TÉCNICO LISBOA

Graph-based 6D SLAM for RGB-D Sensors and Hybrid Vision and Depth Camera Localization

Diogo Alexandre Rolo

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

October 2013

Our greatest weakness lies in giving up.
The most certain way to succeed is always to try just one more time.

Thomas Edison

Acknowledgments

This project is not only the finalization of a masters degree but it also a personal milestone that allowed me to work in a complex problem in such an interesting area as Search and Rescue with mobile robots and for that possibility I would like to thank my project advisors, professor Pedro Lima and professor Rodrigo Ventura.

I would also like to thank my family for the advice and support within this period, even giving me the possibility of staying one month home-alone during the summer holidays to work at my own pace.

A special thank you to my girlfriend Sara for encouraging me when I needed encouragement, supporting me when I needed support and for the walks for unwinding when I needed to clear my head.

Thanks to all my colleagues and friends that provided me with insightful discussions, guidance talks and overall help. Thank you Bernardo Costa, Filipe Jesus, Henrique Silva, Pedro Santos, Tiago Carreira.

Abstract

The focus of this thesis is to develop a method of performing a full 6D Simultaneous Localization and Mapping (SLAM) using an RGB-D sensor in an unknown environment. The map is represented in 3D plus color which allows for a human operator to easily identify interest zones. The built map also allows posterior navigation with two different sensors: an RGB-D sensor such as the used for mapping, or an RGB camera. While navigating in the built map it is possible to track other vehicles in the scenario as long as they were previously identified with an visual marker.

The robot pose is estimated based on natural features of the environment using feature extraction methods such as SURF or GFT. These features are used as landmarks for pose estimation in frame associations.

In order to perform the complete map and trajectory estimation, the algorithm is based on graph-based minimization methods and libraries such as iSAM with square root information matrix factorization.

To reduce the complexity of the optimization, it is used a keyframe based approach where only selected frames contribute for the map construction. Furthermore, while performing online SLAM, the complexity is further reduced by only searching for pose estimation associations in similar frames within a neighborhood.

The experimental results were executed both in a quadcopter and handheld cameras, showing robustness to very different environments.

Keywords

Simultaneous Localization and Mapping, incremental Smoothing and Mapping, RGB-D map, Visual Odometry, Vision-Based Localization

Resumo

O foco deste trabalho é desenvolver um método capaz de realizar Localização e Mapeamento Simultâneo (*SLAM*) em 6D, usando para tal um sensor *RGB-D*, num ambiente desconhecido. O mapa é representada em 3D com cor, permitindo a um operador humano a identificação facilitada das zonas de interesse. O mapa construído também permite a navegação posterior com dois sensores diferentes: um sensor *RGB-D*, como o utilizado para o mapeamento, ou uma câmara *RGB*. Durante a navegação no mapa criado é possível identificar e localizar outros veículos no cenário, desde que tenham sido previamente identificados com um marcador visual.

A estimativa da pose do robô é feita baseado em *landmarks* naturais do ambiente usando como métodos de extracção de *landmarks* tais como SURF GFT. Esses *landmarks* são usados como pontos de referência para a estimativa de pose entre associações de imagens.

Para realizar o mapa completo e a respectiva estimativa de trajectória, o algoritmo é baseado em métodos de minimização de grafos e bibliotecas como o *iSAM* com *Square Root Information Matrix*.

A fim de reduzir a complexidade da optimização é utilizada uma abordagem baseada no imagens-chave onde apenas magens seleccionadas são usadas para a construção do mapa. Além disso, durante a execução do *SLAM on-line* a complexidade é ainda mais reduzida por haver apenas uma procura para estimação de associações de pose em imagens semelhantes, dentro de uma área circundante.

Os resultados experimentais foram executados tanto num micro veículo aéreo não tripulado como em mão, mostrando a robustez para ambientes variados.

Palavras Chave

Localização e mapeamento simultâneo, Mapeamento e suavização incremental, mapeamento *RGB-D*, Odometria Visual, Localização Baseada em Visão

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

COLAMD Column Approximate Minimum Degree

EPnP Efficient Perspective-n-Point

ESC Electronic Speed Controller

FREAK Fast REtinA Keypoint

FIFO First In First Out

G-ICP Generalized Iterative Closest Point

GFT Good Features to Track

iSAM incremental Smoothing and Mapping

IMU Inertial Measurement Unit

IR Infrared

ICP Iterative Closest Point

MLE Maximum Likelihood Estimator

MAV Micro Aerial Vehicle

ORB ORientated Binary robust independent elementary features

PPM Pulse Position Modulation

RANSAC Random Sample Consensus

RGB Red-Green-Blue

RGB-D Red-Green-Blue-Depth

ROS Robot Operating System

RPY Roll, Pitch, Yaw

SLAM Simultaneous Localization and Mapping

SVD Singular Value Decomposition

SURF Speeded-Up Robust Features

Nomenclature

$\mathbf{\Gamma}$	Landmark measurement covariance matrix
$\mathbf{\Lambda}$	Robot model covariance matrix
L	Vector of landmarks
t	Translation vector
U	Vector of control inputs
V	Vector of Gaussian noise associated with the landmark measurement
W	Vector of Gaussian noise associated with the robot pose
X	Vector of robot states
X’	World coordinates in homogeneous coordinates
x’	Pixel coordinates in homogeneous coordinates
Z	Vector of landmark measurements
Q	Quaternion

1 Introduction

Contents

1.1 Motivation

Using Search and Rescue robots in urban environments in catastrophes or emergency situations is becoming increasingly important. They are able to actuate in environments that could cause harm to a human, such as radioactive contaminated zones, collapsed buildings or even environments where humans cannot reach such as small confined spaces.

The robots can then be instructed to perform various tasks such as exploration and mapping of unknown areas, transport of heavy or dangerous loads, surveillance and rescue.

Using teams of robots instead of a single robot to perform a task has several advantages[1] such as:

- Building several smaller, simpler robots allows for a more flexible and robust system;
- Approach to more complex problems by dividing the problem in several sub-problems that then can be approached by different robots;
- If a robot has a failure, the remaining robots can replace him, adding redundancy to the system.

For these reasons, it is extremely important to enable cooperation among a team of robots in a search and rescue scenario.

In this thesis the concept of Simultaneous Localization and Mapping is used because building a map while being able to localize in a unknown environment is an essential skill for mobile robots in GPS-deprived zones.

Using RGB-D sensors as means to acquire dense 3D data allows for a robust pose estimation procedure due to the rich data that is provided from the camera, both giving an RGB image for landmark extraction and a depth image for position extraction of given landmarks.

Furthermore, using an exact and efficient solving algorithm to the SLAM problem, such as iSAM, allows for both incremental and offline optimizations providing a final globally optimal solution.

The rich data provided joined with the versatility of iSAM allows the development of robust algorithms based on the built map, that give localization pose estimations on any robot that navigates in the environment with an RGB camera or RGB-D sensor.

1.2 Problem Statement

In almost any mobile robot application, the robot must be able to accurately estimate its own pose. The pose estimation is performed with proprioceptive readings, such as wheel encoders or IMU, and exteroceptive readings, such as cameras or laser scanners.

While on many land robots the pose estimation problem can be reduced to estimating only Cartesian coordinates and the orientation, when using aerial robots it becomes necessary to estimate all six degrees of freedom.

Using the sensors that a robot is usually equipped with, getting a pose estimate via odometry, either wheel odometry or visual odometry, is trivial. The problem resides in maintaining an accurate estimation over time, since odometry suffer from incremental drift since it doesn't provide any mean of direct pose correction and therefore it has unbounded error accumulation. The process of incrementally integrating the odometry readings to obtain a pose estimation is known as *Dead Reckoning*

In order to improve this method, it is then useful to provide pose correction algorithms that allow the robot to recognize areas that it already visited thus correcting the estimated pose. This method is known as *Explicit Loop Closure.* Another possible method for pose correction is the usage of absolute sensors such as GPS or beacons. Since the focus of this thesis is on indoor unknown environments, the latter approach is discarded since it requires previous contact with the environment.

The construction of the map which the robot uses to navigate can be an abstract object. For a robot, a map may be a sparse collection of landmarks with an associated pose. On the other hand, for a human operator such map wouldn't make sense since it would be hard to understand what's happening. For a human operator a map in the form of a blueprint or a dense 3D representation would be more suitable. In that perspective, the usage of a RGB-D sensor seems appropriate since it allows for both a map representation that the robot understands, such as sparse landmarks, and a map that a human operator understands, such as a dense colored 3D map.

To provide a globally optimal constructed map, it is important to have an estimate of the complete trajectory of the robot and not only the last pose. As such, a full SLAM approach makes more sense in this context instead of a online approach since the full SLAM approach maintains and optimizes the full robot trajectory. In the context of graph-based SLAM, the online approach is also known as *filtering* approach, while the full SLAM approach is also known as *smoothing* approach.

1.3 Literature Review

SLAM has been a subject of study in the area of robotics for several years and has had many different approaches to the problem. In the 80s, a stochastic approach of measuring spacial relationships by combining sequential states modeled with uncertainty was presented by Smith, Self and Cheeseman [2]. Their approach was to use the Extended Kalman Filter to iteratively estimate the robot's pose. A few years later, in 1993, Gordon et al. published their work on Monte Carlo Filter[3], or Particle Filter. In this work, the approach was to estimate the posterior density of state variables given the observation variables knowing that their algorithm didnt require any assumption on the noise of the system or state-space.

In 1997, Lu and Milios introduced the graph-based SLAM approach[4], that consisted in describing the so called full SLAM problems, which takes in consideration the full set of measurements, as a least square problem. It took, however, several years to graph-based SLAM to became popular due to the complexity of solving the minimization problem. Recent developments in the fields of sparse linear algebra resulted in efficient methods to solve the optimization problems and therefore the graph-based SLAM gradually started getting attention, reaching the top state-of-the-art techniques with respect to speed and accuracy[5] and as such it has been increasingly used in the field of mobile robotics.

Graph-based SLAM representation also has the particularity of being extremely intuitive since the construction of the graph represents the robot or landmark poses as nodes and connects them via a sensor measurements. Currently, the graph-based SLAM problem is usually presented with one of three different representations:

- Belief Net - the SLAM is represented as a direct acyclic graph where the nodes represent random variables which only direcly depend on its predecessors.
- Factor Graph - where there are nodes for the random variables and probability factors that structure the graph concerning their connections.
- Markov Random Field - the problem is represented in an undirected graph and does not have factor nodes. Instead, its adjacency structure implies which variables are linked by a common factor

While these representations are only an abstraction, they are helpful in creating an intuitive way of visualizing the problem.

Nowadays there are three major approaches to the SLAM problem, even-though in this thesis only the third will be approached:

- Kalman and information filter[2, 6, 7]
- Particle, or Monte Carlo, filter [8–10]
- Graph based slam [4, 11–14]

While initially, the optimization methods for graph-based SLAM were based on offline optimizations, continuous work has provided the changes necessary for the minimization algorithms to perform iteratively.

Kaess and Dellaert [14] have been continuously working in the past years in the development of SLAM algorithms via Square Root Information Smoothing eventually releasing incremental Smoothing and Mapping (iSAM)[13].

iSAM performs fast incremental updates of the square root information matrix and yet is able to compute the full map and trajectory of the vehicle at any given time.

While usually SLAM is performed to build a 2D map, using laser range finders and odometry, in this thesis it is approached the concept of a full 3D map which implies 6D robot pose estimation in an MAV[15, 16]. The fact of using an MAV deprives the possibility of using wheel odometry for pose estimation and as such visual odometry[17, 18] methods are used.

The abstraction layer provided by the graph-based SLAM methods also allow for the integration of several sensors, parameterizations and models creating an almost out-of-the-box tool for any SLAM system.

1.4 Contributions

The contributions that this thesis provides are divided in two major categories:

1. A method for an out-of-the-box visual SLAM algorithm, based on RGB-D keyframes, that involves explicit loop closure techniques with an incremental graph solving approach. The SLAM algorithm can be run on any kind of robot, being it aerial or land, or can even be run handheld. The SLAM algorithm implemented in this thesis requires no prior knowledge of the mapping scenario, the used robot or any other environment parameter, as long as the RGB-D sensor can properly acquire data. The SLAM algorithm can be run online while mapping for fast results or can be latter run in offline mode for stronger, more accurate results.

2. The usage the acquired map for localization and tracking of other vehicles. The map itself can be used with any vehicle and not only the one that created the map, allowing for a wider range of choices when performing a search and rescue operation. The localization and tracking can be done with a RGB-D sensor or a simpler RGB camera.

The goal of this work is to develop a method to build a 3D map running in the quadcopters which then can be passed to a land vehicle to self-localize. The quadcopter should be able to perform SLAM to localize itself and incrementally build a map and, on a posterior run, should be able to track its own and other robot's path.

1.5 Thesis Outline

The thesis is organized in the following way: Chapter 2 presents the main theoretical aspects used in the development of the thesis related to the iSAM graph optimization, the camera model and pose estimation algorithms. Chapter 3 presents the hardware and sensors used in the project, along with the developed software architecture. Chapter 4 shows the tests and results obtained, using handheld and quadcopter mounted sensors , while mapping, self-localizing and tracking. Finally, Chapter 5 wraps up this thesis and comments on possible improvements and future work.

2

Theoretical Background

Contents

This chapter is focused on the main theoretical aspects related with the development of this thesis and the underlying project.

iSAM is introduced as a possible solution for the Simultaneous Localization and Mapping (SLAM) problem, detailing the techniques that create the backbone of the algorithm. Computer vision related topics are approached explaining the used approaches to relevant data extraction and are also introduced methods of pose estimation with the extracted 3D data.

2.1 Graph based SLAM for Pose Estimation

2.1.1 Introduction

The goal of any SLAM algorithm is to provide an estimation of both the map and the robot pose.

In this project it is used a graph-based SLAM algorithm named iSAM[12, 19]. iSAM is a SLAM algorithm based on the square root information matrix[20] that is able to compute fast incremental updates. It is then possible to add the sensors data as soon as it is available and get the current estimate for the robot pose at that given instant.

iSAM uses a smoothing approach to SLAM, based on the least square problem and reaches a solution based on matrix factorization. It also does variable reordering[13] in order to sparsify the square root information matrix which speeds up the solving process.

2.1.2 Probabilistic Model

iSAM is modeled in terms of a belief network consisted by robots states **X**, landmarks **L**, control inputs **U** and landmark measurements **Z**[19]

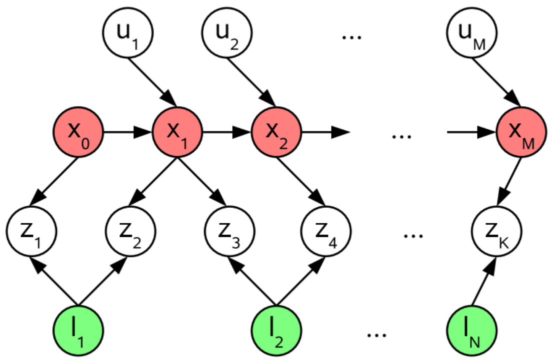

Figure 2.1: iSAM Belief Network[13]

In the above figure, x_i represent the robot pose at instant i. l_j represent the location of landmark j. u_i is the control input at instant i and z_k is the k^{th} landmark measurement. The edges represent conditional dependences between the connected variables.

Assuming known correspondences between landmarks $i_k \rightarrow j_k$, the joint probability is then given by[14]:

$$P(\mathbf{X},\mathbf{L},\mathbf{U},\mathbf{Z}) \propto P(x_0)\prod_{i=1}^{M} P\left(x_i|x_{i-1},u_i\right)\prod_{k=1}^{K} P\left(z_k|x_{i_k},l_{j_k}\right) \tag{2.1}$$

where $P(x_0)$ is a prior for the first robot state, $P(x_i|x_{i-1}, u_i)$ is the motion model and $P(z_k|x_{i_k}, l_{j_k})$ is the landmark measurement model.

A Gaussian noise model is also assumed for the robot motion model[21], **W** with associated covariance matrix $\mathbf{\Lambda}$, and landmarks measurement model,**V** with associated covariance matrix $\mathbf{\Gamma}$:

$$\begin{aligned} x_i &= f_i\left(x_{i-1}, u_i\right) + w_i \\ z_k &= h_k\left(x_{i_k}, l_{j_k}\right) + v_k \end{aligned} \tag{2.2}$$

where w_i is a normally distributed zero-mean process noise with the covariance matrix $\mathbf{\Lambda}$ and v_k is a normally distributed zero-mean measurement noise with the covariance matrix $\mathbf{\Gamma}$

2.1.3 Least Squares Model

The associated least square problem, which gives the optimal **X*** and **L*** is then obtained by minimizing the negative log of the joint probability distribution in Equation (2.1):

$$\begin{aligned} \mathbf{X}^*, \mathbf{L}^* &= \arg\max_{\mathbf{X},\mathbf{L}} P(\mathbf{X}, \mathbf{L}, \mathbf{U}, \mathbf{Z}) \\ &= \arg\min_{\mathbf{X},\mathbf{L}} \ -\log P(\mathbf{X}, \mathbf{L}, \mathbf{U}, \mathbf{Z}) \end{aligned} \tag{2.3}$$

Then, getting the optimal **X*** and **L*** results to solving the following non-linear least square problem:

$$\mathbf{X}^*, \mathbf{L}^* = \arg\min_{\mathbf{X},\mathbf{L}} \left\{ \sum_{i=1}^{M} ||f_i(x_{i-1}, u_i) - x_i||^2_{\Lambda_i} + \sum_{k=1}^{K} ||h_k(x_{ik}, l_{jk}) - z_k||^2_{\Gamma_k} \right\} \tag{2.4}$$

The notation $||\mathbf{e}||^2_{\Sigma}$ is used to describe the squared mahalonobis distance $\mathbf{e^T\Sigma^{-1}e}$ with covariance matrix Σ.

This equation can then be solved with non-linear optimization methods such as Gauss-Newton or Levenberg-Marquardt if the functions f_i or h_k are not linear and if a good linearization point is not available.

2.1.4 System Linearization

While the previous Equation (2.4) can be solved with non-linear optimization methods, these methods rely on linearizing the functions in each iteration. The linearized function can then be used with linear solving methods such as QR factorization which allow for efficient and incremental solving.

In Equation (2.4), the non-linear terms are f_i and h_k. Linearizing these terms can be done via first-order Taylor expansion.

On the first term of Equation (2.4), the linearization is given by:

$$\begin{aligned} & f_i(x_{i-1}, u_i) - x_i \\ \approx & \left\{f_i(x^0_{i-1}, u_i) + F^{i-1}_i \delta x_{i-1}\right\} - \left\{x^0_i + \delta x_i\right\} \\ = & \left\{F^{i-1}_i \delta x_{i-1} - \delta x_i\right\} - a_i \end{aligned} \tag{2.5}$$

where F_i^{i-1} is the Jacobian of the robot motion model $f_i(.)$ at the point of linearization x_{i-1}^0 and is defined by:

$$F_i^{i-1} = \left.\frac{\partial f_i(x_{i-1}, u_i)}{\partial x_{i-1}}\right|_{x_{i-1}^0} \tag{2.6}$$

and where $a_i = x_i^0 - f_i(x_{i-1}^0, u_i)$ is the odometry prediction error.

The linearization of the second term is done similarly:

$$\begin{aligned} & h_k(x_{ik}, l_{j_k}) - z_k \\ \approx & \left\{h_k(x_{i_k}^0, l_{j_k}^0) + H_k^{i_k}\delta x_{i_k} + J_k^{j_k}\delta l_{j_k}\right\} - z_k \\ = & \left\{H_k^{i_k}\delta x_{i_k} + J_k^{j_k}\delta l_{j_k}\right\} - c_k \end{aligned} \tag{2.7}$$

where $H_k^{i_k}$ is the Jacobian of the measurement function $h_k(.)$ with respect to a change in x_{i_k} and $^{j_k}_{k}$ is the Jacobian of the measurement function $h_k(.)$ but this time with respect to a change in l_{j_k}, both evaluated around the linearization point $(x_{i_k}^0, l_{j_k}^0)$ getting:

$$H_k^{i_k} = \left.\frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial x_{i_k}}\right|_{(x_{i_k}^0, l_{j_k}^0)} \tag{2.8}$$

$$J_k^{j_k} = \left.\frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial l_{j_k}}\right|_{(x_{i_k}^0, l_{j_k}^0)} \tag{2.9}$$

and where $c_k = z_k - h_k(x_{i_k}^0, l_{j_k}^0)$ is the measurement prediction error.

After the linearization of the model (2.5) and measurements (2.7) is perfomed, Equation (2.4) becomes:

$$\begin{aligned} \delta\theta^* = \arg\min_{\delta\Theta} & \left\{\sum_{i=1}^{M} ||F_i^{i-1}\delta x_{i-1} + G_i^i\delta x_i - a_i||_{\Lambda_i}^2 \right. \\ & \left. + \sum_{k=1}^{K} ||H_k^{i_k}\delta x_{i_k} + J_k^{j_k}\delta l_{j_k} - c_k||_{\Gamma_k}^2 \right\} \end{aligned} \tag{2.10}$$

where $G_i^i = -I_{d_x \times d_x}$ is used as substitution of δx_i in order to have the same notation along the whole equation. The vector θ is the concatenation of robot poses and landmark variables.

Equation (2.10) is already a full linear least square problem but it can be further simplified in order to be solved efficiently.

The matrices Λ_i and Γ_k can be removed from the equation if the terms F_i^{i-1},G_i^i and a_i are multiplied by ${\Lambda_i}^{T/2}$ and the terms $H_k^{i_k}$, $J_k^{j_k}$ and c_k are multiplied by ${\Gamma_k}^{T/2}$.

These simplifications allow to reach the simple least squares problem that can be described with:

$$\Theta^* = \arg\min_{\Theta} ||A\Theta - b||^2 \tag{2.11}$$

where A is a large but sparse measurement matrix, composed of the Jacobian matrices and b is the concatenation of vectors a_i and c_k.

2.1.5 Matrix Factorization

In the scope of normal equations, the linear least squares problem (2.11) can be also expressed as:

$$A^T A\Theta = A^T b \tag{2.12}$$

QR decomposition helps to solve this kind of equations efficiently.

Matrix A can be decomposed into the product $A = QR$ where $Q \in \mathbb{R}^{m\times m}$ is a orthogonal matrix and $R \in \mathbb{R}^{n\times n}$ is a upper triangular matrix and in the case of iSAM is also the square root information matrix:

$$A = Q\begin{bmatrix}R\\0\end{bmatrix} \tag{2.13}$$

Applying the factorization to the least square problem of Equation (2.11) yields in:

$$\begin{aligned}\|A\Theta - b\|^2 &= \left\|Q\begin{bmatrix}R\\0\end{bmatrix}\Theta - b\right\|^2\\ &= \left\|Q^TQ\begin{bmatrix}R\\0\end{bmatrix}\Theta - Q^Tb\right\|^2\\ &= \left\|\begin{bmatrix}R\\0\end{bmatrix}\Theta - \begin{bmatrix}d\\e\end{bmatrix}\right\|^2\\ &= \|R\Theta - d\|^2 + \|e\|^2\end{aligned} \tag{2.14}$$

where $\begin{bmatrix}d\\e\end{bmatrix} = Q^Tb$ leaving $\|e\|^2$ as the residual of the least square problem.

The factorization then simplifies the problem to a linear problem with a unique solution:

$$R\Theta^* = d \tag{2.15}$$

Due to R being upper triangular, solving this equation is just back-substitution and the result Θ^* is the whole robot trajectory and map.

iSAM uses QR factorization using Given's rotation (2.16) in order to clear the entries bellow the diagonal one at a time. While this is not the optimal way to do a full QR factorization, it is useful when computing it incrementally since the matrix is sparse and only a handful of entries must be computed.

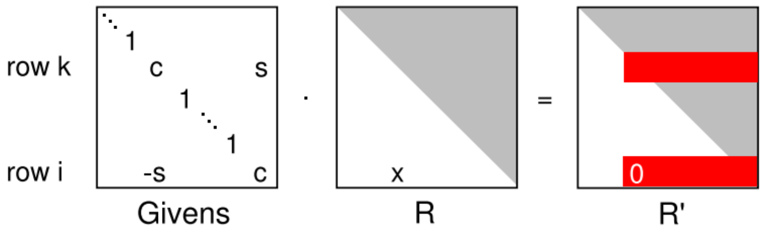

Figure 2.2: Given's rotation used to eliminate an entry[13]

Using Given's rotation (2.16) to perform QR factorization starts by eliminating the left-most non-zero entry, continuing column wise, and it is applied to rows i and j, $i > j$, such that the chosen entry (i, j) becomes zero.

$$\Phi = \begin{bmatrix} cos(\phi) & sin(\phi) \\ -sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.16}$$

After all entries bellow the main diagonal are zero, the resulting upper triangular matrix is the square root of the R matrix of the QR factorization. By applying the same rotations to the vector **b** of the least square problem, the formation of the dense Q matrix is avoided. This way, solving the least square problem $A\mathbf{x} = \mathbf{b}$ with a sparse A matrix allows for faster solving.

The Given's rotation are numerically stable if the rotations are determined the following way[22]:

$$(cos(\phi), sin(\phi) = \begin{cases} (1,0) & \text{if } \beta = 0 \\ \left(\frac{-\alpha}{\beta\sqrt{1+(\frac{\alpha}{\beta})^2}}, \frac{1}{\sqrt{1+(\frac{\alpha}{\beta})^2}} \right) & \text{if } |\beta| > |\alpha| \\ \left(\frac{1}{\sqrt{1+(\frac{\beta}{\alpha})^2}}, \frac{-\beta}{\alpha\sqrt{1+(\frac{\beta}{\alpha})^2}} \right) & \text{otherwise} \end{cases} \tag{2.17}$$

where $\alpha = a_{kk}$ and $\beta = a_{ik}$.

2.1.6 Variable Reordering

With loops in the trajectory and multiple landmarks observations, correlation between current poses and previously observed landmarks appears in the graph. These correlations create a large increase of non-zero entries in the factor matrix which then results in a increased computation complexity.

Finding the right order of the variables is NP-hard but there are efficient heuristics that yield good results when applied to SLAM-specific problems such as Column Approximate Minimum Degree (COLAMD).

iSAM applies variable reordering every n-steps in order to keep the the R matrix sparse while not increasing the overhead too much.

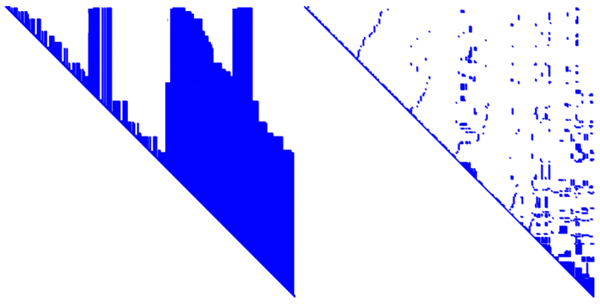

Figure 2.3: Matrix R before and after variable reordering[13]

2.1.7 State Definition

It is assumed that at each iteration, the robot performing SLAM has access to both a prior, which could come from visual odometry or similar, and to an RGB-D image (or just RGB if the robot is only using the graph to navigate).

The depicted problem has two main coordinate systems:

- **World Frame** (X^W, Y^W, Z^W) is a fixed frame and both robot poses and map are in respect to it.
- **Camera Frame** $(X^{C_t}, Y^{C_t}, Z^{C_t})$ is a moving frame that related the pose of the camera with the World Frame. The landmarks associated with each robot pose are given in the camera's frame.

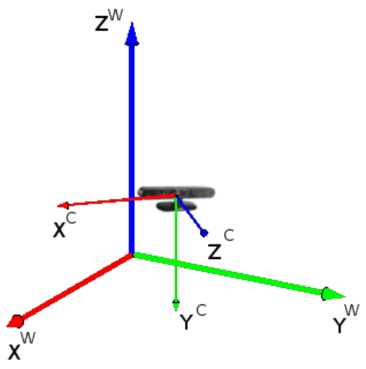

Figure 2.4: World and Camera Frames

Every landmark observation is described in the respective camera frame, as explained in Figure (2.1).

The odometry from pose x_i to x_{i+1} is given in form of a quaternion Q and a translation vector $\mathbf{t}$:

$$Q = [q_x, q_y, q_x, q_w]$$
$$T = [t_x, t_y, t_z] \tag{2.18}$$

A quaternion to be valid must have unit norm so the following equation must be true:

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{2.19}$$

When using aerial vehicles it is very commom to use the euler angles Roll, Pitch, Yaw (RPY) which give a better intuitive values to the robot pose than using quaternions.

To convert from quaternion to RPY:

$$\begin{aligned} roll &= atan2(2 * (q_w * q_x + q_y * q_z), q_w^2 - q_x^2 + q_y^2 + q_z^2) \\ pitch &= asin(2 * (q_w * q_y - q_z * q_x)) \\ yaw &= atan2(2 * (q_w * q_z + q_x * q_y), q_w^2 + q_x^2 - q_y^2 - q_z^2) \end{aligned} \tag{2.20}$$

and the reverse operation, from RPY to quaternion:

$$\begin{aligned} s_y &= sin(yaw * 0.5); \qquad c_y = cos(yaw * 0.5); \\ s_p &= sin(pitch * 0.5); \qquad c_p = cos(pitch * 0.5); \\ s_r &= sin(roll * 0.5); \qquad c_r = cos(roll * 0.5); \\ q_w &= c_r * c_p * c_y + s_r * s_p * s_y; \\ q_x &= s_r * c_p * c_y - c_r * s_p * s_y; \\ q_y &= c_r * s_p * c_y + s_r * c_p * s_y; \\ q_z &= c_r * c_p * s_y - s_r * s_p * c_y; \end{aligned} \tag{2.21}$$

Throughout this thesis both the quaternion representation and the RPY or rotation matrix representation will be used interchangeably since they all represent the same rotation.

2.2 Computer Vision

2.2.1 Introduction

In order to be able to perform SLAM in a full 3D environment it is necessary to use different sensors than the usual laser range finder. It is necessary to get a dense pointcloud from which the depth can be perceived and the interesting landmarks could be extracted. Using RGB-D cameras gives the possibility of using the depth information to compute camera pose estimation as well as a RGB image that allows for feature tracking and matching.

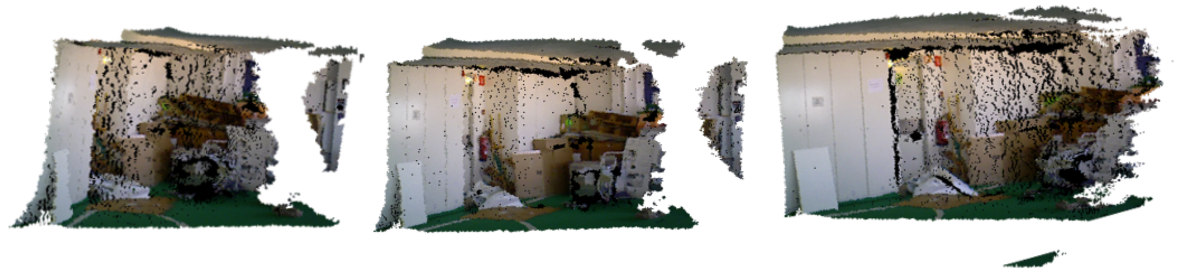

Figure 2.5: One RGB-D pointcloud captured with a Asus-Xtion Pro. Views of the same cloud from different perspectives

With all the information that these kind of sensors provide, computer vision then becomes a main component of 3D-SLAM.

2.2.2 Camera Model

Cameras are able to project 3D points into a 2D image plane (see Figure (2.6))[23]. The projection is described by a perspective transform:

$$\lambda \mathbf{x}'_0 = \Pi \mathbf{X}'_0 \tag{2.22}$$

where $\mathbf{X}'_0$ are the points in the world frame using homogeneous coordinates, $\mathbf{x}'_0$ are the points in image frame using homogeneous coordinates, λ is a scale factor and Π is the camera matrix:

$$\Pi = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \tag{2.23}$$

The **K** matrix is the intrinsic parameters matrix and the **R**,**t** are the extrinsic parameters which define the camera center and orientation in the world coordinate frame.

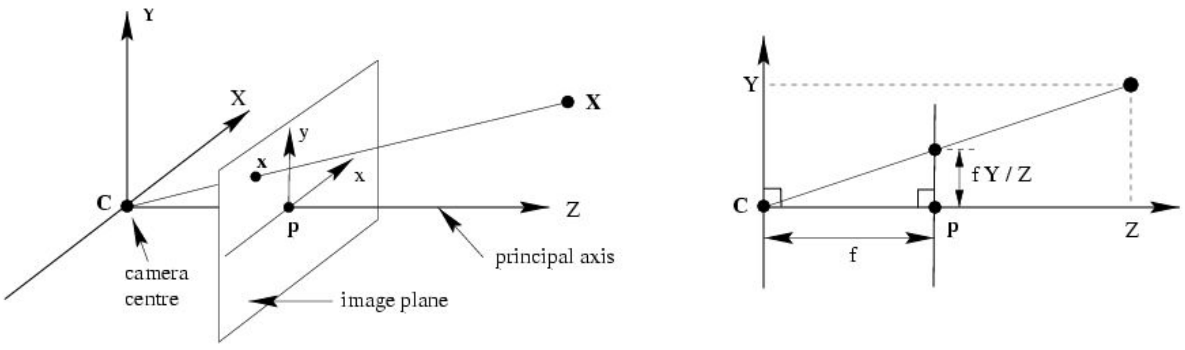

Figure 2.6: Model of the projection of a World object into the camera frame [24]

The intrinsic matrix can be further detailed as such:

$$\mathbf{K} = \begin{bmatrix} k_x & \gamma & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

The parameters k_x, k_y represent the focal length in terms of pixels in the camera frame, c_x, c_y represent the principal point of the image, normally around the center of the image and γ represents the skew factor, normally around 0.

Since Equation (2.22) has a free variable λ, it can be solved up to a scaling factor. Using depth information, it is possible to add a constraint that uniquely solves the equation.

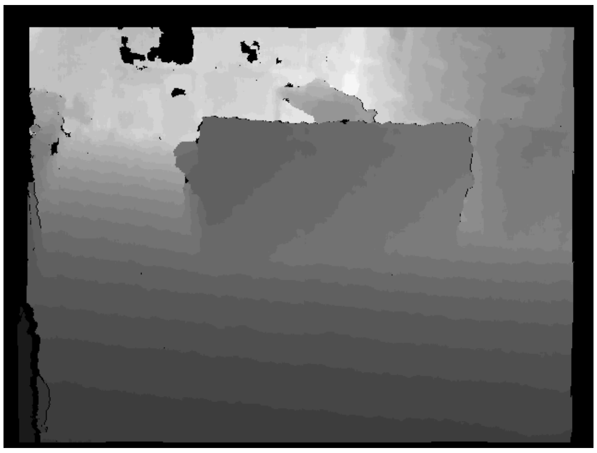

Figure 2.7: Depth Map, black means a bad reading

It is assumed that the depth image is registered, which means both the depth and RGB image data points are in the same coordinate system. Being in the same coordinate systems allows for the direct correspondec of pixel (i, j) in the RGB with pixel (i, j) in the depth image. To perform the registration, the extrinsic parameters between the two cameras were already applied to the images and as such they can now be assumed as identity **R** = **I**, **t** = **0**. Assuming also skew factor 0 for simplification, the new equations are as follows:

$$\Pi' = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.25}$$

Getting the points in the 2D image to 3D coordinates can be calculated:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.26}$$

Further solving the system gets to:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & c_x \\ 0 & k_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.27}$$

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} k_x X + c_x Z \\ k_y Y + c_y Z \\ Z \end{bmatrix} \tag{2.28}$$

Since Z is given by the depth map, it is possible to fix the λ value:

$$\lambda = Z \tag{2.29}$$

Solving in order of X:

$$X = \frac{Z(x - c_x)}{k_x} \tag{2.30}$$

and Y:

$$Y = \frac{Z(x - c_y)}{k_y} \tag{2.31}$$

With Equations (2.29), (2.30) and (2.31) is possible to go back and forth between 2D image points and 3D world points.

2.2.3 Feature Detection

In order to extract useful landmarks and features from the mapping environment, it is necessary to run feature extraction on the gathered images. All the feature detection is made on the RGB image since it is far more information rich that the depth map (see Figure (2.7)).

Getting sparse features from a dense image [25] is also important when fast computational speed is required because it represents a very significant complexity decrease.

2.2.3.A SURF: Speeded Up Robust Features

Speeded-Up Robust Features (SURF)[26] is a scale and rotation invariant detector and descriptor based on the use of integral images and basic Hessian-matrix approximations.

Calculating the integral image is done the following way:

$$\mathbf{I}_{\Sigma}(x, y) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j) \tag{2.32}$$

With Equation (2.32), the computation of the integral of any rectangular section is done with only 3 operations which allows for higher order filters to be applied without losing computational speed.

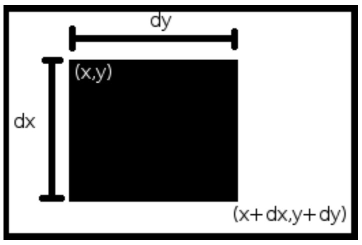

Figure 2.8: The integral of the section (x,y) to (x+d_x,y+d_y) can be calculated with only 3 sums

$$\sum_{i=x}^{i\leq x+\delta x}\sum_{j=y}^{j\leq y+\delta y} I(i,j) = \mathbf{I}_{\Sigma}(x,y) + \mathbf{I}_{\Sigma}(x+\delta x, y+\delta y) - \mathbf{I}_{\Sigma}(x, y+\delta y) - \mathbf{I}_{\Sigma}(x+\delta x, y) \tag{2.33}$$

To select the features to use, using Equation (2.33) helps computing the Hessian Matrix, which is the image convolution over a second order Gaussian derivative, by only performing three additions:

$$H(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{yx}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix} \tag{2.34}$$

After computing the determinant of the Hessian matrix for all image points, the local maximums are chosen using non-maximum suppression [27].

Regarding the feature descriptor, in order to distinguish one feature from another, this descriptor creates a description of the intensity content within the interest point neighborhood.

To create rotation invariance, Haar wavelet responses are calculated in x and y directions.

After computing the descriptor, the scale invariance is implemented by normalizing the resulting vector.

2.2.3.B Good Features to Track

Good Features to Track (GFT)[28] is a tracking algorithm for features based on dissimilarity and motion models. This algorithm uses a translation motion model to compare consecutive frames and affine motion model to compute the dissimilarity between distant frames. In this project, the GFT algorithm is only used to estimate visual odometry so the translation motion model is preferred.

The image motion can be described by the following equation:

$$I(x,y,t+\tau) = I(x-\xi(x,y,t,\tau), y-\eta(x,y,t,\tau)) \tag{2.35}$$

where $\delta = (\xi,\eta)$ is the displacement of each point in the instants between t and $t+\tau$.

Even though this model describes the generic motion model, it is often violated due to reasons such as occlusions, disappearing features and lightning changes.

It is then used a local model that tries to compute the motion of a feature inside a given window:

$$\delta = D\mathbf{x} + \mathbf{d} \tag{2.36}$$

where D is the deformation matrix and **d** is the translation.

In small windows, the deformation matrix is very prone to error which causes a bad estimate of the feature motion. On the other hand, small windows are best used to tracking because they are most likely to contain features at a similar depth. Therefore, it is assumed that D is an identity matrix and a pure translation model is adopted for tracking:

$$I_2(\mathbf{x} + \mathbf{d}) = I_1(\mathbf{x}) \tag{2.37}$$

where I_2 is the second image and I_1 is the first image.
Computing the motion is done by solving the following equation:

$$\epsilon = \int \int_W [I_2(\mathbf{x} + \mathbf{d}) - I_1(\mathbf{x})]^2 \omega(\mathbf{x}) d\mathbf{x} \tag{2.38}$$

The main advantages of using GFT are its computational speed and ability to track moving features. Due to these reasons, it is a good algorithm to base the visual odometry on.

2.2.4 Feature Matching

Feature matching consists in finding the same feature in two different images.

The method basically consist in minimizing the difference between the descriptors of one image with the descriptors of another.

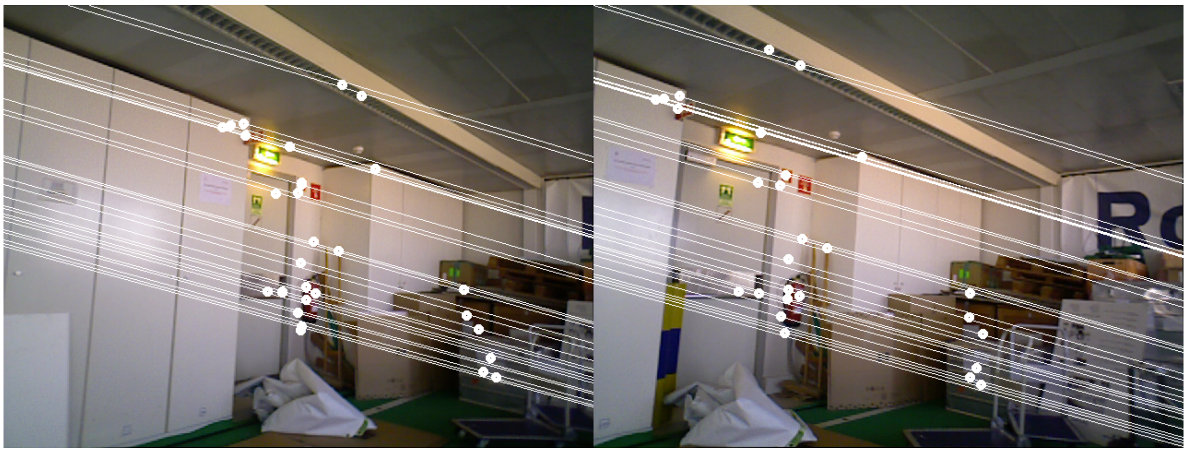

Figure 2.9: Matched features between two different images

When implementing feature matching with SURF descriptors, the algorithm is based on the minimization of the euclidean distances among the features to match. As such, any algorithm that perform this minimization is suitable to do the feature matching, being it *K-dimensional Tree* algorithms or brute force algorithms. The trace of the Hessian matrix, described in Equation (2.34), is taken into account in order to be robust to contrast variations.

In order to create a more robust matching operation, other techniques can be implemented:

- Ration test - Accept only a match if the best match distance is q times smaller than the second best.
- Symmetry test - Accept only a match if it is both the best match from $I_1 \rightarrow I_2$ and $I_2 \rightarrow I_1$
- Epipolar geometry:
  - Select 8 matches at random and estimate the Fundamental Matrix.

- Accept only a match if it is in the epipolar line $\pm\delta$ pixels of the other image.
- Retain the matches that were acquired from the fundamental matrix that discarded the fewer matches

Algorithm 2.1 Robust SURF matching

```
Input: All detected keypoints
Output: Robust keypoints matched
for i = 1 to 2 do

  for all detected features in $I_i$ do
    $match_1 \leftarrow$ GetBestMatch()
    $match_2 \leftarrow$ GetSecondBestMatch()
    if $match1.distance < match2.distance * q$ then
      $matches_i = matches_i + match_1$
    end if
  end for
end for
for all match in $matches_1$ do
  if not match in $matches_2$ then
    Reject match
  end if
end for
for all match in $matches_2$ do
  if not match in $matches_1$ then
    Reject match
  end if
end for
$MaxMatches$ = 0
$RobustMatches$ = {}
for i = 1 to iterations do
  $RandomMatches \leftarrow$ Get8RandomMatches()
  $Matrix \leftarrow$ EstimateFundamentalMatrix($RandomMatches$)
  $CurrentMatches \leftarrow$ GetEpipolarMatches($\delta$)
  if $CurrentMatches.count$ > $MaxMatches$ then
    $RobustMatches \leftarrow CurrentMatches$
    $MaxMatches \leftarrow CurrentMatches.count$
  end if
end for
```

With the implementation of the Algorithm (2.1) it is possible to get a very robust feature matching system with near zero false positives. As a downside, depending on the threshold parameters, the algorithm is prone to get a large quantities of false negatives. As such, its usage is dependent on the final goal. In this project, this algorithm is mainly used to identify frames that belong to the same mapping area so, in this case, it is better to have fewer stronger matches than the opposite.

2.2.5 ArUco

ArUco is a minimal Augmented Reality tracking library developed by the Universidad de Córdoba. It is mainly used to create Augmented Reality[29] applications, like the name suggests, but it is also useful in other applications such as tracking other robots.

This library detects markers places in the environment and gives the camera position relative to the marker itself.

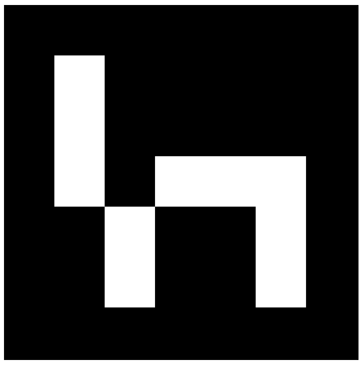

Figure 2.10: Marker ID 26 generated from the ArUco library

This library functions in two major stages:

- Marker Detection
- Camera's Pose Estimation

Regarding the Marker Detection stage, the image is first binarized using adaptive thresholding or any similar method[30].

Then, the connected components are analyzed to search for interest areas where the marker might be. Knowing that the marker is a rectangle, heuristics can be applied to select the best candidate areas and thus extracting the marker position.

In order to identify the marker ID, a CRC error code is implemented to get the correct ID even when small errors or occlusions appear in the image. If no valid ID is found, the detected candidate marker is discarded.

Regarding the Camera's Pose Estimation stage, Equation (2.26) is to be solved to estimate the camera's pose. When two parallel sides of the detected square marker are projected into the camera frame, its equations can be described by:

$$\begin{aligned} a_1\text{x} + b_1\text{y} + c_1 = 0 \\ a_2\text{x} + b_2\text{y} + c_2 = 0 \end{aligned} \tag{2.39}$$

Given that the camera's calibration is known and the marker's size. Joining the two Equations (2.26) and (2.39),the following system is reached:

$$\begin{aligned} a_1k_xX + b_1k_yY + (a_1c_x + b_1c_y + c_1)Z = 0 \\ a_2k_xX + b_2k_yY + (a_2c_x + b_2c_y + c_2)Z = 0 \end{aligned} \tag{2.40}$$

Solving these equations and obtaining two normal vectors regarding the world frame, in order to get the third vector the cross product is applied: $n_1 \times n_2$

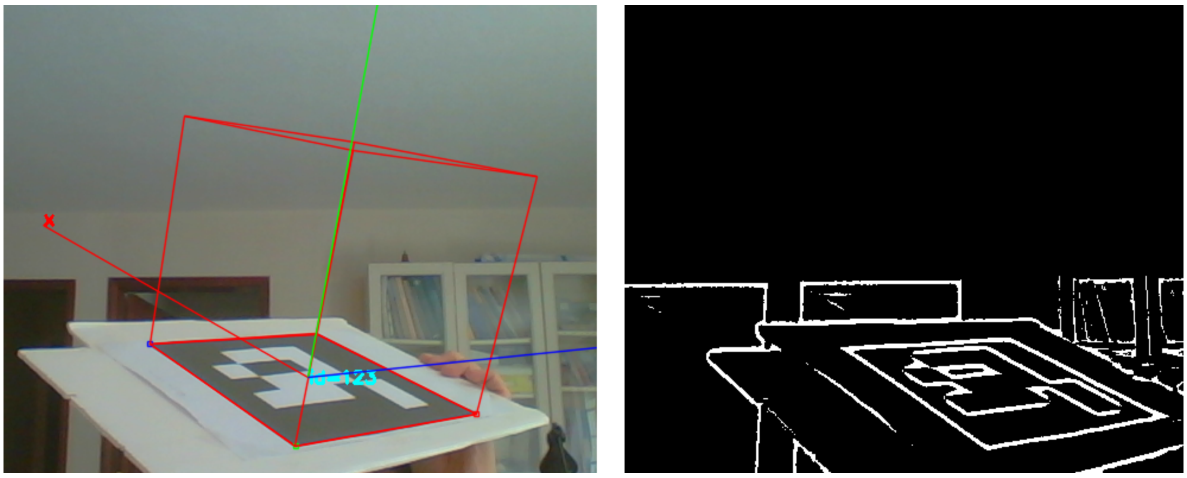
Figure 2.11: A detected ArUco marker showing the associated axis

2.3 Pose Estimation

2.3.1 Introduction

In order for robot to be able to correctly navigate any environment, it is crucial for it to know its own position and orientation, also known as pose.

The task associated with the determination of the pose of a robot is referred as pose estimation and is the core of SLAM:

In order to the robot to be able to localize itself it needs a map, but in order to create a map it needs to know its pose.

In order to solve this chicken and egg problem a map can be incrementally created and later be corrected with loop closing techniques such as feature matching and pose estimation between different frames gathered along mapping.

2.3.2 Generalized Iterative Closest Point

While Iterative Closest Point (ICP) is an algorithm employed to minimize the difference between two pointclouds, comparing point to point, Generalized Iterative Closest Point (G-ICP)[31] is a generalized model for standard ICP. G-ICP is based on attaching a probabilistic model to the minimization algorithm of ICP. This description is made general such that both point-to-point ICP and point-to-planeICP are particular cases of the model.

The standard point-to-point ICP algorithm is described the following way:

Algorithm 2.2 Iterative Closest Point

```
$T \leftarrow T_0$
while not converged do

  for $i \leftarrow 1$ to $N$ do
    $m_i \leftarrow FindClosestPointInA(T \cdot b_i)$
    if $||m_i - T \cdot b_i|| \leq d_{max}$ then
      $w_i \leftarrow 1$
    else
      $w_i \leftarrow 0$
    end if
  end for
  $T \leftarrow \underset{T}{\arg\min} \; \{ \sum_i w_i ||T \cdot b_i - m_i||^2 \}$
end while
```

Assuming perfect correspondence between point a_i from pointcloud A and point b_i from pointcloud B, using the correct transformation $\mathbf{T}^*$ it is known that:

$$\hat{b_i} = \mathbf{T}^* \cdot a_i \tag{2.41}$$

Considering an arbitrary transformation $\mathbf{T}$, the distance between the points a_i and b_i is given by:

$$d_i^{\mathbf{T}} = b_i - \mathbf{T} \cdot a_i \tag{2.42}$$

The standard ICP minimization step is the following:

$$\underset{T}{\arg\min} \left\{ \sum_i w_i ||T \cdot b_i - m_i||^2 \right\} \tag{2.43}$$

Assuming a_i and b_i are points that are given from independent Gaussian Distributions, described in equation 2.42, can be statistically described by:

$$d_i^{\mathbf{T}^*} = N(\hat{b_i} - (\mathbf{T}^*) \cdot \hat{a_i}, C_i^B + \mathbf{T}^* C_i^A \mathbf{T}^{*T}) = N(0, C_i^B + \mathbf{T}^* C_i^A \mathbf{T}^{*T}) \tag{2.44}$$

Using Maximum Likelihood Estimator (MLE) to the above equation in order of $\mathbf{T}$ yields:

$$\mathbf{T} = \underset{T}{\arg\max} \prod_i p(d_i^{\mathbf{T}}) = \underset{T}{\arg\max} \sum_i \log(p(d_i^{\mathbf{T}})) \tag{2.45}$$

In order to simplify the equation, using equation 2.44 the key step of the G-ICP algorithm is acquired:

$$\mathbf{T} = \underset{T}{\arg\min} \sum_i d_i^{\mathbf{T}^T} (C_i^B + \mathbf{T} C_i^A \mathbf{T}^T)^{-1} d_i^{\mathbf{T}} \tag{2.46}$$

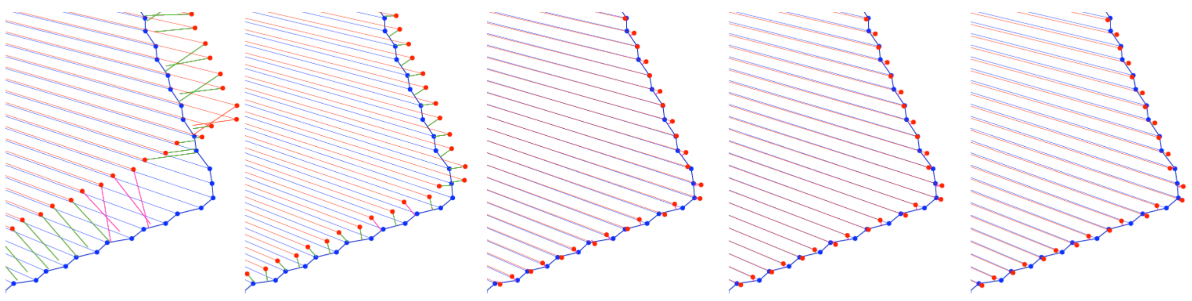

Figure 2.12: Visualization of the Iterative Closest Point Algorithm [32]

Comparing Equation (2.46) with the minimization step of point-to-point ICP in Equation (2.43), it can be observed that the later is a special case the equation where:

$$C_i^B = I$$
$$C_i^A = 0$$

Comparing with point-to-plane algorithm [31], it can be observed that it is also a special case where:

$$C_i^B = \mathbf{P}_i^{-1}$$
$$C_i^A = 0$$

where $\mathbf{P}_i$ is the projection of a point into the surface normal at b_i.

2.3.3 Random Sample Consensus

Random Sample Consensus (RANSAC)[33] is a robust estimator that is used to fit a model to experimental data. It is a model that takes into account all given data and iteratively tries to select a subsection of the data that minimizes a given cost function.

The algorithm works in a simple way[34]:

- Select a random subset of data
- Fit model parameter into the subset of data
- Evaluate the quality of the model into the whole input data

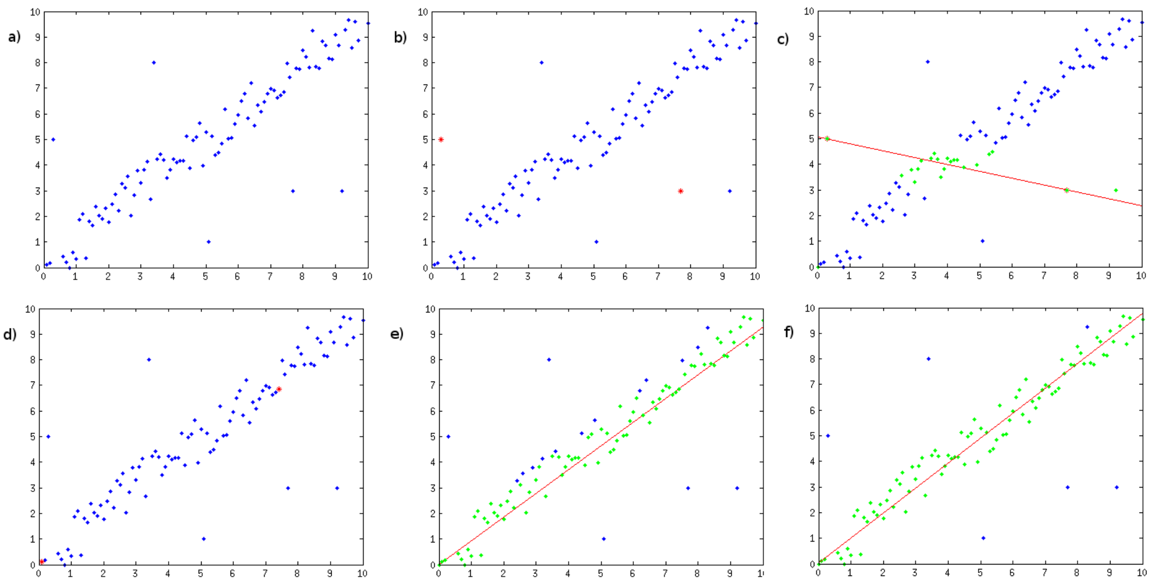

Figure 2.13: RANSAC algorithm trying to get the model of a line. a) Raw data. b) Two random points selected. c) Model estimated with those points and inliers. d) 2 other random points selected. e) Model estimated with the new points and inliers. f) Model estimated with the best model inliers

These steps are repeated until a maximum iteration threshold is reached or when the probability of finding a better model is lower than a predefined threshold η_0.

First, a subset of points is randomly selected from all the estimated keypoints.

The model used in RANSAC to estimate the pose is the Rigid Body Transform:

$$d_i = \mathbf{R}m_i + \mathbf{t} + \mathbf{V}_i \tag{2.47}$$

where d_i and m_i are two matched points and **R** and **t** are the rotation matrix and translation vector that transform one point into the other, corrupted by the noise $\mathbf{V}_i$.

Finding $\mathbf{R}^*$ and $\mathbf{t}^*$ consists in minimizing the least squares criterion given by:

$$\epsilon^2 = \sum_{i=1}^{N} ||d_i - \mathbf{R}m_i + \mathbf{t}||^2 \tag{2.48}$$

In order to minimize equation 2.48, the point set d_i and m_i should have the same centroid.

Calculating the centroids

$$\begin{aligned} \overline{d} &= \frac{1}{N}\sum_{i}^{N} d_i \qquad d_{ci} = d_i - \overline{d} \\ \overline{m} &= \frac{1}{N}\sum_{i}^{N} m_i \qquad m_{ci} = d_i - \overline{m} \end{aligned} \tag{2.49}$$

which then rewriting equation 2.48 yields in:

$$\begin{aligned} \epsilon^2 &= \sum_{i=1}^{N} ||d_{ci} - \mathbf{R}m_{ci}||^2 \\ &= \sum_{i=1}^{N} (d_{ci}^T d_{ci} + m_{ci}^T m_{ci} - 2d_{ci}\mathbf{R}m_{ci}) \end{aligned} \tag{2.50}$$

To minimize the equation, the last term has to be maximized. This problem is also known as the *orthogonal Procrustes problem*. This problem can be solved with Singular Value Decomposition (SVD) to maximize the trace of the matrix:

$$H = \sum_{i=1}^{N} m_{ci} d_{ci}^T \tag{2.51}$$

SVD decomposes the matrix **H** into $\mathbf{H} = \mathbf{U}\Lambda\mathbf{V}^T$. From there, $\hat{\mathbf{R}}$ is given by:

$$\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^T \tag{2.52}$$

When estimating $\hat{\mathbf{R}}$, if the two datapoints are planar it may happen that the determinant of $\hat{\mathbf{R}}$ is -1 which indicates a reflection instead of a rotation. In this special case, the optimal rotation[35] is expressed as:

$$\hat{\mathbf{R}} = \mathbf{U} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(\mathbf{U}\mathbf{V}^T) \end{bmatrix} \mathbf{V}^T \tag{2.53}$$

Calculating the translation **t** is then:

$$\hat{\mathbf{t}} = \overline{d} - \hat{\mathbf{R}}\overline{m} \tag{2.54}$$

Getting the parameters $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$ that describe the transformation, using equation 2.47 gives the distance that each point is from the corresponding match. This way it is possible select a list of inliers by selecting the matches whose distances is less than δ and thus evaluating the quality of the estimation.

Regarding the stopping criteria, the number of iterations needed to ensure with probability z that the calculated model is the best model is given by[33]:

$$k = \frac{\log(1-z)}{\log(1-b)} \tag{2.55}$$

where b is w^n and w is the probability of a match being an inlier and n is the number of data points used to estimate the model. In practical cases, since w is difficult to know beforehand, this value is estimated with each iteration giving $w = \frac{Best\ Number\ of\ correct\ Matches}{Total\ Number\ of\ Matches}$

3 Implementation

Contents

This chapter gives an overview of the used hardware specifications as well as the system architecture and development decision that were made.

It is mainly focused the aspects of the RGB-D sensor used, the quadcopter characteristics and the system's software arquitecture.

3.1 ASUS Xtion Pro Live

3.1.1 Introduction

ASUS Xtion Pro Live (see figure 3.1) is a kinect-like sensor that can provide RGB-D images from a USB camera. It has a multi-language (C#, C++, JAVA) API that facilitates the integration with other applications. Robot Operating System (ROS)[36] has a package that allows to receive and process RGB-D images from this camera and sends it to other ROS nodes which is helpful in the context of robotics and real time operations.

Figure 3.1: ASUS Xtion Pro Camera

3.1.2 Specifications

Regarding its specifications:

Power Usage	< 2.5W
Distance of usage	0.8m to 3.5m
Sensors	RGB, Depth, 2 microphones
Depth Image Resolution	VGA (640x480) : 30 fps - QVGA (320x240): 60 fps
RGB Resolution	SXGA (1280*1024) - VGA (640x480)
Dimensions	18 x 3.5 x 5 cm - 230g

Table 3.1: ASUS Xtion Pro Live Specifications
[37]

The most interesting aspects, in the scope of Micro Aerial Vehicles (MAVs) and robotics are its low power usage and small dimensions. The fact that it is small, comparatively with others sensors such as Kinect (30 x 7.5 x 6.5 cm - 1.3kg) makes it a better choice to use in small vehicles. Regarding the official max distance of 3.5m, the sensors outputs distances further than that up to 7m. The values in the further away readings are not as accurate as in the closer readings.

3.1.3 RGB and Depth Images

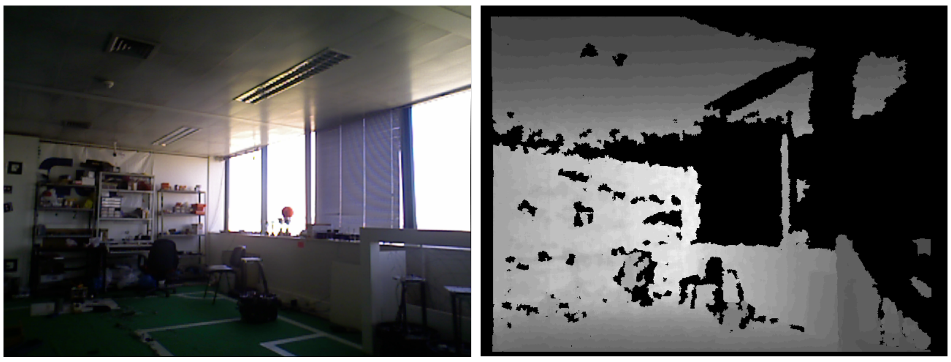

Figure 3.2: RGB-D Image from the Xtion Pro

The RGB camera in the Xtion Pro ends up being a low cost camera with no customization available regarding the aperture size or shutter speed. Not being able to tweak the shutter speed may make the camera's captured images blurry, specially in rotational movements. Regarding to the camera's aperture size auto regulation it creates abrupt brightness and contrast changes when the camera automatically decides to change its value. Fortunately, the camera does not perform auto-focus so the intrinsic parameters remain constant. In order to compensate for the lack of customization of the camera parameters, more robust and computational expensive algorithms have to be used in order to get good computer vision results.

When calibrating both the RGB and IR camera the following intrinsic parameters were obtained:

Camera	K_x	K_y	C_x	C_y
RGB	544.422	544.119	315.385	237.311
IR	570.342	570.343	314.500	235.500

Table 3.2: Camera's intrinsic parameters obtained via calibration

and the distortion parameters k_c for the Red-Green-Blue (RGB) camera:

k_{c1}	k_{c2}	k_{c3}	k_{c4}	k_{c5}
0.037	-0.127	0.002	-0.002	0

Table 3.3: RGB camera distortion parameters

3.2 Quadcopter

3.2.1 Introduction

In the context of Search and Rescue, the used approach is to use a MAV that is running a SLAM algorithm and then a land vehicle navigates in the processed map and is tracked by the MAV As such, a *UAVision Quadcopter UX-4001 mini* is used as the MAV. The quadcopter hardware was developed by a Portuguese company UAVision®while the middle-ware and low level software is handled by *Paparazzi* autopilot.

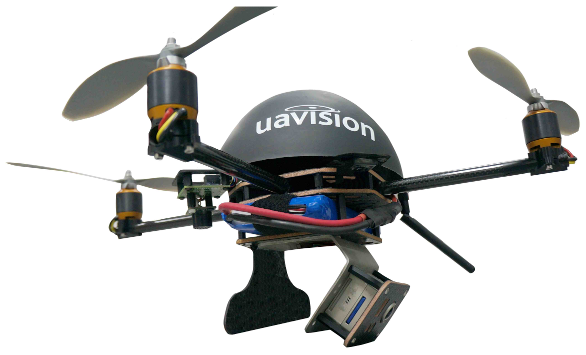

Figure 3.3: UAVision Quadcopter UX-4001 mini [38]

3.2.2 Hardware Specifications

The quadcopter has a wingspan of $65,7cm$ and its hardware can be divided in four different categories: Chassis, Sensors, Actuators and Communication.

The Chassis is made of carbon fiber which is lightweight and resistant.

The sensors are:

- 3 gyroscopes
- 1 3-axis Accelerometer
- 1 pressure sensor
- 1 magnetometer
- 1 outdoor GPS receiver

Regarding the actuators, each wing has a brush-less DC motor controlled by an Electronic Speed Controller (ESC).

The communication part consists of bluetooth module, a Pulse Position Modulation (PPM) radio receiver and a XBee module.

Without any additions to the quadcopter the payload is $500 - 700g$.

The above hardware is the core of the quadcopter but other hardware can be added such as Laser Range Finders, a PandaBoard computer, sonars, optical flow sensors and cameras.

The algorithm is being run of on an Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz with 8GB of RAM computer, outside the quadcopter.

3.3 Software Architecture

3.3.1 Introduction

The software that is running to perform SLAM is divided into three major modules: Visual Odometry, Keyframe Mapping where full slam is implemented, and Localization where the robot uses an already computed map and self-localizes in it.

The nodes were implemented in C++ and were incorporated into the ROS framework in order to facilitate the communication between different parts of the system. Using ROS is also helpful in the way that a great number of repositories are freely available with a vast diversity of drivers and utility libraries. OpenCV[39] also provides several useful functions to use while handling visual processing algorithms.

3.3.2 Visual Odometry

Since a MAV is unable to get a viable odometry via wheel encoders, other means to estimate the vehicle motion must be used. The most common method used then is visual odometry which consists of determining the vehicle motion by analyzing camera images[17, 18].

With any kind of odometry, real time results are mandatory and as such it is important to use algorithms that perform simple computations to estimate the pose of the vehicle.

To estimate the camera motion, the problem can be first split into two parts: getting the interesting features, and the motion estimation itself.

Regarding the first part, computing the features, this step is essential because it creates a much simpler analysis and instead of analyzing the full cloud, only a selected number of points tested. While a full pointcloud has several thousand of points, if only the interesting features are used, the system has a much lower complexity due to only using around 100 to 200 features. To detect the features GFT algorithm is used since it selects the best features to use in a tracking scenario and is also very fast to compute. This step has a tremendous impact on the overall performance.

Algorithm 3.1 Visual Odometry Architecture

```
Input: pointcloud
Output: Pose estimation
$Pose \leftarrow Identity()$
while running do

  $Image \leftarrow GetNewImage()$
  $Features \leftarrow DetectFeatures(Image)$
  $Points \leftarrow GetDepthForFeatures(Features)$
  if $Model$ is $Empty$ then
    $Model \leftarrow Points$
  else
    $Transformation \leftarrow GICP(Model)$
    if $Transformation$ is $Ok()$ then
      $Pose = Pose * Transformation$
    end if
  end if
  $Publish(Pose)$
end while
```

The next step is estimating the motion that the camera performed. After the key features are selected, those features are aligned into a $DataModel$[18] using G-ICP. The $DataModel$ is a pointcloud of the features that were detected so far, up to a cap, represented in the World Frame. Having a model of the features allows for a robust G-ICP when compared to pure incremental G-ICP because it helps to reduce the drift that normally occurs with odometry.

To implement the G-ICP, the cost function between two features is given by the Mahalanobis distance from a point to a distribution:

$$\begin{aligned} dist(f_a, f_b) &= \sqrt{\Delta_{\mathsf{f}_a,\mathsf{f}_b}(\Sigma_a + \Sigma_b)^{-1}\Delta^T_{\mathsf{f}_a,\mathsf{f}_b}} \\ \Delta_{\mathsf{f}_a,\mathsf{f}_b} &= \mu_a - \mu_b \end{aligned} \tag{3.1}$$

where f$_i$ is feature i and μ_i and Σ_i are its mean and covariance matrix.

After the the cost function is minimized and the motion parameters estimated, the model is updated to contain the most recent points.

Considering the general rigid body transformation:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

the image points can be transformed into the model frame:

$$\begin{aligned} \mu' &= \mathbf{R}\mu + \mathbf{t} \\ \Sigma' &= \mathbf{R}\Sigma\mathbf{R}^T \end{aligned} \tag{3.3}$$

A point is considered if the distance on Equation (3.1) to the nearest point is lower than a given threshold ϵ. The features that were not associated are then added to the $DataModel$. Since the model has a maximum number of features, to preserve real-time computational speed, when the cap is reached the oldest features start to be replaced by the newer features.

Finally, the $DataModel$ is updated with a Kalman Filter where the previous distribution m is treated as the prior and the current observation d is treated as the observation.

Both the $DataModel$ and the observation d are a set of 3D features containing a mean and a covariance: $\mathbf{f} = \{\mu, \Sigma\}$. This way the $DataModel$ is:

$$\mu_t^{[M]} = \mu_{t-1}^{[M]} + \Sigma_{t-1}^{[M]} \tag{3.4}$$

where $\mu_t^{[M]}$ is the $DataModel$'s current state and $\Sigma_{t-1}^{[M]}$ is the model noise. The observation d model is similar:

$$\mu_t^{[D]} = \mu_{t-1}^{[M]} + \Sigma_{t-1}^{[D]} \tag{3.5}$$

where $\mu_t^{[D]}$ is the observation state and $\Sigma_{t-1}^{[D]}$ is the observation noise. Note that there are points that are inserted into the $DataModel$ outside the Kalman Filter procedure and as such the dimensionality of the filter increases. These points are inserted into the model with the sensor's reading and associated noise. Also note that there are points in the model that are not observed in every observation. These points are not updated in the corresponding iteration.

The Kalman Filter prediction at time t is:

$$\begin{aligned} \tilde{\mu_t} &= \mu_{t-1}^{[M]} \\ \tilde{\Sigma_t} &= \Sigma_{t-1}^{[M]} \end{aligned} \tag{3.6}$$

and the update:

$$\begin{aligned} K_t &= \tilde{\Sigma}_t \left(\tilde{\Sigma}_t + \tilde{\Sigma}_t^{[D]}\right)^{-1} \\ \mu_t^{[M]} &= \tilde{\mu}_t + K_t \left(\mu_t^{[D]} - \tilde{\mu}_t\right) \\ \Sigma_t^{[M]} &= (I - K_t)\,\tilde{\Sigma}_t \end{aligned} \tag{3.7}$$

Since in ROS any process can listen to the visual odometry publications, it is useful to have it as correct as possible. As such, when the localization node, described in Section 3.3.4, is running and the SLAM graph is optimized, the newest pose from the graph is used to correct the odometry reading.

3.3.3 Keyframe Mapping

Using RGB-D sensors to extract the most important frames of a video is a approach often used in Visual SLAM [40, 41] since it reduces considerably the complexity of the problem. Several metrics can be used when selecting the keyframes such as clustering [42] or more general cost functions [43].

The approach for this project consist in saving a frame as a keyframe every time the MAV moves more than a predefined distance **d** or rotates more than a predefined angle θ.

d and θ should be such that the system has enough information to create a correct and useful trajectory estimate but it cannot be too small which would create the concept of keyframe useless. A value between 10 to 20 cm on **d** and 5 to 10º on θ give good results for the tested environments.

The keyframe mapper node should be running in parallel with the visual odometry in order to get the current position estimate to use as prior and to allow the choosing of keyframes.

Algorithm 3.2 Keyframe Mapping Architecture

```
Input: RGB-D image sequence
Output: 3D Map + 6D robot trajectory
while navigating do

  $Pose \leftarrow GetPoseFromOdometry()$
  if $Pose$ is $FarEnough(\mathbf{r})$ then
    $Image \leftarrow GetNewImage()$
    $Frame \leftarrow CreateFrameWithFeatures(Pose, Image)$
    $AddKeyframe(Frame)$
    $FrameMatches \leftarrow FindCorrespondences(Frame)$
    $AddToGraph(FrameMatches)$
    $Pose \leftarrow OptimizeGraph()$
    $SendCorrection(Pose)$
  end if
end while
$Correspondences \leftarrow CreateGlobalCorrespondences()$
$AddToGraph(Correspondences)$
$OptimizeGraph()$
$SaveMap()$
```

Once a keyframe is chosen, the image keypoints and descriptors are computed using the SURF algorithm. SURF is used because it provides a robust result in moving environments and lightning changes.

Then the similar keyframes within a range **r** are selected in order to try and compute rigid body transformations from the current keyframe to the others. Only searching within a range **r** allows for real time processing because a large amount of keyframes are automatically excluded from the computation. Within the selected range, the candidate keyframes for processing are the keyframes that present the highest similarity to the current keyframe.

Then, a robust matching Algorithm (2.1) is run so the false positives are discarded.

With the remaining candidate keyframes, the pose transformation is computed based on RANSAC. If the pose estimation is successful, the graph factor that connects the two keyframes is added to the optimization graph. If the pose estimation fails due to small number of features, no graph factor is added and instead the remaining landmarks are added. Adding the features helps the getting a better estimate if in the future any other keyframe observes any of those added landmarks. In no other occasion landmarks are added to keep the graph as simple as possible.

When all correspondences are created and added to the optimization graph, iSAM is used to incrementally optimize it and get a more recent estimate of the map and the robot's trajectory. The most recent robot pose is then sent to the odometry algorithm to correct the pose.

While real-time SLAM gives acceptable results for a rough map, in order to get an accurate map the optimization should be run again after all the data is acquired in order to refine the map and trajectory estimated. In this offline optimization the searching of a keyframe within a range **r** is dropped and all similar keyframes are searched for a possible transformation. While this slows the process, it gives more information that the graph can use to create a fine SLAM estimate.

Finally, the map and trajectory are saved so that any other program can later use it to self-localize.

The main ROS node structure can be visualized in the following figure:

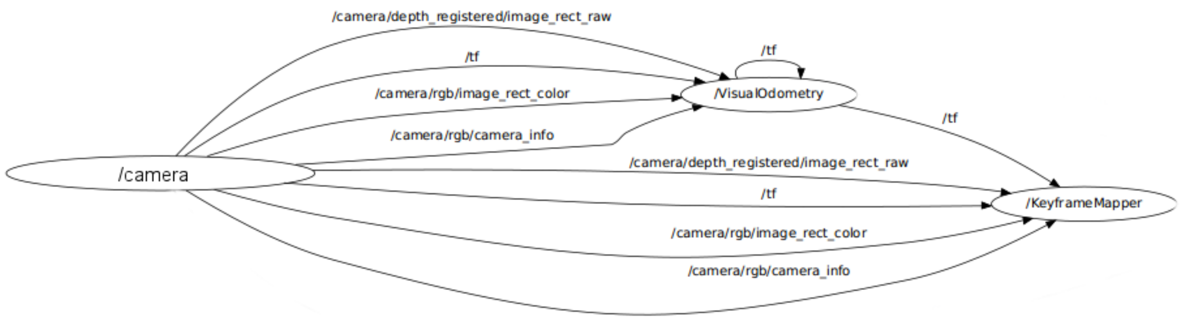
/camera/depth_registered/image_rect_raw
/tf
/tf
/camera/rgb/image_rect_color
/VisualOdometry
/camera/rgb/camera_info
/tf
/camera
/camera/depth_registered/image_rect_raw
/tf
/KeyframeMapper
/camera/rgb/image_rect_color
/camera/rgb/camera_info

Figure 3.4: ROS mapping nodes

3.3.4 Localization

Localization ends up being a subproblem already addressed in the keyframe mapping but since time complexity, while important, is not as critical as when performing SLAM it is then possible to add other features while self-localizing such as locating and tracking other vehicles.

The localization process can be executed with two distinct camera types, a normal RGB camera and a RGB-D. This allows for the diversification of the robots hardware which also allows for diversification on their tasks since normally a RGB camera is smaller and lightweight.

The program runs similarly for both sensors, with the only difference being the algorithm to estimate the camera's pose. With the RGB-D sensor, since full depth information is available, the algorithm is simple estimating a Rigid Body Transformation as in the Mapping part. With the RGB sensor, due to lack of depth data, the Perspective Transform is estimated instead.

Algorithm 3.3 Localization Architecture

```
Input: RGB-D/RGB image
Output: Pose, tracked pose
$Pose \leftarrow Identity()$
$ArucoPose \leftarrow Null$
while running do

  $Image \leftarrow GetNewImage()$
  $Frame \leftarrow CreateFrame(Pose, Image)$
  $AddKeyframe(Frame)$
  if $FoundAruco(Image)$ then
    $ArucoPose \leftarrow AssociateAruco(Frame)$
    $AddToGraph(ArucoPose)$
  end if
  $FrameMatches \leftarrow FindCorrespondences(Frame)$
  $AddToGraph(FrameMatches)$
  $Pose \leftarrow OptimizeGraph()$
  $Publish(Pose)$
  $Publish(ArucoPose)$
end while
```

The program starts by first loading the previously computed map. The map consists of the keyframes along with their pose. The keyframe itself consists of an RGB and a depth image.

An iSAM graph is then created with all the relevant keyframe data and each keyframe pose is initialized with a high value associated in the information matrix so that posterior optimizations during the localization are unable to change the map nodes.

Since this ROS node runs in soft-real time characteristics, time while important, is not critical. As such a frame is added as keyframe as soon as there are computational resources available.

Then, on each keyframe, an ArUco marker is searched. If one is found the procedure to track other vehicles, described in Section 3.3.5, is run.

Just as in the mapping process, described in Section 3.3.3, the most similar map frames within a range are selected to estimate a pose and the features are matched.

In the case of the RGB localization, the camera pose is estimated by trying to find the **R** and **t** that minimizes the projection error of the 3D points into the 2D images.

$$\begin{aligned} \lambda \mathbf{x}'_0 &= K \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{X}'_0 \\ \epsilon^2 &= \|\lambda \mathbf{x}'_0 - \mathbf{x}_0\|^2 \end{aligned} \tag{3.8}$$

This equation can be solved iteratively using numeric optimization methods or can be latter used to measure the total error of the estimation.

There are efficient algorithms such as Perspective-Three-Point [44] and Efficient Perspective-n-Point (EPnP) [45].

In practice, EPnP is used for estimating the pose and Equation (3.8) is used for accepting the estimation if it is below a certain threshold.

Finally, the computed pose is added do the optimization graph and the graph is optimized to yield the latest camera pose estimate.

The main functional block and module interaction between Visual Odometry and Localization can be seen in the following figure:

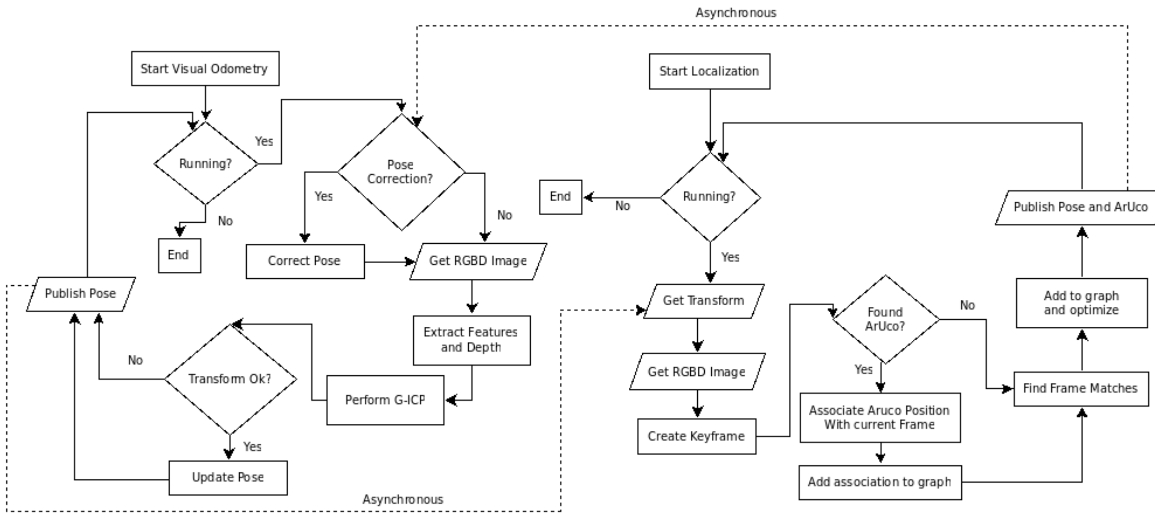

Figure 3.5: VO + Localization Architecture

3.3.5 Tracking other vehicles

The feature to track other vehicles in an extension to the localization part of the project.

The main reason the tracking cannot be done while performing SLAM resides with the fact that a keyframe has to be added everytime another vehicle is found and that would slow down the algorithm to a point where it would be unusable due to the clustering of keyframes. Although while localization there is also the possibility of keyframe clustering, since the localization is done in regard to the original map keyframes, the computational complexity remains constant throughout the whole process.

Each vehicle to be tracked has an ArUco marker (see Figure (2.10)) attached to its body so that it can be seen from the tracking vehicle.

Since each ArUco marker has a different ID, this allows for multiple vehicle tracking with ease.

In each keyframe, the RGB image is searched for markers and if one is found, the camera's pose is extracted as explained in Section 2.2.5.

The marker itself is identified in the graph as a 3D landmark since only the position is tracked. Creating an association in the optimization graph between that landmark and the camera's pose allows for the marker's position estimate to change if the associated camera position also changes. Since it is assumed that the vehicle is moving, if the marker is seen in another frame a new landmark is added along with a new constraint.

To avoid outliers, usually in the form of salt and pepper noise, a robot is only considered to be correctly identified if there were more than one reading in the radius $\mathbf{r}_{marker}$ of any reading. All the readings are then clustered in $\mathbf{r}_{marker}$ size clusters and the output reading is the average of the readings inside. This average works as a low pass filter that removes small Gaussian noise that occurs, due to the tracking vehicle estimate not being perfect, when the tracked vehicle is stopped and the tracking vehicle is moving.

4 Experimental Results

Contents

In this chapter is focused on showing the experiment results obtained with the development of this thesis.

It is given an overview of the used datasets in experimentation and the corresponding results regarding visual odometry, keyframe mapping and navigation with emphasis in the time performance of each task the optimization cost function error in various different modes.

4.1 Datasets

The experimental results presented in this chapter are processed from two different datasets that show different aspects of the implemented software architectures.

In both datasets the data is recorded into a file using the *ROSBAG* application from ROS. The dataset data is composed of four different information streams: the rectified RGB image, the RGB camera's calibration, the rectified depth image and a stream that gives the relative transformation among the physical system's most important sections. While this last stream is constant, it may be useful in the future in the case of using a servo to change the relative pose of the camera with the quadcopter.

The transformation stream outputs at approximately 40Hz and the other streams output at approximately 30Hz. Note that this is the frequency at which the program receives data, not the frequency at which the algorithm works.

Due the impossibility to have a reliable ground truth system such as Vicon, other means for the analysis of the quality of the solution are resorted to.

There are numerous thresholds and constants that can be tweaked and fine-tuned to give slightly better results concerning each environment. In the case of both datasets used, no values are changed to show robustness concerning variability in the mapping scenario.

The most important thresholds and constant are the following:

Visual Odometry:

- Max linear distance between two features: $1 * 10^{-4}$m
- Minimum number of matches: 15 pairs
- Max model size: 1600 points
- Feature detector: GFT

Mapping and Navigation:

- Max distance between keyframes: 10 cm
- Max angular distance between keyframes: 10º
- Number of features per image: 500
- Max number of candidates frames: 10
- Minimum number of matches: 16
- Feature detector: SURF

In order to better define each dataset a brief description is given for each one:

- Dataset 1 - Flight

This dataset consists of real data acquired with a flying quadcopter 3.2.1. On this dataset, the quadcopter starts landed on the floor. It then takes off on a vertical movement and executes two rough consecutive square trips with two distinct orientations, as seen in Figure (4.1). After the two trips it lands again in a slight different position.

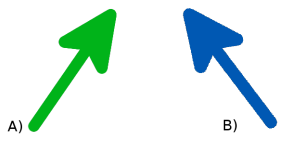

Figure 4.1: Quadcopter orientation of dataset 1.
A) First trip orientation. B) Second trip orientation

For this dataset, the metrics used for analyzing the quality of the solution is the total error after the graph is optimized. The total duration of the dataset is 1 minute and 34 seconds (94 seconds).

The following picture gives an overview of the captured dataset 1:

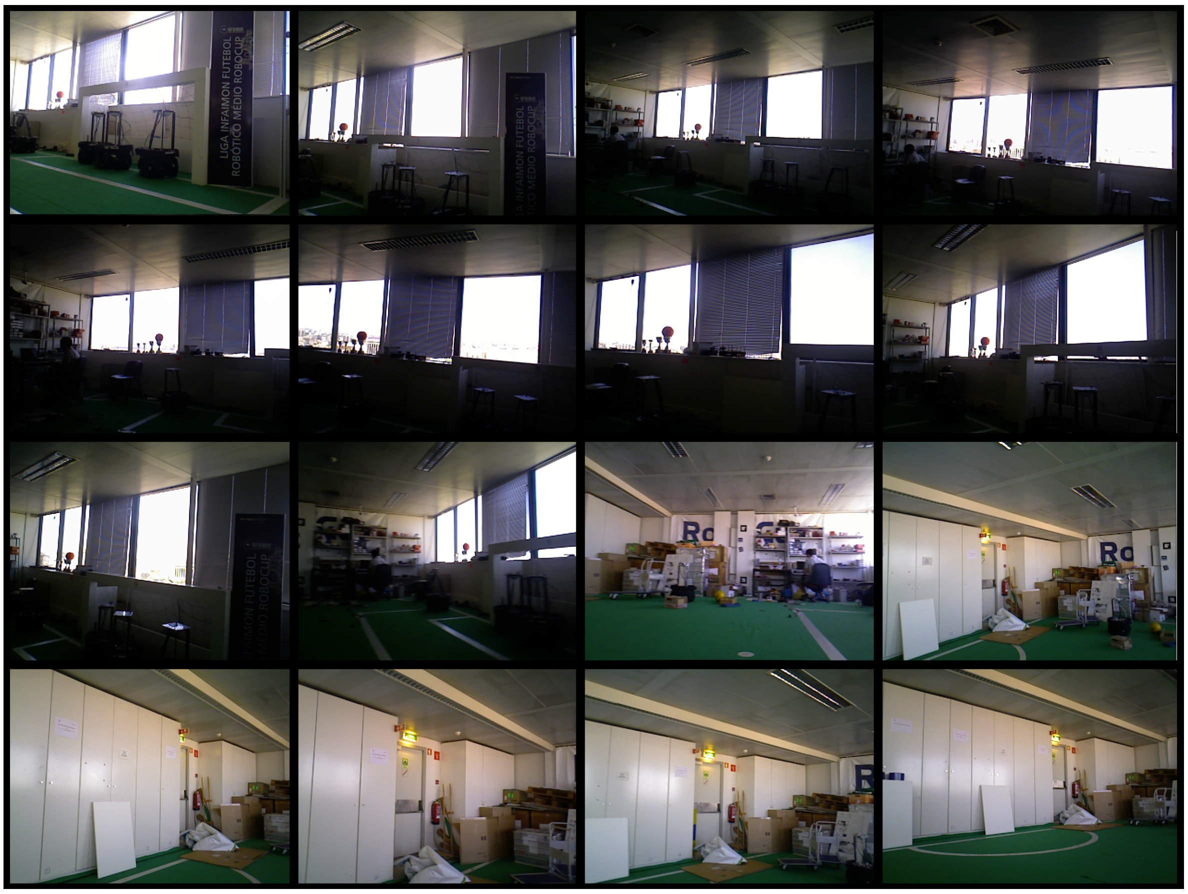

Figure 4.2: Dataset 1 overview

- Dataset 2 - Handheld

This datasets consists of real data acquired with the RGB-D sensor navigating handheld throughout two runs. The goal is to get a stable trajectory where a ground-truth is terms of position can be easily obtained. In this dataset, the sensor starts already facing the path, it then navigates three meters forward. When it reaches that point it navigates one meter back and one meter of the left. Finally it returns one meter to the right and two meters back to the starting point. The sensor is always with roughly the same orientation throughout all the dataset. This run is used for mapping the environment.

The following picture given an overview of the run used for mapping the environment:

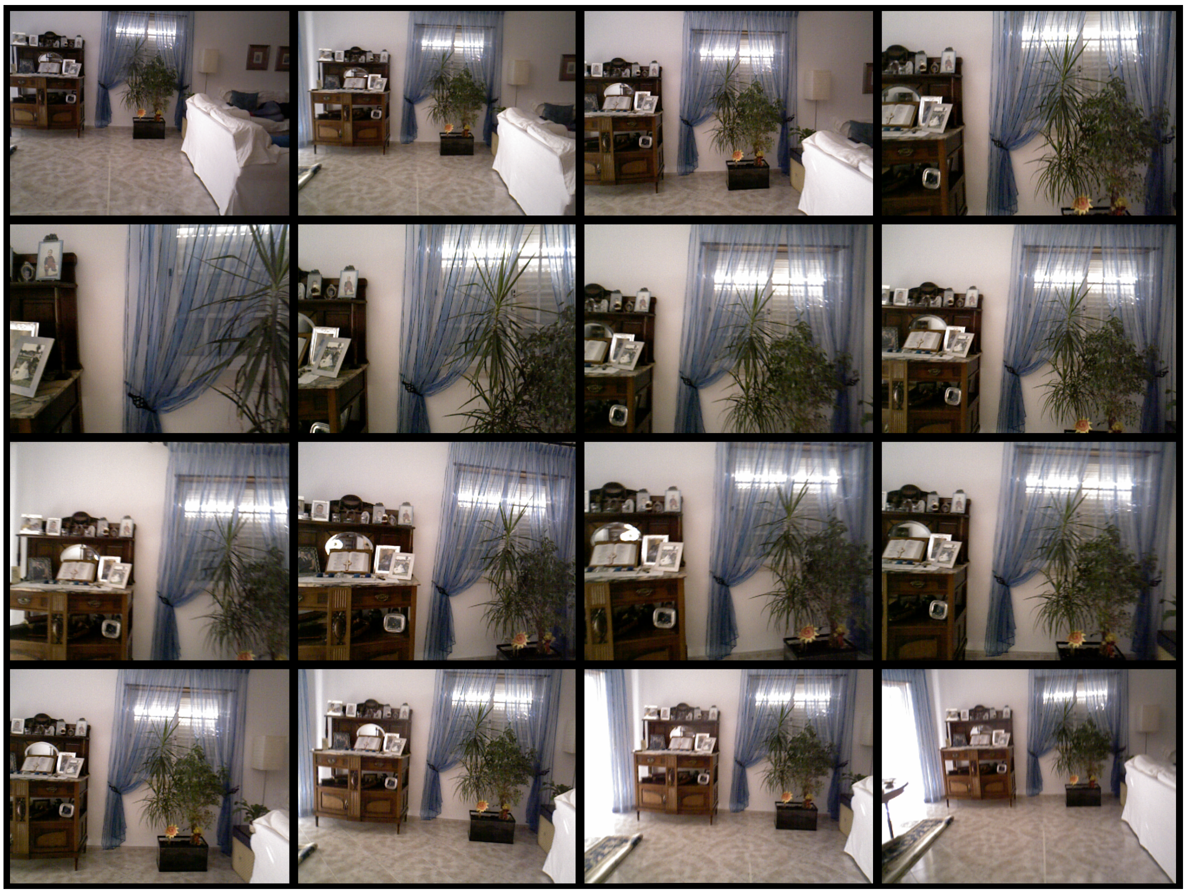

Figure 4.3: Dataset 2 mapping overview

On another run, used for navigation, while the sensor is performing the same maneuver, there is another robot with an ArUco marker standing four meters away in the direction the sensor is facing. When the sensor returns to the starting position, the robot with the marker moves three meters towards the sensor.

The following picture given an overview of the run used for navigating and tracking in the environment:

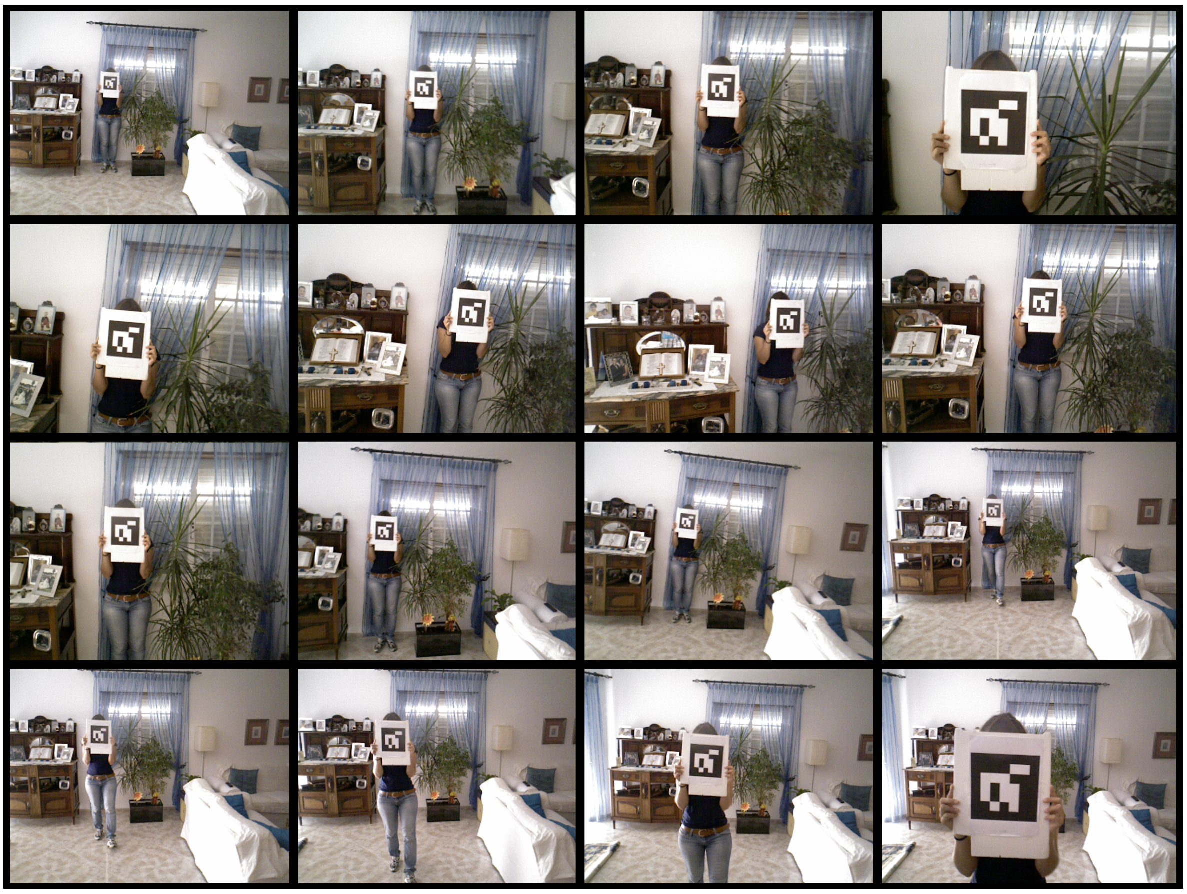

Figure 4.4: Dataset 2 mapping overview

On both runs the trajectory is approximately the same with a constant orientation, following the path described in Figure (4.5).

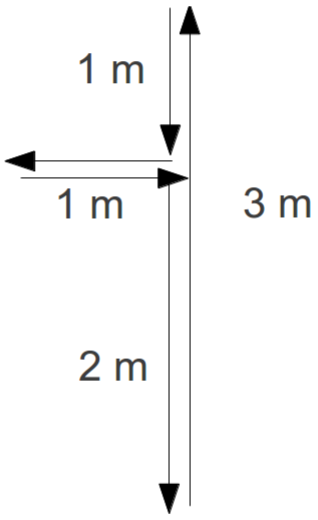

Figure 4.5: Sensor position while performing dataset 2

The *ROSBAG* files gathered while creating the datasets are available in [46].

4.2 Visual Odometry

Regarding Visual Odometry, there is mainly one issue that has to be addressed due to the real time performance necessity and that is the time complexity of the algorithm.

Time performance is highly related to the specification of the computer where the algorithm is being run on.

4.2.1 Time performance

Concerning time performance, it is useful to see how much time is being used to process each individual frame, Figure (4.6) and (4.7), and to see which processes take more time, Figure (4.8) and (4.9).

The goal is then to obtain a visual odometry estimate with such a frequency that it is useful and meaningful in a real time performance scenario.

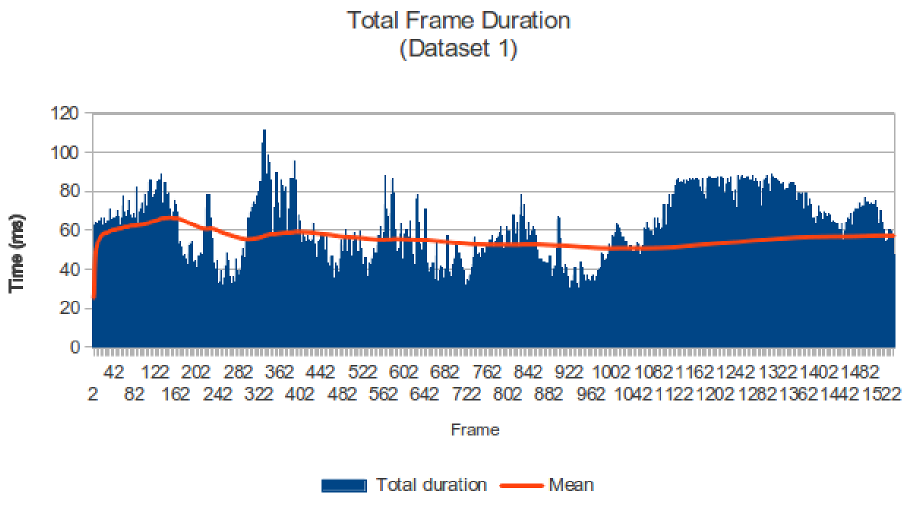

Figure 4.6: Visual Odometry Frame Duration on Dataset 1

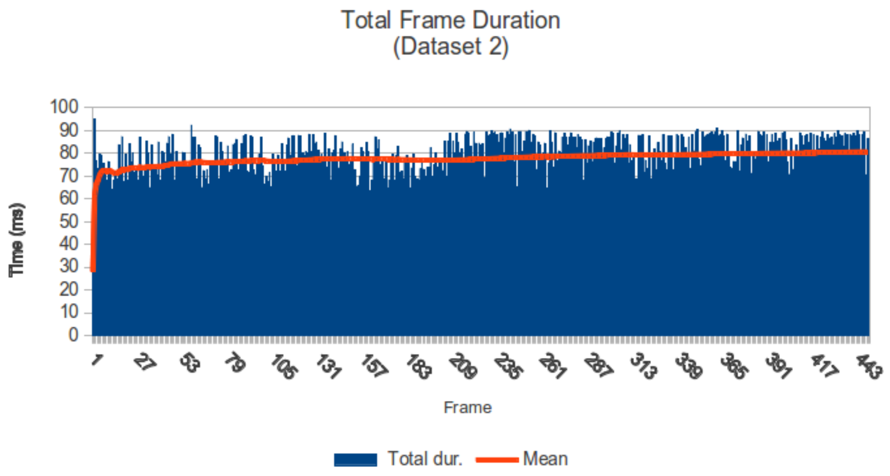

Figure 4.7: Visual Odometry Frame Duration on Dataset 2

From the comparison of the last two figures, it is possible to see that there are some differences in processing time related to different mapping scenarios. Even in Dataset 2, where the average processing time is superior, a decent odometry frequency can be obtained.

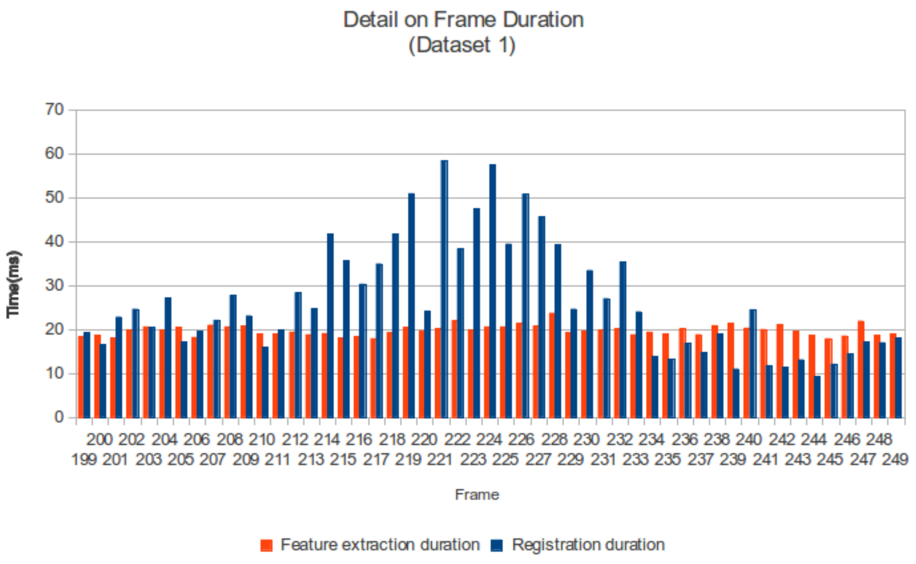

Figure 4.8: Detail on Visual Odometry processing time, dataset 1

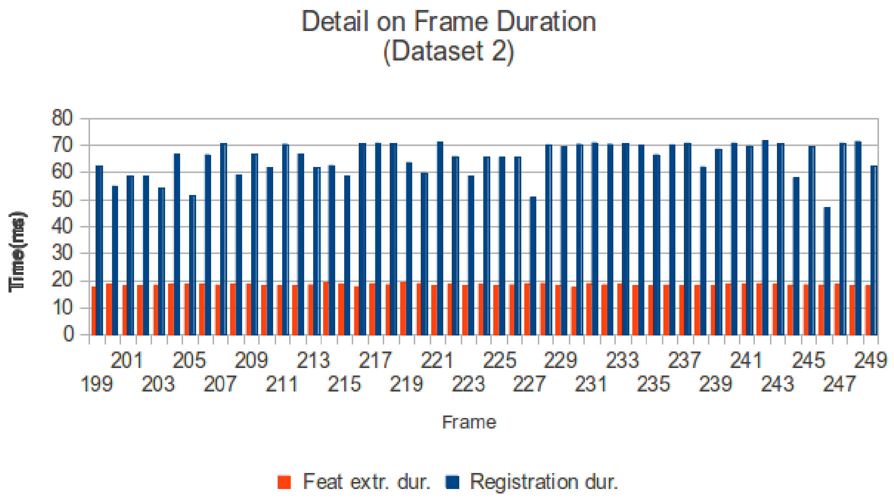

Figure 4.9: Detail on Visual Odometry processing time, dataset 2

To understand why there is variation in the registration procedure time, the number of features and time spent can be seen in the following figure:

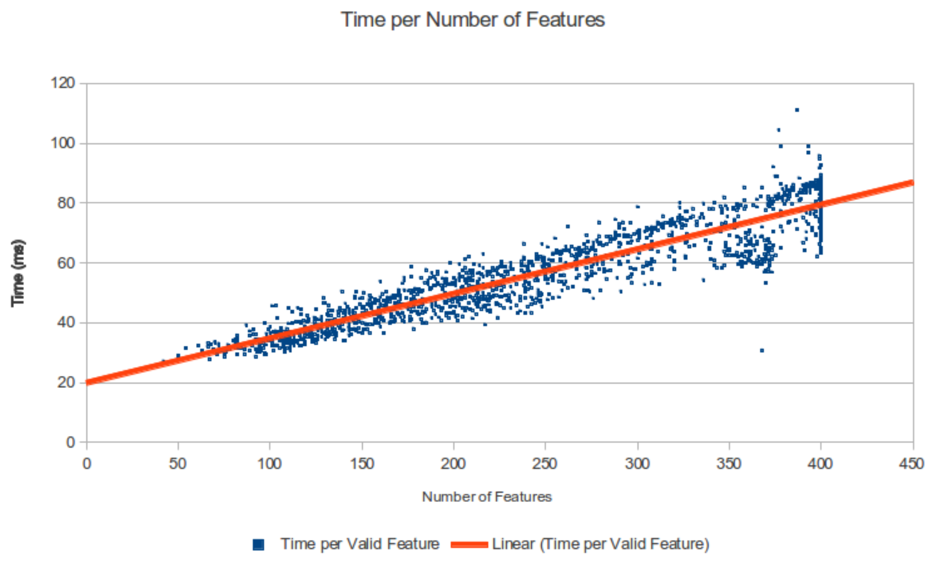

Figure 4.10: Relation between number of features and the time spent processing

From the Figure (4.10) it's possible to see that there is correlation between the two variables. The linear regression indicates the following equation:

$$Time(ms) = 0.14887 * \#Features + 19.9$$
$$R^2 = 0.8602 \tag{4.1}$$

From Figure (4.8) it is easy to see that the variations of processing time observed in Figure (4.6) are due to the variations of registration process where G-ICP is used to estimate the pose.The time variation that appears is related to the different number of features that were processed in each frame due to the continuous changes in the orientation of the quadcopter from and to areas that the RGB-D sensor cannot read.

On the second dataset, since the sensor is always facing the same direction, the number added features is more constant at a higher value than on dataset 1, creating a slightly larger processing time.

The feature extraction process remains fairly constant throughout both datasets since it mainly depends on the image size, which is always constant.

	Total Time Average	Total Time Variance	Feat. Extract Average	Feat. Extract Variance	Registration Average	Registration Variance
Dataset 1	54.34 ms	272.86 ms^2	18.50 ms	0.95 ms^2	38.78 ms	271.97 ms^2
Dataset 2	80.60 ms	56.26 ms^2	18.37 ms	0.85 ms^2	62.17 ms	56.31 ms^2

Table 4.1: Mean and Variation time of various parts of the Visual Odometry system

4.2.2 Size Complexity

Size complexity is highly correlated with time complexity since in most problems, the time the problem takes to solve it depends on the number of features. In this case, the more the features extracted, the stronger the estimate is. In that perspective, there is a trade-off between robustness and speed.

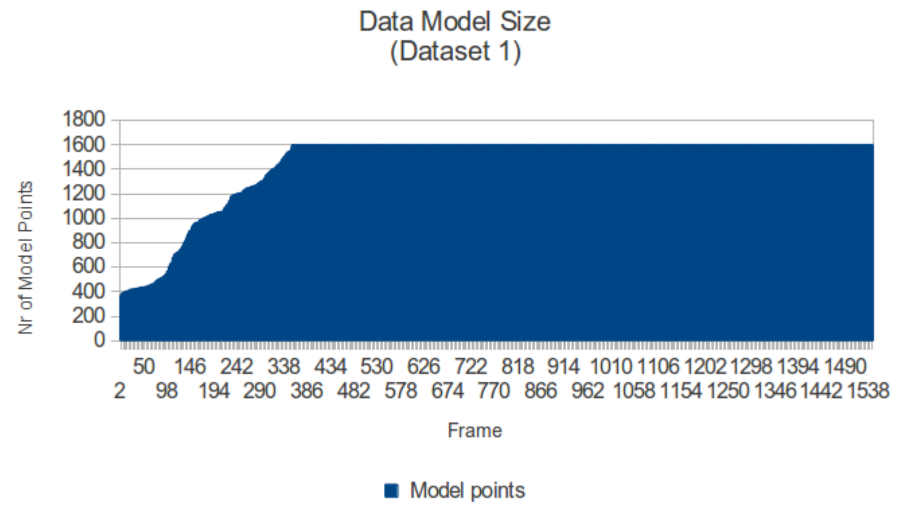

Figure 4.11: Number of features on the model to perform ICP in Dataset 1

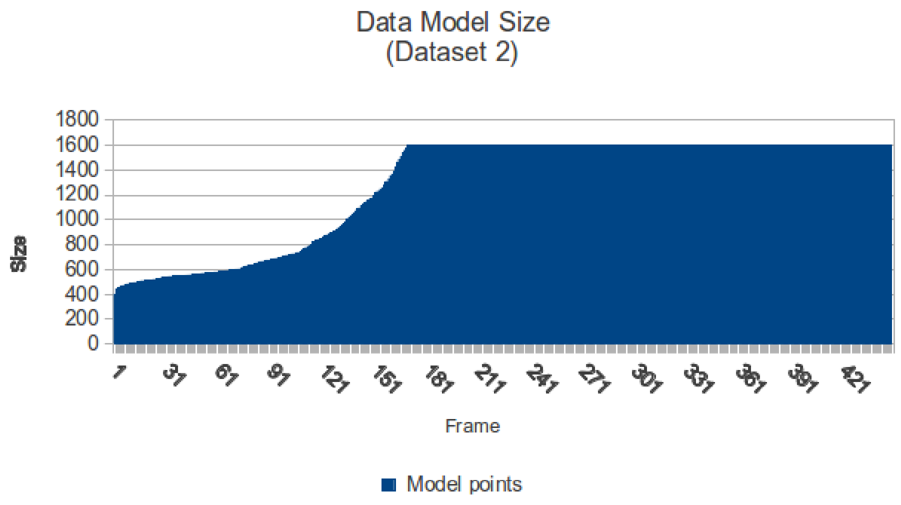

Figure 4.12: Number of features on the model to perform ICP in Dataset 2

Figure (4.11) and (4.12) shows the total size of the model where the G-ICP runs in order to get the pose estimate. There is a maximum number of allowed features to be inserted in the model in order to avoid letting it grow indefinitely. After the maximum is reached, the features are excluded using an FIFO, where the oldest features are removed first. The slope of the graph in the beginning is only dependent on the number of features added per frame.

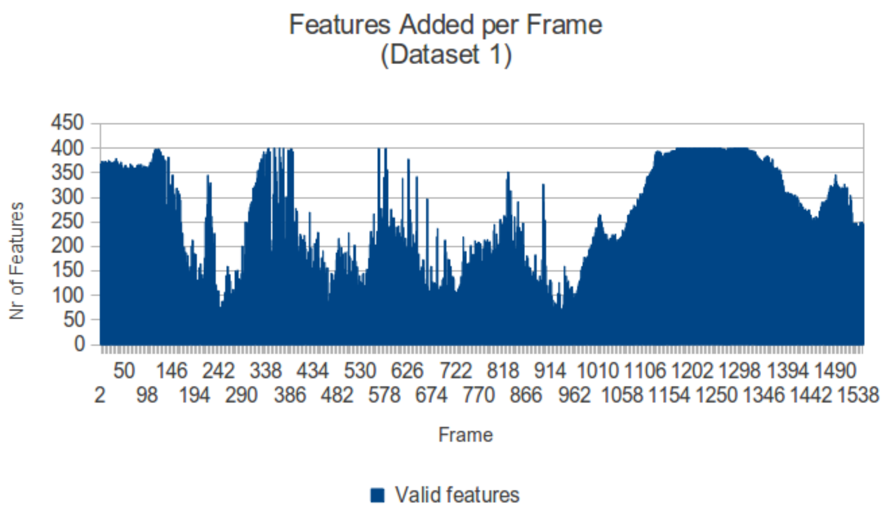

Figure 4.13: Number of features added per frame to the model in Dataset 1

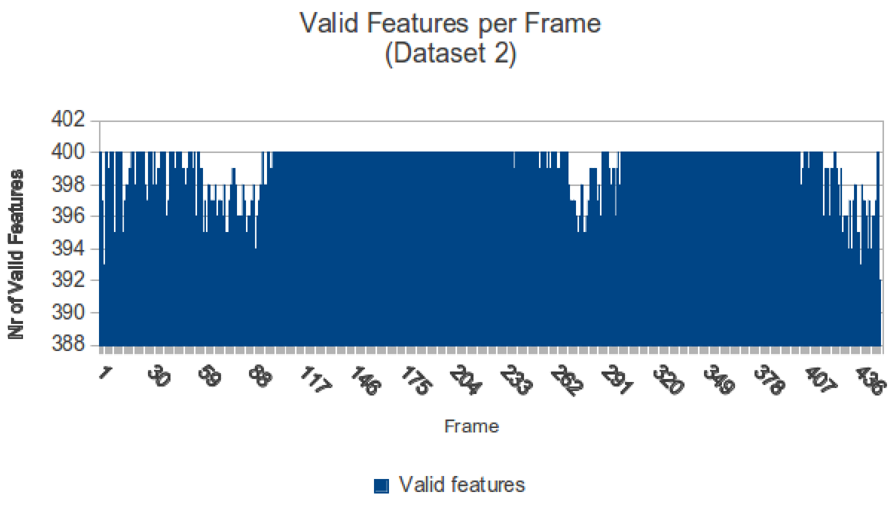

Figure 4.14: Number of features added per frame to the model in Dataset 2

Figure (4.13) and (4.14) show the number of features that are added per frame to the model.

Comparing Figure (4.13) with figure (4.6) it is possible to see that the frames that take the longest to process are the frames that have the biggest amount of features.

The fluctuation of features added per frame is usually related to filming areas where the RGB-D sensor cannot get readings which creates a fewer number of features to add. Note that while the frames with more features have a slower processing speed, the robustness of the estimate is usually stronger than one frame with less features.

4.3 Keyframe Mapping

Keyframe Mapping is the core application when performing SLAM. It allows to incrementally build a map and perform explicit loop closure when loops are found. Since the SLAM algorithm can be run in either real-time or off-line, with different end results, it is interesting to compare them since the results differ significantly with the method used. In the best case scenario, the online mapping will give the roughly the same result as the offline mapping.

4.3.1 Online Performance

In online mode, there are some performance cuts that are made in order to improve time performance, namely the number of maximum associations, the maximum distance from a query keyframe to the others and also the number of iterations that a pose estimation may run.

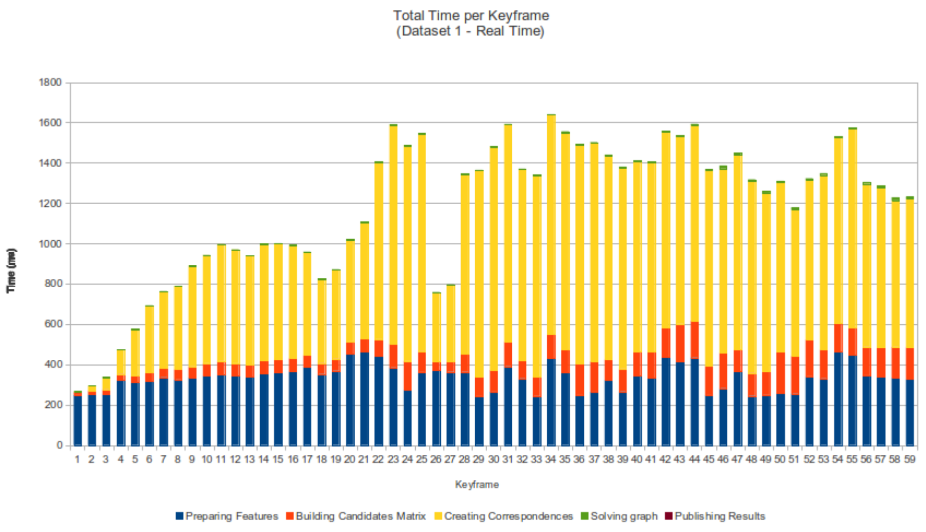

Figure 4.15: Mapping processing time for each frame in dataset 1, in real-time mode

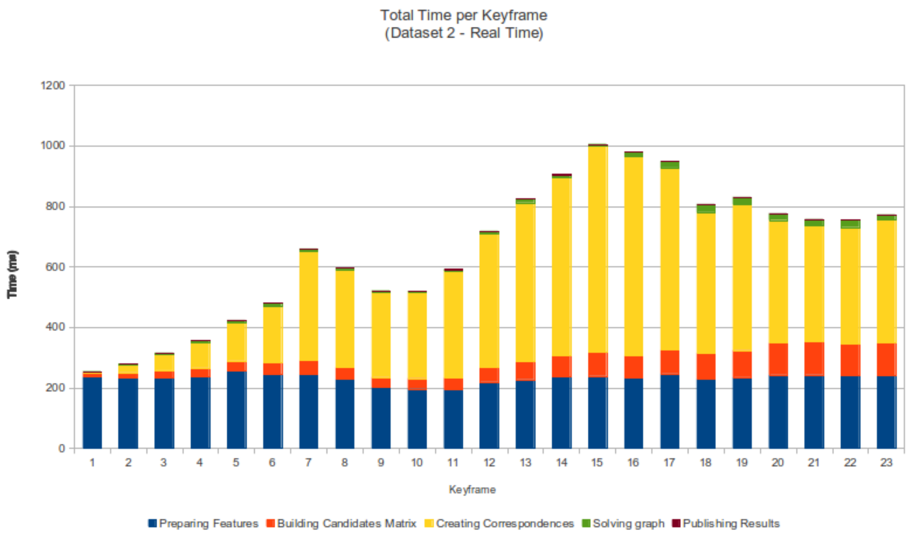

Figure 4.16: Mapping processing time for each frame in dataset 2, in real-time mode

Figures (4.15) and (4.16) show the time breakdown into different functions that each keyframe needs to run.

It is interesting to note that while feature extraction starts by using a large percentage of time per keyframe in the initial frames, it is quickly overrun by the creation of the correspondences to other keyframes. This latter process is a computational heavy processes where all the pose estimation is performed. In order to cap the maximum time performance, there was imposed a maximum number of associations as indicated in Chapter 4.1.

Building the candidate matrix is a process which time performance linearly varies along the number of keyframes. In order to avoid an unbounded growth, only the keyframes within a certain distance are considered for pose estimation.

The solving of the graph is a rather inexpensive procedure since most calculations were already made before, while creating the pose transform from all the candidate keyframes. There is only a slight increase in the processing time associated with the graph solving when there were added landmarks to the graph due to a small number of matches.

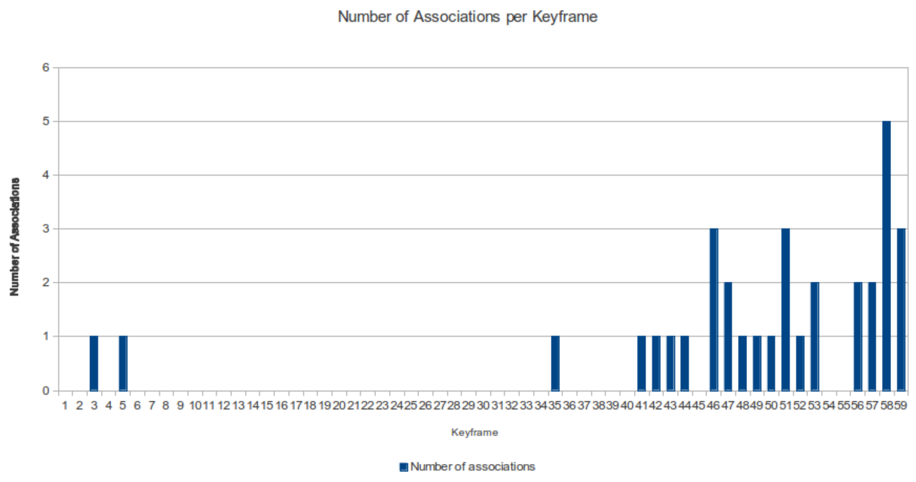

Figure 4.17: Number of Associations per frame in dataset 1, online

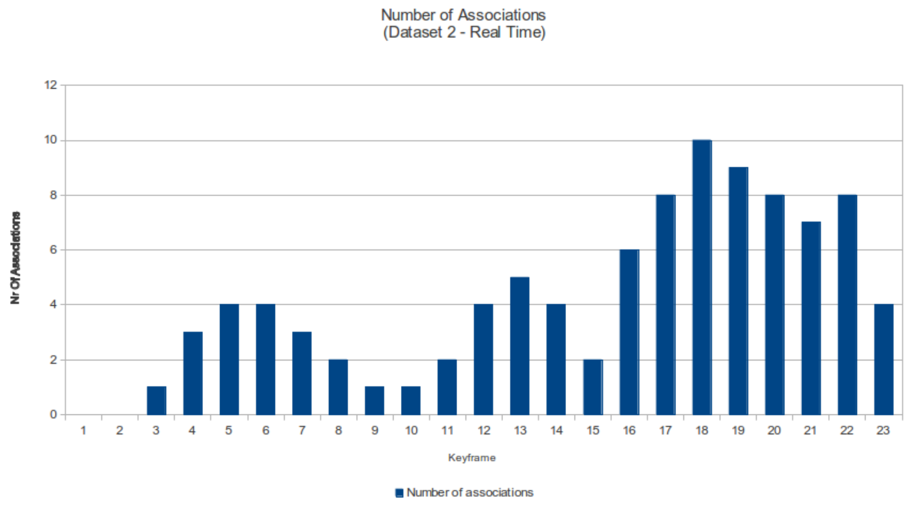

Figure 4.18: Number of Associations per frame in dataset 2, online

Since the dataset 1 in performed in a hostile environment for mapping, with large windows with sunlight and the quadcopter was performing fast movements, the low number of associations per keyframe pictured in Figure (4.17) are justified. The results only begin being acceptable in the last frames where the robot revisits an already mapped area and is able then to perform some pose estimations.

In dataset 2, since the environment was better controlled, it is easy to see from the graph on Figure (4.18) that there was a successful online mapping of the area since most keyframes have connections. When the slope is negative the robot is exploring new areas and when the slope is positive or constant the robot is revisiting areas already seen.

4.3.2 Offline Performance

Regarding offline mapping it is useful to note that a keyframe can be connected to any other and not only those within a range and also the threshold of the number of iterations on the pose estimation procedure is put at a higher value. An advantage of the offline method versus the online method is that in offline mode, a keyframe that perform a connection to a posterior keyframe and not only with one that happened before.

While comparing the two methods against the same dataset it is possible to see that the offline mapping method has more keyframes that the only method. This happens because offline mapping first gathers the frames necessary running only visual odometry and this allows to get the real distance between keyframes near the minimum distance specified. Online mapping has to wait for a frame to be processed before saving the next frame so when the vehicle is moving too fast, the computational speed may not be able to catch up and frames that would be selected as keyframes in the offline mode are thus discarded.

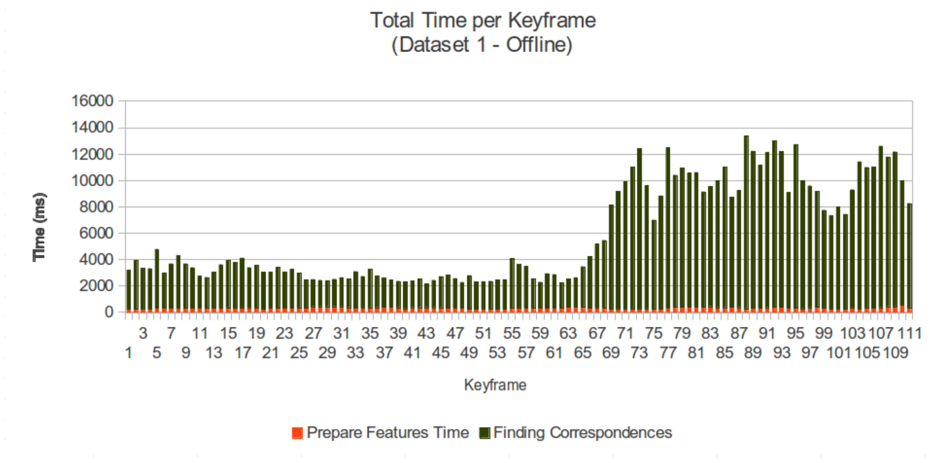

Figure 4.19: Mapping processing time for each frame, in dataset 1, offline mode

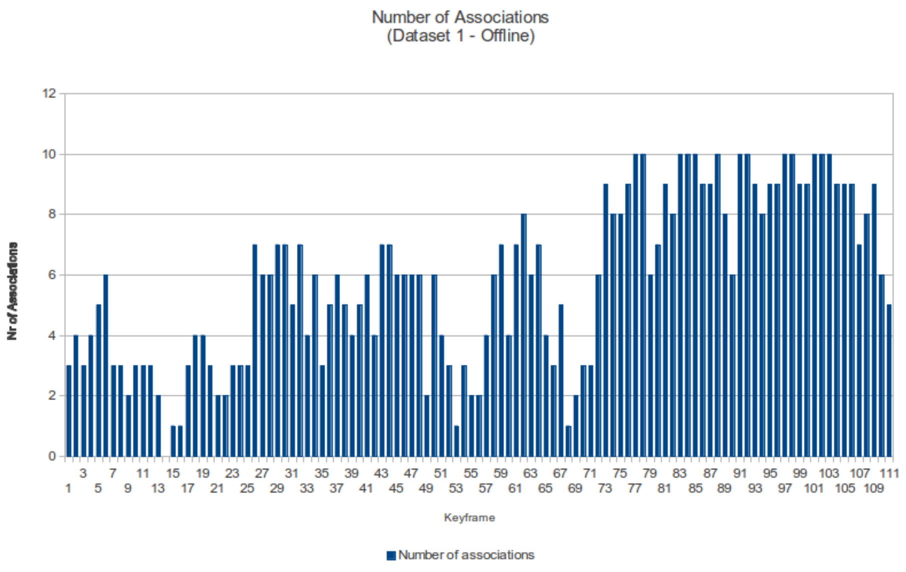

Figure 4.20: Number of Associations per frame in dataset 1, offline

Comparing Figure (4.20) with Figure (4.17) it is possible to compare the number of associations performed with both methods. It is obvious to notice that the offline method performed much better in term of number of associations. Each keyframe takes more time to run, as pictured in Figure (4.19), getting values as high as 12 seconds per frame.

It is also possible to see that the online mode has fewer keyframes than the offline mode. This difference is due to the fact that the computational requirements for each keyframe analysis take more time than the time the quadcopter takes to travel the defined distance that separates the keyframes. Furthermore, the number of associations among keyframes has a bigger value with the offline mode, which then provides a stronger pose estimate. This last happens because the offline mode has all keyframe available to compare while the online mode only has the past keyframes.

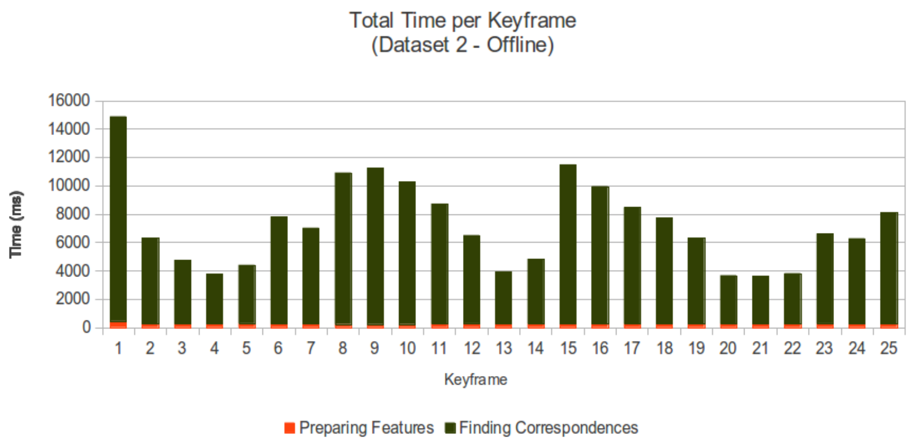

Figure 4.21: Mapping processing time for each frame, in dataset 2, offline mode

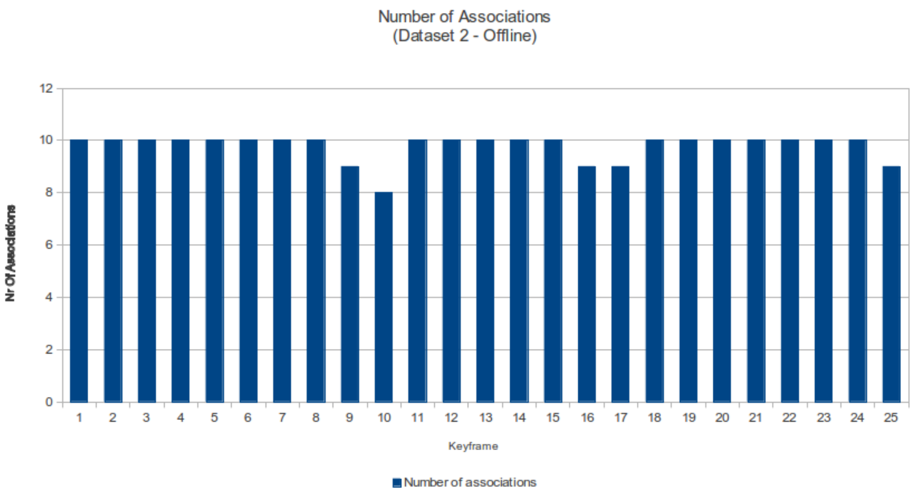

Figure 4.22: Number of Associations per frame in dataset 2, offline

In dataset 2, comparing Figure (4.18) with Figure (4.22), it can be concluded that in fact the number of associations higher per frame which results in a map built with higher certainty. The time spend per frame, shown in Figure (4.21), is also much higher than in online mode.

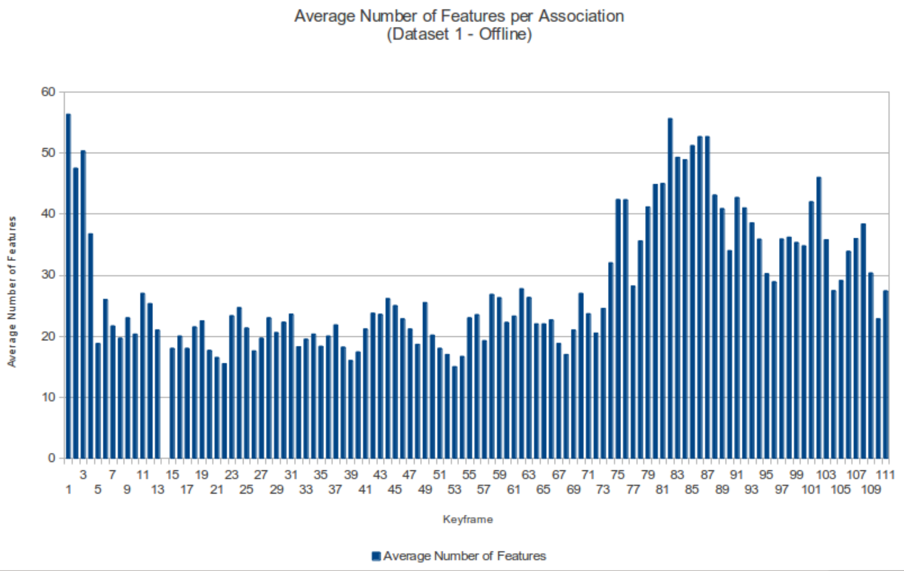

Figure 4.23: Number of features per association, dataset 1, offline

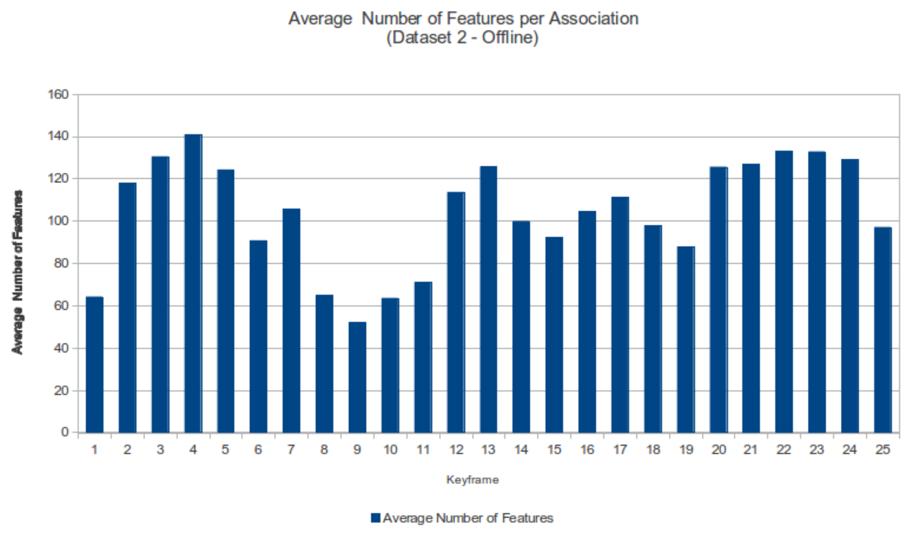

Figure 4.24: Number of features per association, dataset 2, offline

Analyzing the average number of features per association, pictured in Figures (4.23) and (4.24), it is possible to see that the average number of features per association in the first dataset is lower than on the second dataset. This happends because the large windows on the first dataset create large areas of the image that cannot be analyzed to depth information and therefore the number of useful features per frame is much lower.

4.3.3 Trajectory Estimate

Due to the publishing of the estimated pose values over time, it is possible to draw the estimated trajectory that the robot performed during the visits to the dataset locations. To facilitate this task, the RVIZ tool from the ROS package is used which provides out of the box plotting tools for robot poses and more. This allows for an easy interface with the user which enables to see the robot's pose in a 3D space.

This way is possible to have a visual comparison between the results of both mapping execution modes.

When visualizing the 3D map it is necessary to have attention to the following details:

- Red Arrow with a blue number - Pose of the keyframe indicated by the number
- Green line - Trajectory of the robot
- Black sphere - Position of tracked robots
- Orange line - Trajectory of tracked robots

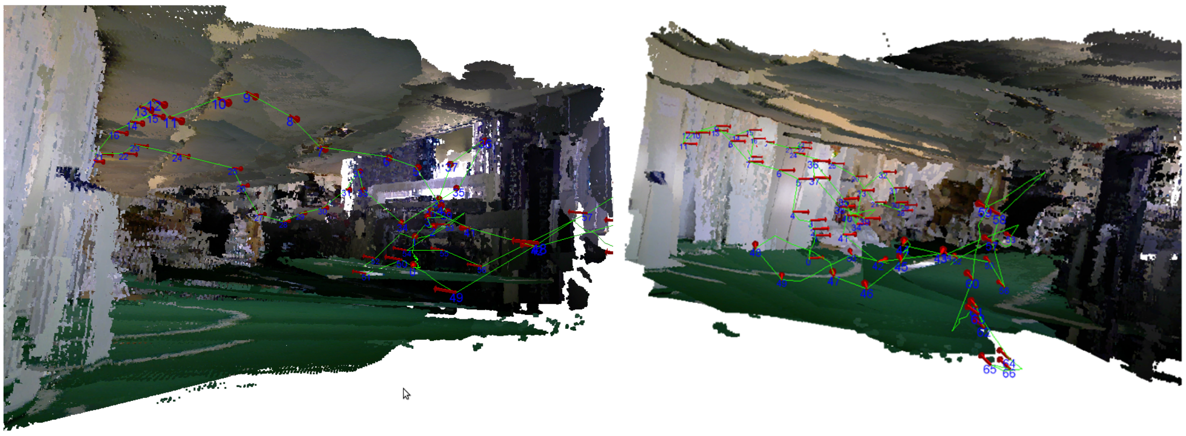

Figure 4.25: Robot trajectory estimation in online mode in dataset 2, visualised with RVIZ

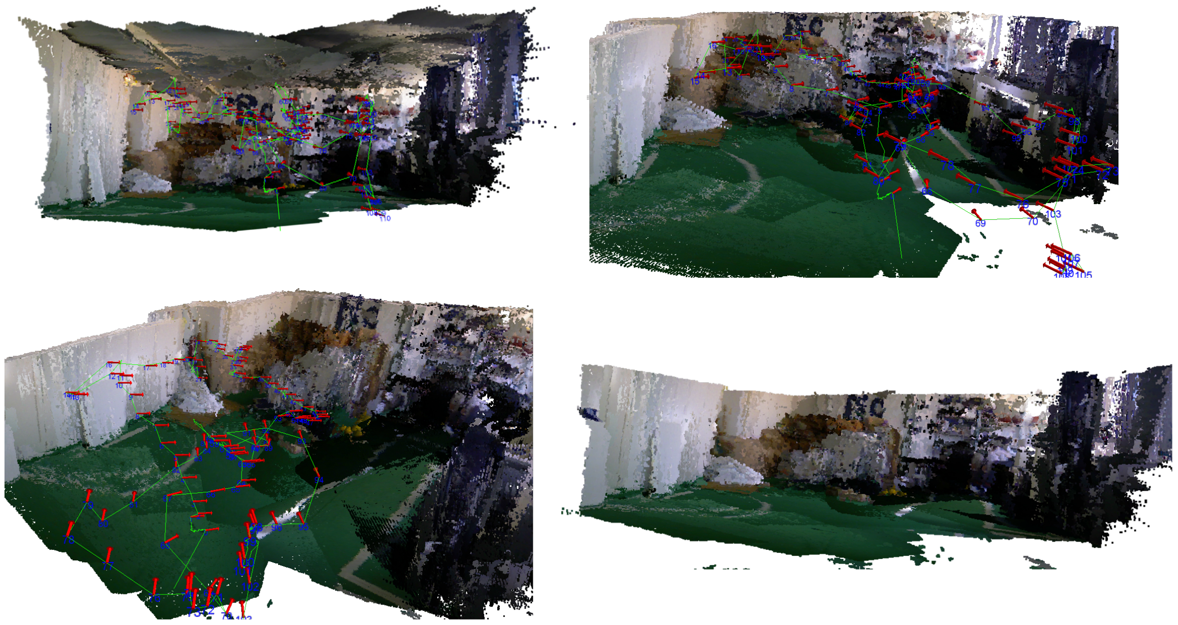

Figure 4.26: Robot trajectory estimation in offline mode in dataset 2, visualised with RVIZ

Regarding dataset 1, it is possible to see that the outcome using online mapping, shown in Figure (4.25), was not as good as it may be needed for a real robotic scenario. It is possible to see that some of the keyframes were badly estimated and as such there are different layers of ground and ceiling at different heights and an overall weak performance. Using the offline execution, the overall outcome has a much higher quality in terms of good pose estimations. It is possible to see that most keyframes have a correct pose estimation and the back projection of the map gives a correct result. One thing that may be confused with bad results are the different changes in color of concurrent walls/floor. This phenomenon happens due to the automatic aperture size regulation of the camera, as explained in the Chapter 3.1.3.

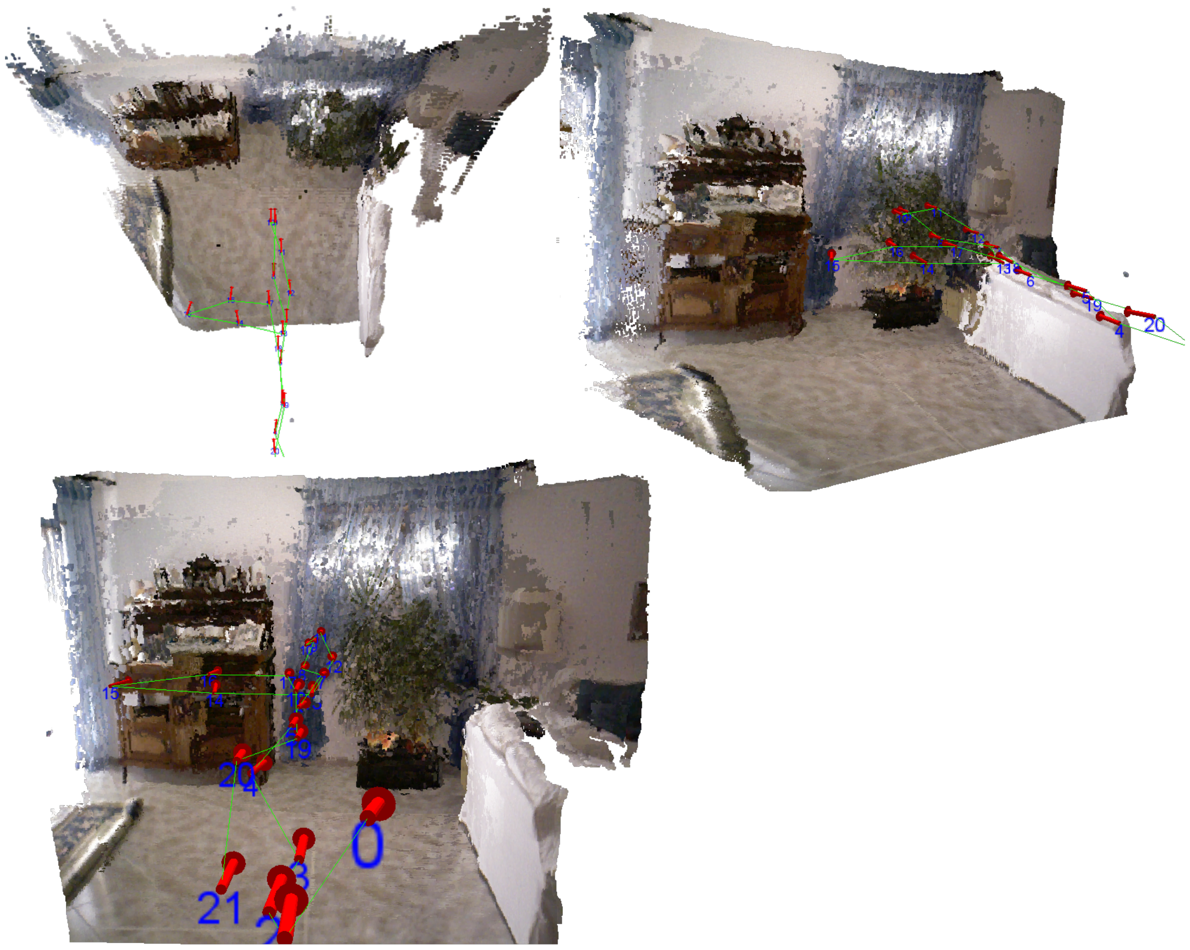

Figure 4.27: Robot trajectory estimation in online mode in dataset 2, visualised with RVIZ

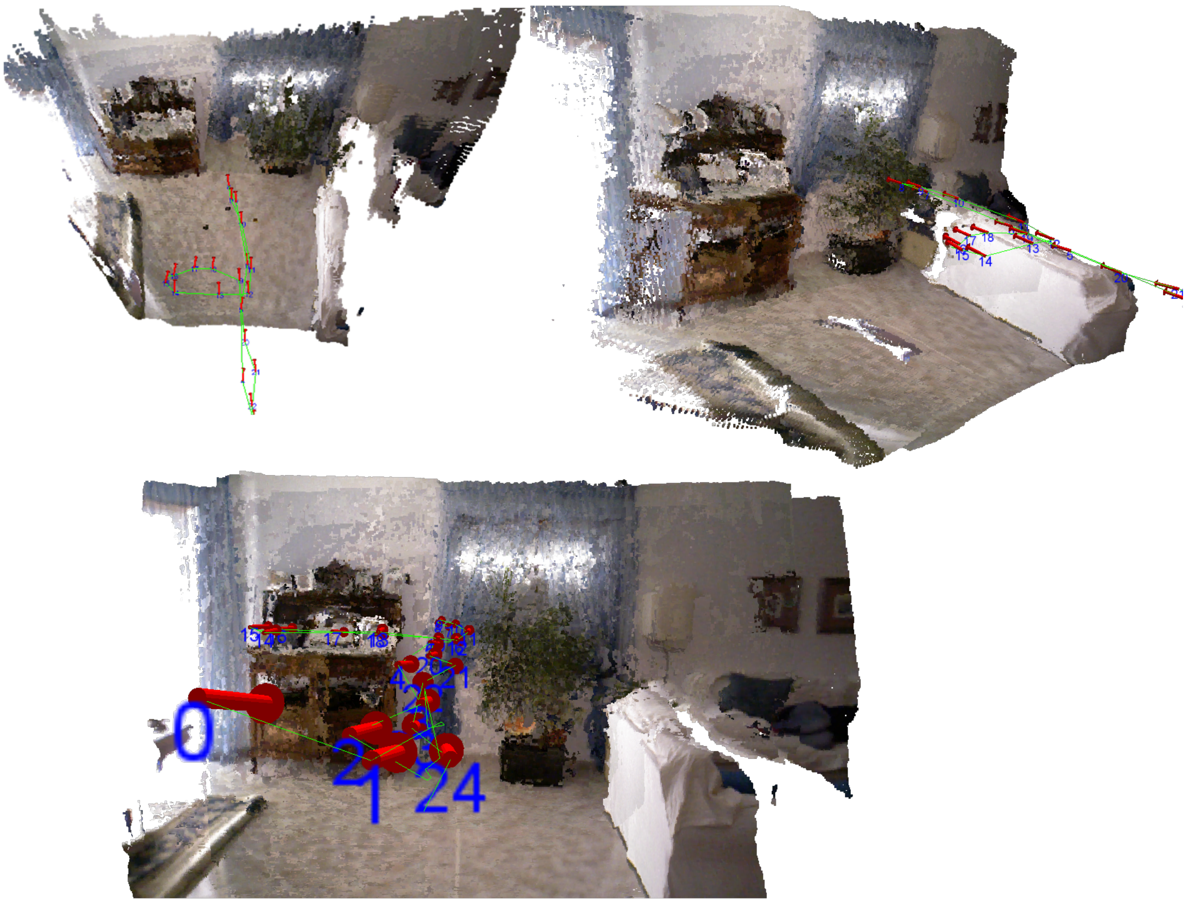

Figure 4.28: Robot trajectory estimation in offline mode in dataset 2, visualised with RVIZ

In regard to dataset 2, comparing both execution modes in Figure (4.27) and Figure (4.28) it is possible to see that both results are very similar. This was an already foreseen outcome since there were a significant number of association among keyframes in online execution, as seen in Figure (4.18). From the top view, it is also possible to see easily the trajectory described in the dataset introduction which further assures the result is correct.

Interesting keyframes that can be compared to the "groundtruth" are keyframe #9 (#11 in online) that is positioned at the tip of the path, keyframe # 15 that indicates the left extremity and finally, the keyframe # 24 which is the end point.

Location	"Groundtruth" x;y	Offline x;y	Offline Euclidean Error	Online x;y	Online Time Euclidean Error
tip	3 ; 0	2.9 ; 0.33	0.35	3.01 ; 0.37	0.37
left	2 ; 1	1.88 ; 0.98	0.12	1.94 ; 1.08	0.1
end	0 ; 0	0.17 ; -0.26	0.31	0.34 ; -0.17	0.38

Table 4.2: Comparison to groundtruth

As seen in the Table (4.2), there are some differences compared to the groundtruth but since the the groundtruth was obtained by following a path it should just be used as an indicator and not taken strictly. Note also that since the path was only to give guidance on the x and y axis, it doesn't make sense to use any ground truth in the z axis. Another important aspect to consider is that since the map is constituted by keyframes, there may not be a keyframe exactly at the location of analysis thus increasing the error. The keyframes selected were the nearest to the points of reference.

4.4 Localization

After the mapping has been done, it is possible to use that map to left-localize with any robot that has a RGB-D sensor or a simple calibrated RGB camera.

In this scenario, while time is important for real time performance, since the localization intent is to give discrete pose corrections, the time complexity isn't as crucial as in mapping where a delay may seriously impact the final result. While self-localizing, if one frame takes more time to analyze than predicted, the only drawback is that the pose correction lags by that amount, while in mapping the lag can create a great drop in possible matches which would degrade the quality of the map.

As in mapping, due to the lack of ground truth, the method to compare the performance is by analyzing the optimization solver cost function error.

It is important to notice that Visual Odometry is not running while performing the RGB localization since the depth data is not available from a simple RGB camera. This way, when estimating the robot trajectory with the RGB-D sensor, there is already a prior that is added to the graph before the optimization.

While self-localizing, there is also the interesting feature that allows to track other vehicles identified with an ArUco marker.

Note while the full localization algorithm consists in both visual odometry and pose corrections, in this chapter only the pose correction performance is being analyzed since the visual odometry was already analyzed in section 4.2.

4.4.1 RGB-D Localization

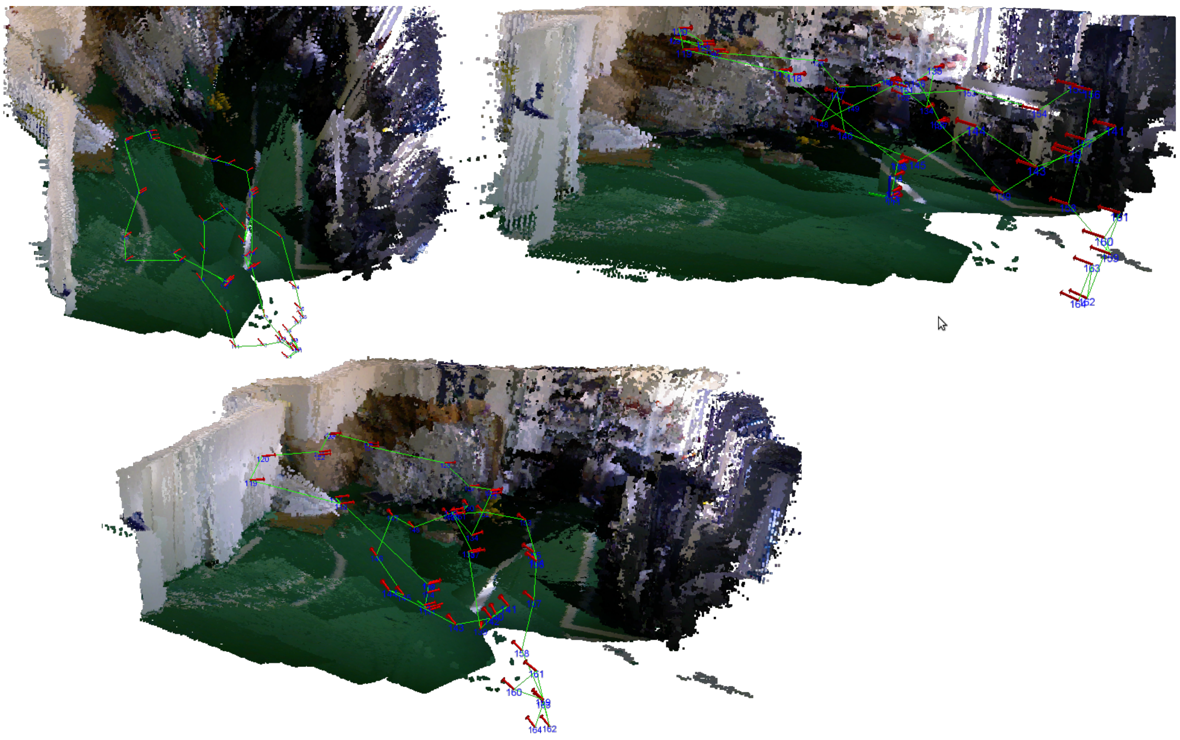

Figure 4.29: RGB-D Localization on Dataset 1

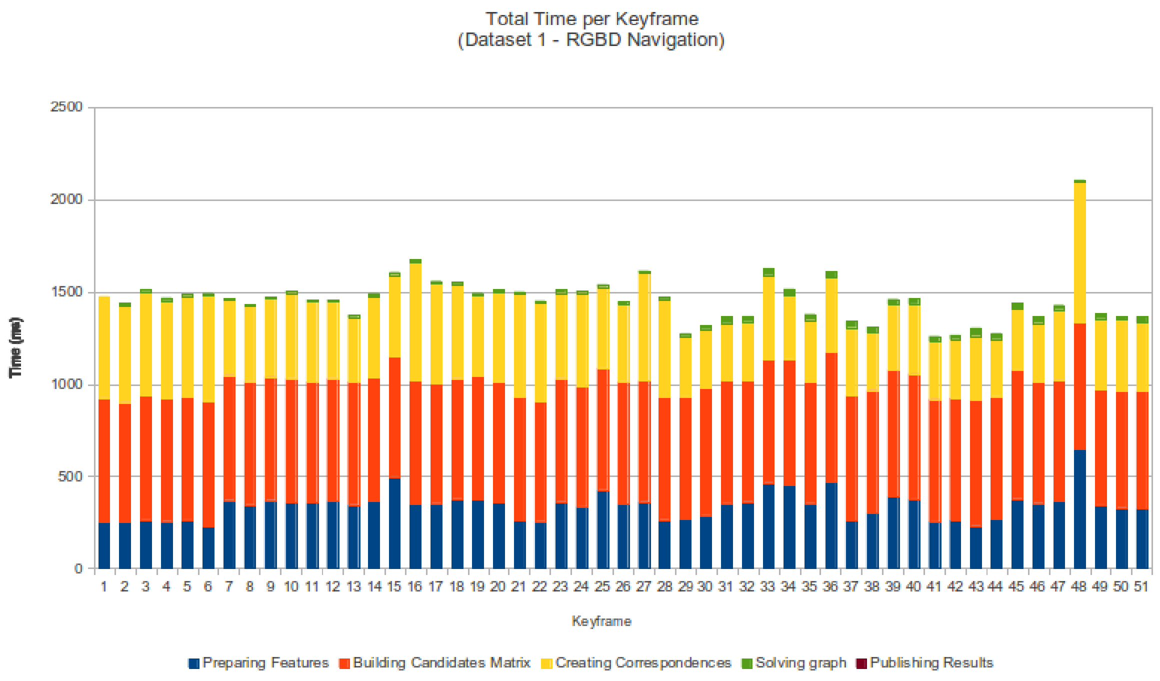

Figure 4.30: RGB-D Localization on Dataset 1

The localization of dataset 1 was based on the same input that was used to build the map. As such the trajectory that is built here is similar to the trajectory in figure 4.26. In this dataset no tracking of other vehicles was made so the localization algorithm only indicates the position of the RGB-D sensor.

In figure 4.30, it is possible to see that the time taken to analyze each frame is approximately equal since the complexity depends on the map size and on the number of associations made per frame.

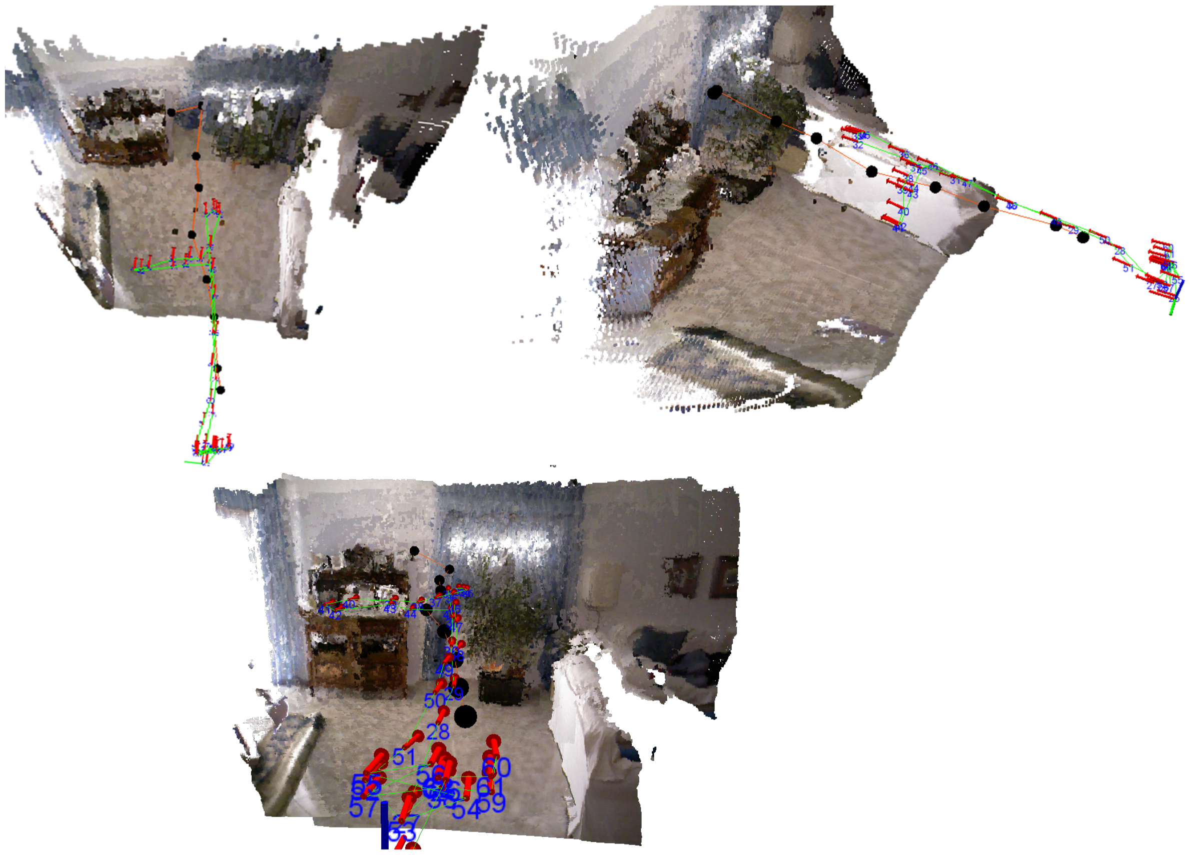

Figure 4.31: RGB-D Localization on Dataset 2

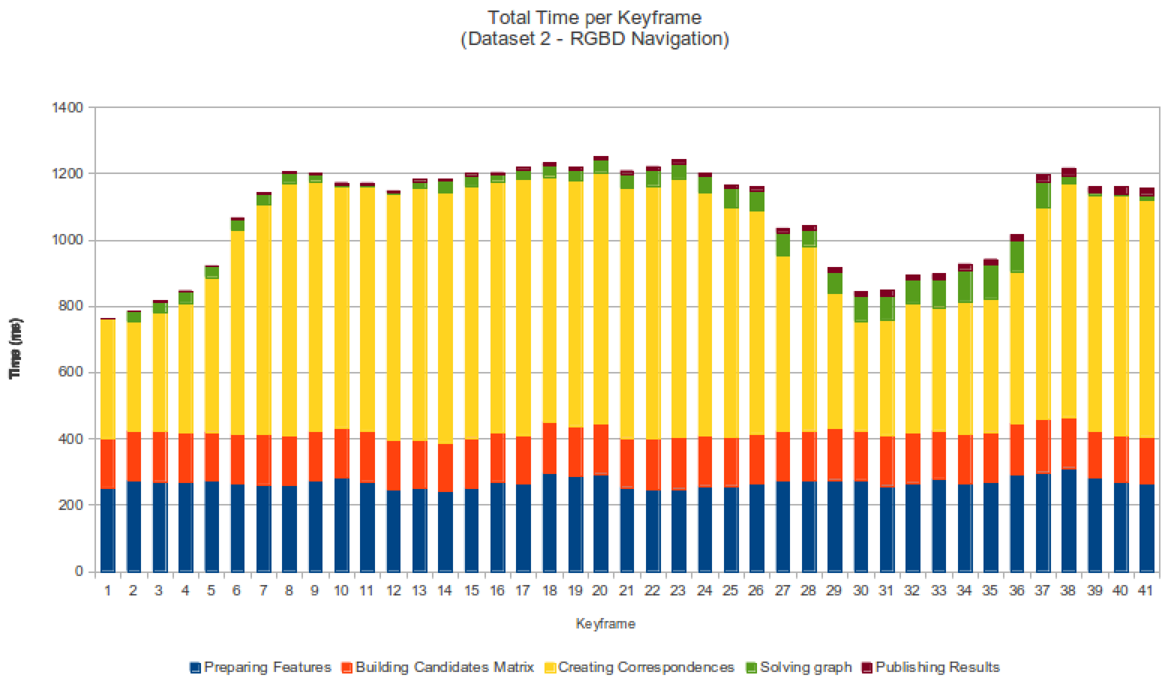

Figure 4.32: RGB-D Localization on Dataset 2

In the dataset 2 localization scenario, as previously explained, an ArUco marker was added so that the RGB-D sensor could track it. It is possible to see that the sensor performs the same maneuver as in the mapping procedure and acceptably track the other robot position (indicated with black spheres).

The processing time for the keyframes varies slightly more than in the dataset 1 example due to more variation on the number of keyframe associations that could be created.

	# Poses	# Nodes	# Factors	Error per Factor	Total Squared Error
Dataset 1	51	162	1045	0.09 m	8855.8 m^2
Dataset 2	41	107	631	0.358 m	50968.9 m^2

Table 4.3: Number of objects in RGB-D localization graph

By the table 4.3 it is possible to see that the error per factor in the first dataset is very small while on the second dataset already has a more significant value. This is due to the fact that the mapping and localization were done with the same dataset and method. In the second dataset, while the method was the same, the localization video is different from the mapping video so the extra moving object create extra error on some frames.

Keyframe	"Groundtruth" x;y	Graph RGB-D x;y	Euclidean Error
tip	3 ; 0	3.19 ; 0.32	0.37
left	2 ; 1	2.03 ; 0.99	0.03
end	0 ; 0	0.15 ; 0.01	0.15

Table 4.4: Comparison to "groundtruth" in RGB-D localization

Note that the "groundtruth" values are only an indication since the sensor navigation was done handheld and the keyframe selected to measure was the closest to the reference point.

4.4.2 RGB Localization

Localization can also be performed with a simple RGB camera which allows for a huge variety of application using smaller, cheaper robots.

Since the method used to estimate the pose between the RGB and RGB-D is very different, it is interesting to see how the results differ.

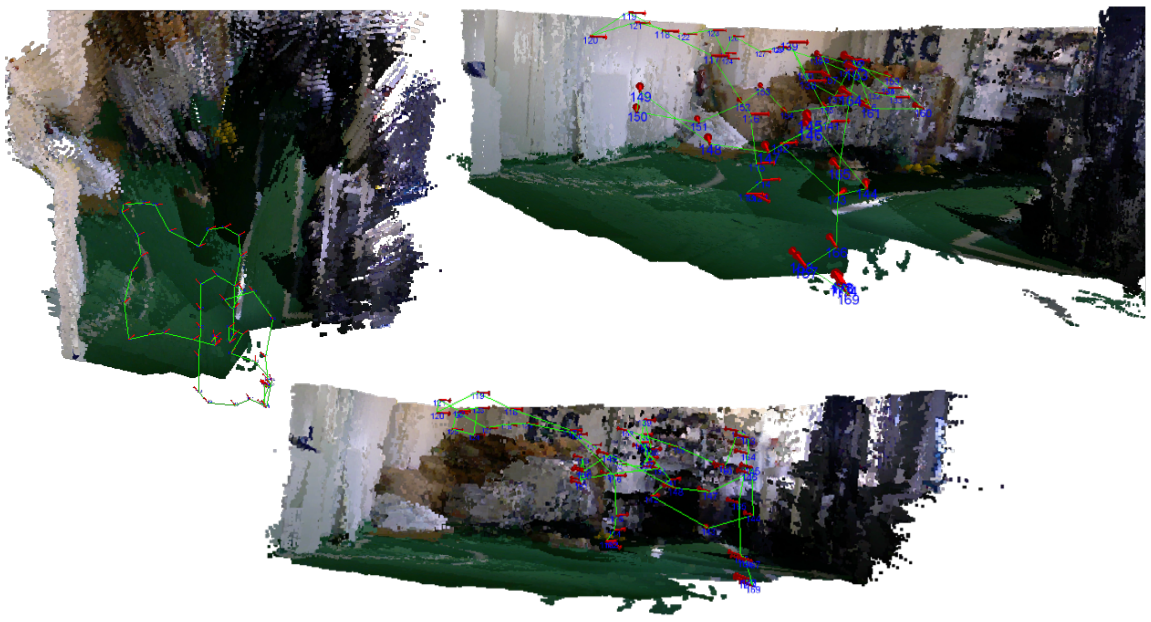

Figure 4.33: RGB Localization on Dataset 1, visualized with RVIZ

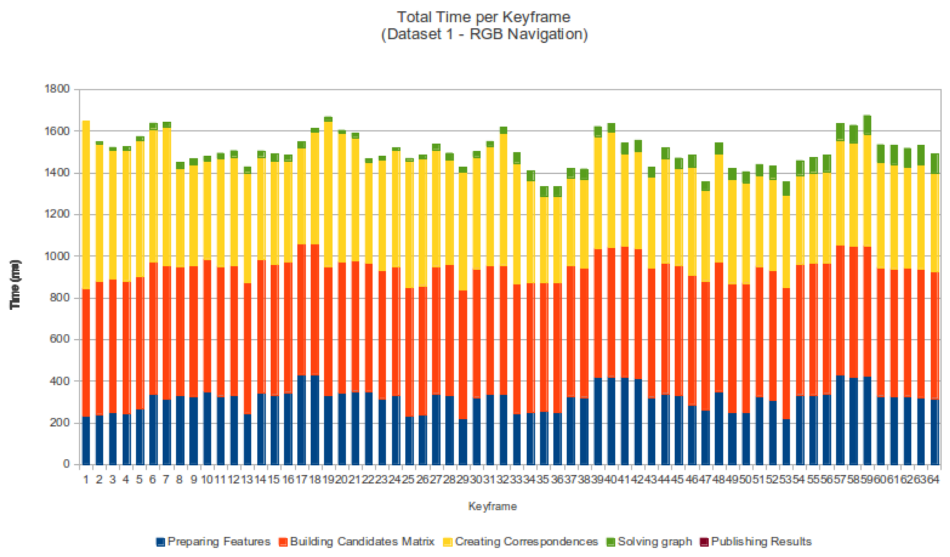

Figure 4.34: RGB Localization on Dataset 1

Analyzing figure 4.33 and comparing it with 4.29 it is possible to see that the trajectory is roughly the same. Analyzing 4.34 it is noticeable that there is a greater number of keyframes in the RGB localization. This happens because a keyframe that has no association with others is not added while self-localizing and since in RGB localization the number of valid keypoints per frame is higher, since the keypoints for which there would be no use in a RGB-D localization due to lack of depth information can be used without any problem in a RGB localization scenario.

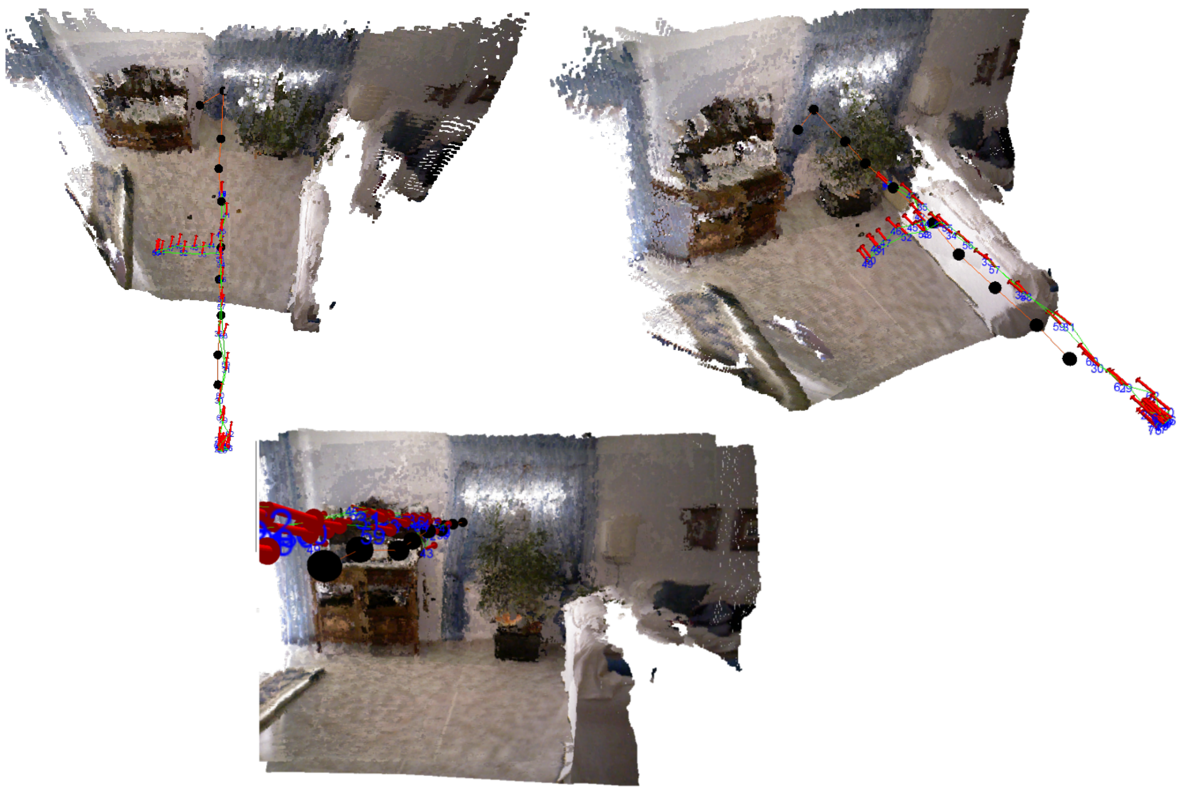

Figure 4.35: RGB Localization on Dataset 2, visualized with RVIZ

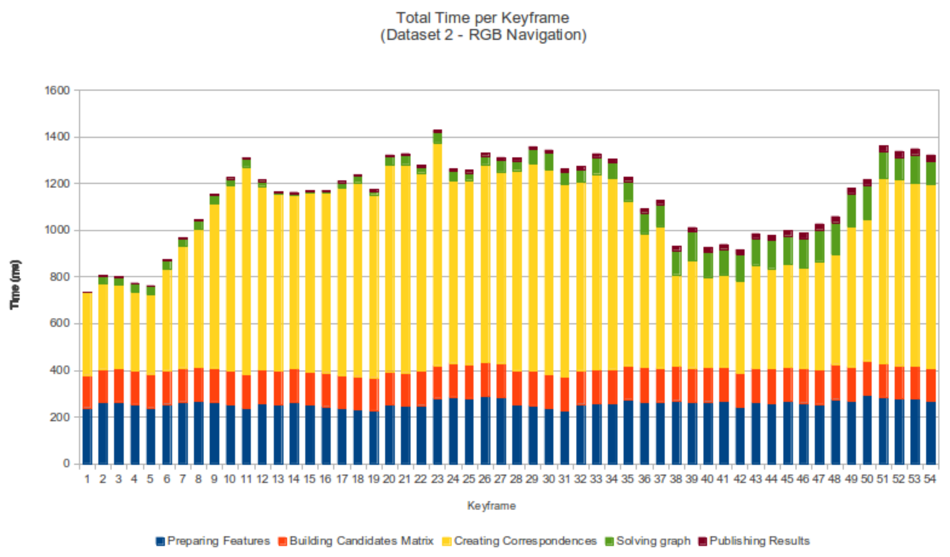

Figure 4.36: RGB Localization on Dataset 2

Once again the results estimated with RGB-D show in figure 4.31 are very similar to the results observed in the RGB localization show in figure 4.35.

	# Poses	# Nodes	# Factors	Error per Factor	Total Squared Error
Dataset 1	64	175	1382	0.145 m	40598.5 m^2
Dataset 2	54	133	810	0.166 m	18170.3 m^2

Table 4.5: Number of objects in RGB localization graph

Keyframe	"Groundtruth" x;y	Graph RGB x;y	Euclidean Error
tip	3 ; 0	2.85 ; 0.29	0.32
left	2 ; 1	2.1 ; 0.95	0.11
end	0 ; 0	0.16 ; -0.27	0.31

Table 4.6: Comparison to "groundtruth" in RGB localization

Once again, the values in the "groundtruth" are only an indication since the sensor has handheld while performing the self-localization and that it was chosen the nearest keyframe to the reference point which may not be exactly at the point itself.

Overall the rates at which the different modules run are:

Module	Rate
Visual Odometry	12-16Hz
Real Time SLAM	0.75-1.25Hz
RGB-D Localization	0.66-1Hz
RGB Localization	0.66-1Hz

Table 4.7: System's Module Performance Rates

5
Conclusions and Future Work

In this thesis RGB-D sensors were shown to provide the versatility necessary to develop a full 6D SLAM algorithm based only on the sensor data. While the visual odometry suffers from the typical drift, noticed specially in fast rotational movements due to blur, the pose correction techniques implemented can easily overcome that obstacle and provide a reliable pose estimation along time.

A visual SLAM algorithm was developed that can both run online and offline. While the online mode has the obvious advantage of fast results, in the presence of environments with a lack of unique features, the results may be sub-optimal. The offline algorithm offers the robustness that the online algorithm lacks and provides ann optimal map in most conditions, at the trade-off of speed.

While the usage of graph-based solutions is common to the SLAM problem, by implementing incremental updates with iSAM to the graph-solving algorithm it is then possible to run online optimization-based SLAM algorithms. Due to the versatility of iSAM, it can be used both incrementally and in batch-solving which allows for only one solver for all depicted SLAM and navigation problems reached in this thesis.

It can be noticed from the given results that both the RGB-D and RGB pose correction algorithms give similar results which allows for heterogeneity in robot builds, once the map has been constructed.

Finally and as future work, while some issues were approached, others were left unaddressable and could be developed for a more complete search and rescue environment, such as:

- Usage of both RGB and RGB-D features when both are available for a more robust approach to moving objects
- Integration with other sensors of the robots such as an IMU, which could help as a prior for pose estimation and thus avoiding the motion blur problem.
- Experimentation with different feature extractors for faster performance, such as ORB[47] or different feature descriptors, such as FREAK[48]
- Using other RANSAC algorithms in pose estimation [49]

Bibliography

[1] Y. Cao, A. Fukunaga, A. Kahng, and F. Meng, “Cooperative mobile robotics: antecedents and directions,” in *Intelligent Robots and Systems 95. ’Human Robot Interaction and Cooperative Robots’, Proceedings. 1995 IEEE/RSJ International Conference on*, 1995, pp. 226–234 vol.1.

[2] R. Smith, M. Self, and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, 1987, vol. 4, pp. 850–850.

[3] N. Gordon, D. Salmond, and A. F. M. Smith, “Novel approach to nonlinear/non-gaussian bayesian state estimation,” *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2, pp. 107–113, 1993.

[4] F. Lu and E. Milios, “Globally consistent range scan alignment for environment mapping,” *Autonomous Robots*, vol. 4, pp. 333–349, 1997.

[5] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg, “Hierarchical optimization on manifolds for online 2d and 3d mapping,” *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 273–278, 2010.

[6] J. Castellanos, J. M. M. Montiel, J. Neira, and J. Tardos, “The spmap: a probabilistic framework for simultaneous localization and map building,” *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 5, pp. 948–952, 1999.

[7] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, “Simultaneous localization and mapping with sparse extended information filters,” *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, August 2004.

[8] D. Hähnel, D. Fox, W. Burgard, and S. Thrun, “A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements,” in *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[9] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges,” in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico: IJCAI, 2003.

[10] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 34–46, 2007.

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, “isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering,” *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3281–3288, 2011.

[12] M. Kaess and F. Dellaert, “Covariance recovery from a square root information matrix for data association,” *Journal of Robotics and Autonomous Systems (RAS)*, vol. 57, pp. 1198–1210, Dec. 2009.

[13] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” *IEEE Trans. on Robotics (TRO)*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.

[14] F. Dellaert and M. Kaess, “Square Root SAM: Simultaneous localization and mapping via square root information smoothing,” *Intl. J. of Robotics Research (IJRR)*, vol. 25, no. 12, pp. 1181–1204, Dec. 2006.

[15] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments,” *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, Feb. 2012.

[16] F. M. O. d. Jesus, *Simultaneous localization and mapping using vision for search and rescue robots*. Instituto Superior Técnico, 2012.

[17] I. Dryanovski, W. Morris, R. Kaushik, and J. Xiao, “Real-time pose estimation with rgb-d camera,” in *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, 2012, pp. 13–20.

[18] I. Dryanovski, R. Valenti, and J. Xiao, “Fast visual odometry and mapping from rgb-d data,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 2305–2310.

[19] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” *IEEE Trans. on Robotics (TRO)*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.

[20] F. Dellaert and M. Kaess, “Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing,” *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, Dec. 2006.

[21] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[22] G. Golub and C. Van Loan, *Matrix Computations*, ser. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996.

[23] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[24] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[25] S. Gumhold, X. Wang, and R. Macleod, “Feature extraction from point clouds,” in *In Proceedings of the 10 th International Meshing Roundtable*, 2001, pp. 293–305.

[26] H. Bay, T. Tuytelaars, and L. V. Gool, “Surf: Speeded up robust features,” in *In ECCV*, 2006, pp. 404–417.

[27] a. Neubeck and L. Van Gool, “Efficient Non-Maximum Suppression,” *18th International Conference on Pattern Recognition (ICPR’06)*, pp. 850–855, 2006.

[28] J. Shi and C. Tomasi, “Good features to track,” in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR ’94., 1994 IEEE Computer Society Conference on*, 1994, pp. 593–600.

[29] J. Rekimoto, “Matrix: a realtime object identification and registration method for augmented reality,” in *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, 1998, pp. 63–68.

[30] H. Kato and M. Billinghurst, “Marker tracking and hmd calibration for a video-based augmented reality conferencing system,” *Augmented Reality, 1999. (IWAR ’99) Proceedings. 2nd IEEE and ACM International Workshop on*, pp. 85–94, 1999.

[31] A. Segal, D. Haehnel, and S. Thrun, “Generalized-ICP.” *Robotics: Science and Systems*, 2009.

[32] A. Censi, “An ICP variant using a point-to-line metric,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008.

[33] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[34] O. Chum and J. Matas, “Optimal randomized RANSAC,” *Pattern Analysis and Machine Intelligence*, vol. 30, no. 8, pp. 1–11, 2008.

[35] S. Umeyama, “Least-squares estimation of transformation parameters between two point patterns,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 4, pp. 376–380, 1991.

[36] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[37] ASUS. Xtion pro live specifications. [Accessed September 2013]. [Online]. Available: http://www.asus.com/Multimedia/Xtion_PRO_LIVE#specifications

[38] UAVision. Quadcopter ux-4001. [Accessed September 2013]. [Online]. Available: http://uavision.com/index.php?option=com_content&view=article&id=163&Itemid=183&lang=pt

[39] R. Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Pub Limited, 2011.

[40] H. Zhang, B. Li, and D. Yang, “Keyframe detection for appearance-based visual slam,” *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2071–2076, 2010.

[41] G. Klein and D. W. Murray, “Improving the agility of keyframe-based SLAM,” in *Proc 10th European Conf on Computer Vision, Marseille, France*, 2008.

[42] A. Girgensohn and J. Boreczky, “Time-constrained keyframe selection technique,” in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1, 1999, pp. 756–761 vol.1.

[43] T. Liu and J. R. Kender, “Optimization algorithms for the selection of key frame sequences of variable length,” in *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ser. ECCV ’02. London, UK, UK: Springer-Verlag, 2002, pp. 403–417.

[44] X. shan Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, “Complete solution classification for the perspective-three-point problem,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 8, pp. 930–943, 2003.

[45] V. Lepetit, F. Moreno-Noguer, and P. Fua, “EPnP: An Accurate O(n) Solution to the PnP Problem,” *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, Jul. 2008.

[46] D. Rolo. Datasets and results. [Accessed September 2013]. [Online]. Available: http://web.ist.utl.pt/diogo.rolo/

[47] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2564–2571.

[48] a. Alahi, R. Ortiz, and P. Vandergheynst, “FREAK: Fast Retina Keypoint,” *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 510–517, Jun. 2012.

[49] D. W. Eggert, A. Lorusso, and R. B. Fisher, “Estimating 3-d rigid body transformations: A comparison of four major algorithms,” *Mach. Vision Appl.*, vol. 9, no. 5-6, pp. 272–290, Mar. 1997.