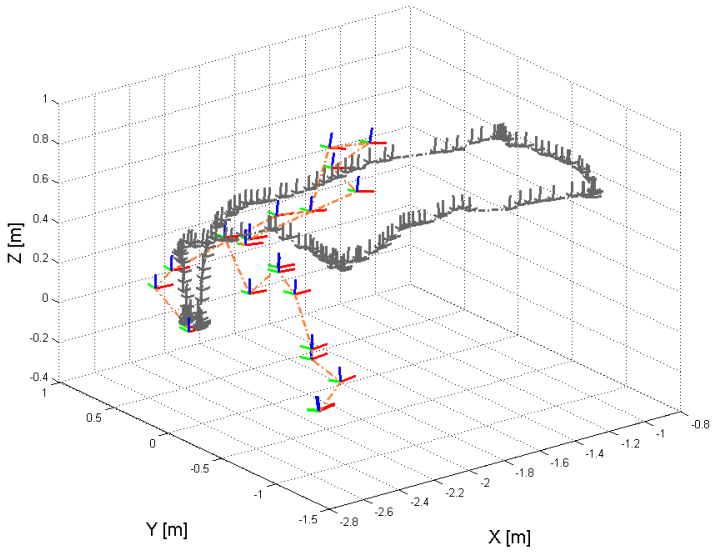
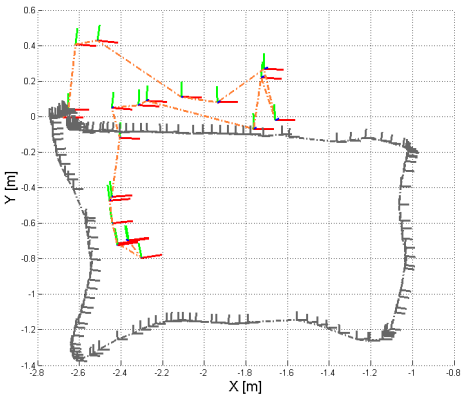
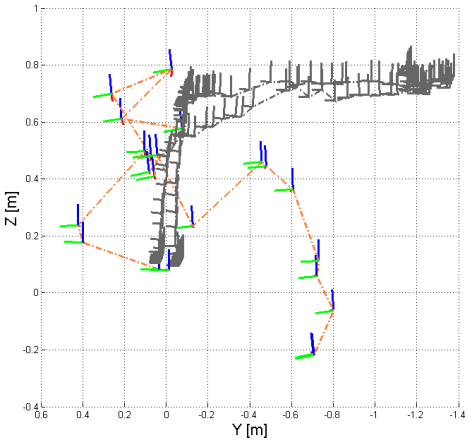


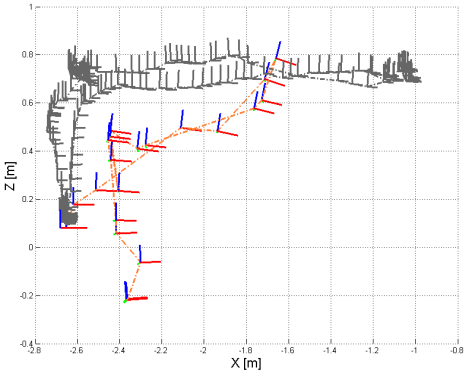
Figure B.16: CCNY RGBD estimated localization without sensor fusion on the Odroid



(a) XY Plot

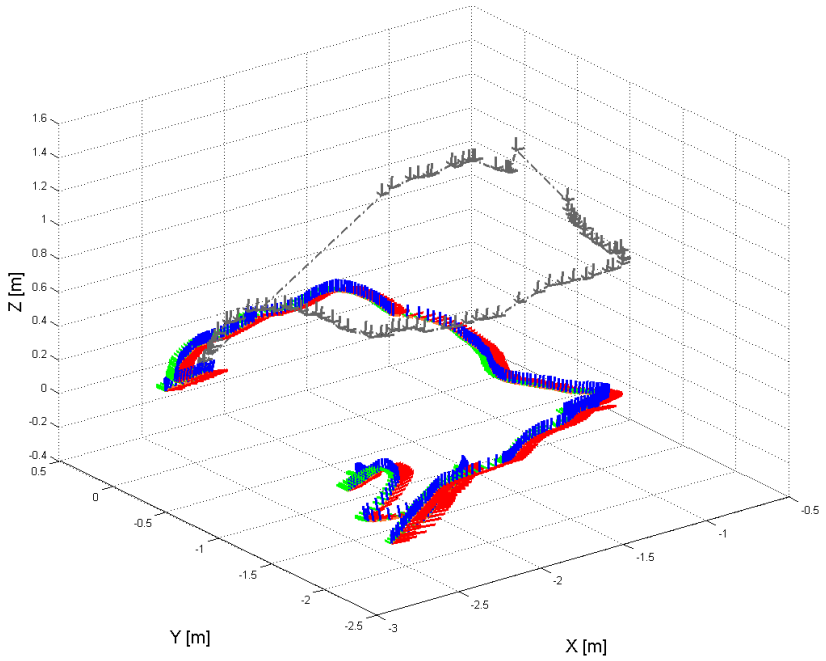


(b) YZ Plot

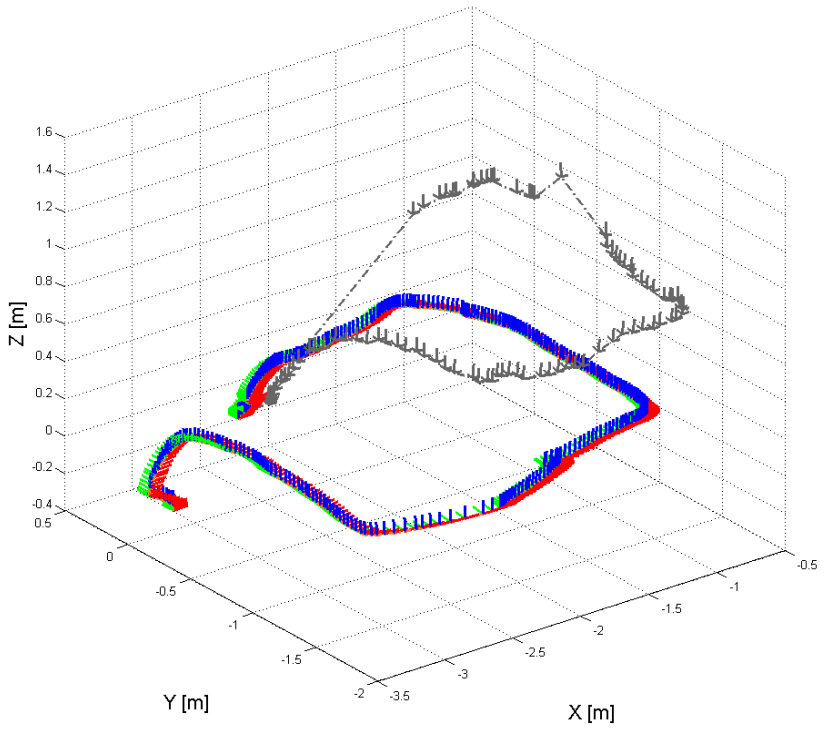


(c) XZ Plot

Figure B.17: CCNY RGBD estimated localization without sensor fusion on the Odroid VS Ground truth - perspectives



(a) DEMO with optical flow fusion VS Ground truth



(b) DEMO without optical flow fusion VS Ground truth

Figure B.18: Comparison between localization estimation with and without optical flow measurements fusion VS Ground truth