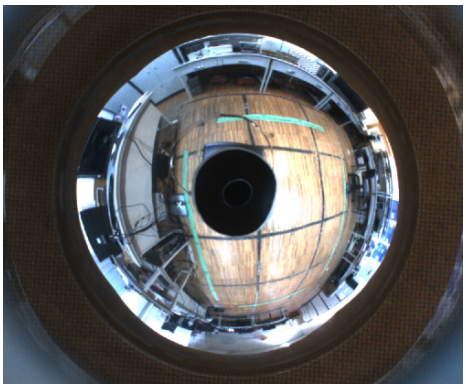
TÉCNICO LISBOA



Improvement of Non-Central Catadioptric Cameras Pose Estimation using 3D Lines

Diogo Emanuel Parreira Maximino

Thesis to obtain the Master of Science Degree in

Electrical Engineering and Computers

Supervisors: Prof. Pedro Daniel dos Santos Miraldo
Prof. Rodrigo Martins de Matos Ventura

Examinatiom Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Pedro Daniel dos Santos Miraldo
Members of the Committe: Prof. José Antonio da Cruz Pinto Gaspar

November 2016

Acknowledgments

I would like to thank my supervisor Pedro Miraldo for all the support. A special thanks to Eng. Andre Mateus who was always available to help.

Abstract

The objective of this work is to keep developing and improve a software to estimate the planar pose of a robot using a visual-based method for a Non-Central Catadioptric System (NCCS). The method works by using the projection model for NCCS with known 3D lines to compute the rigid transformation between the world frame and the camera frame. The software is meant to run as close as possible to real-time, to be robust to noisy observations and to be out-of-the-box. Through this work it will be presented the study made on the theory behind and the process of development. It will be analysed which are the best solutions to solve the problem computationally in respect to computation time and robustness facing errors. It is developed a solution that achieves a fast performance and that is more robust than the method used before.

Keywords

Non-Central Catadioptric Camera Pose

Resumo

O objectivo deste trabalho é continuar a desenvolver e melhorar um sistema de *software* para estimar a posição e orintação de um robot no plano usando um método baseado em visão por computador para camaras catadioptricas não centrais. O metodo funciona usando o modelo de projecção para este tipo de camaras com o auxilio de linas 3D conhecidas para estimar a transformação rigida entre o referencial do mundo e o referencial da camara. O *sofware* terá de ser robusto e *out of the box*. Ao longo do trabalho será apresentado o estudo feito sobre a teoria que está por trás e o processo de desenvolvimento. Serão analizadas as melhores soluções para resolver o problema computacionalmente no que diz respeito à robustez e ao tempo de computação. A solução desenvolvida consegue obter uma performance rapida e mais robusta do que a do método usado anteriormente.

Palavras Chave

Pose Camara Catadioptrica

Contents

List of Figures

List of Tables

Abbreviations

ROS Robotic Operative System

NCCS Non-Central Catadioptric System

CCS Central Catadioptric System

EKF Extended Kalman Filter

FOV Field-of-view

1 Introduction

1.1 Motivation

Absolute pose estimation is a major concern in the field of autonomous systems. The ability of a robot to estimate its pose in the environment is many times a basic feature that allows the robot to do more complex tasks. Either if we need a robot to help at home, play soccer or drive on the road autonomously it must be able to accurately localize itself in the environment. Nowadays there are already several robust ways to do that using a varied kind of sensors. From lasers to RGB-D cameras or even a fusion between different sensors. The truth is that it is never too much to find other ways to solve this problem. In a world highly stochastic, accuracy in pose estimation is not easy to achieve. Furthermore, for different specific situations, there can be techniques that provide more accurate estimations than others. This makes it always worth it to study new ways to solve this problem using different sensor and/or different algorithms. In this work it is presented a technique using a non-central catadioptric camera with quadric mirror to detect the absolute planar location of a mobile robot using known 3D lines in the environment.

1.2 State of the Art

Pose estimation is a problem that has been approached in many different ways, using many different sensors and algorithms. When computing the pose the problem is to find the position and orientation of a robot in a world frame coordinate. As the measurements the robot can perceive from the environment are measured regarding its own referential, the problem can be summed as finding a transformation between the robot frame and the world frame with the help of known landmarks. Due to the improvement of image processing and computer vision methods, cameras proved to be good sensors capable of estimating pose with good accuracy. Many vision-based methods for pose

estimation have thus been proposed.

Cameras have several features, depending on which, different methods can be applied. Perspective cameras have been the most used and those for which more methods have been proposed [3]. The reasons for this are that they are vastly available, simple to use, e.g. there is no need to concern about physical implementation like for Central Catadioptric System (CCS), and their projection model has been densely studied. However the downside of this cameras is that they have a limited Field-of-view (FOV). With the purpose of overcome this limitation, cameras with a higher FOV can and have been used. Examples of this kind of camera are the fisheye camera and the catadioptric camera. In [4] it is studied the benefits of such cameras in pose estimation comparing to perspective cameras for both indoor environments and outdoor environments. It is concluded that, for small confined environments, cameras with higher FOV are definitely preferred. As expected they allow tracking visual landmarks for longer periods, resulting in more measurements which increases the precision of pose estimation and robustness. The only downside presented of using higher FOV cameras is the fact that it decreases angular resolution of the image, lowering the measurement accuracy of a single camera pixel. In this work the camera chosen was the catadioptric camera. It is the one presenting the biggest FOV comparing with the fisheye and perspective cameras. The method proposed also tries to overcome the downside of this kind of camera i.e. the measurement accuracy for just a single pixel. Concerning the vision-based methods both non-minimal and minimal solution methods have been used. As examples of methods used we have [5] and [6] using non-minimal methods using 3D points and 3D lines, respectively. In [7] it is used points and lines for minimal solutions. The method presented here is a non-minimal solution method using 3D lines.

Catadioptric cameras have been used before, but in most of the cases the implementation of the catadioptric system is made in order to use the central projection model i.e. the positions of the perspective camera and the quadric mirror are set in a way that both are aligned allowing an effective single viewpoint, e.g [8], [9]. The problem with this is that in reality the restrictions to use the central projection model are not fulfilled and, by using it, the results will not be precise. The fact that this model does not fit the catadioptric system in a robust way has been potentiating the study of a projection model for non-central catadioptric systems, e.g. a catadioptric systems in which the camera is not aligned with the mirror. In the last few years it has been done by Gonçalves [10] (proposing an iterative solution) and later by Agrawal et. al. [11] (proposing a projection model for Non-Central Catadioptric System (NCCS)). They derived a forward projection equation for a 3D point in space. In this work it will also be used an exact projection model but instead of using a projection model just for one 3D point, it is used a projection model for a 3D straight line. Since lines are one dimensional object the association between the world and camera frames, it is not much more difficult than using just a point. Also for pose estimation purposes it may help increasing the accuracy of the results comparing to the use of a single point, that can induce errors provoked by lower angular resolution, as mentioned before. Other approaches for solving the problem of absolute pose estimation using non-central camera models have been presented by [12], [13] and [14].

1.3 Original Contributions

The original theoretical work was developed by Pedro Miraldo and Andre Mateus [1]. Here it will be briefly presented the improvements in the implementation originated by this work:

- Code optimization in order to perform an almost real-time relationship between the observations and the final output.
- New formulation for the optimization problem that performs better than the method used before.
- Improvement on the image processing phase, resulting in a better detection of the lines.
- Introduction of a filter for the estimations obtained in order to eliminate noise in the output.

1.4 Thesis Outline

This thesis is structured in the following way:

- **Chapter 1** It is the introduction chapter where it is explained what motivates this work, it is presented how similar works have been approached before and it is told what contribution this work brings to this field.
- **Chapter 2** This is the theoretical background chapter. It presents all the theory that was studied in order to understand and be able to develop this work. All the theory presented here was developed before this thesis. It provides an explanation on how catadioptric cameras work, it is presented the projection model for a 3D point in a catadioptric system. Then it is explained how the model applies to lines and how lines can be use to estimate the pose in a camera.
- **Chapter 3** Here is where it is explained the process and the structure of the work developed during this thesis. It is explained the several modules of the implementation and how they were improved through this thesis.
- **Chapter 4** This is the chapter where the important results from this work are listed. It presents qualitative and quantitative analysis based on several experiments described here. The qualitative analysis is regarding an experiment using a robot and a camera to estimate its pose. The quantitative analysis is about robustness tests made with synthetic data-sets.
- **Chapter 5** It is the final chapter where conclusions are taken. Not only about the results listed in the previous chapter but also a comparison with what was already done before. It is also presented a reflection about what else can be done in the future.

2

Theoretical background

2.1 Catadioptric cameras

In this work it is taken the advantage of the properties of catadioptric cameras to estimate the pose of a robot. Catadioptric systems are such that combine reflection and refraction into an optical system. This type of camera is composed by a lens and a curved mirror. This combination provides an enlarged field of view which is the main advantage of this camera. On the other hand, the 3D point projection model for catadioptric cameras is not as easy to come up with as for a common perspective camera.

The projection of a point can be described by several models: from the most simple ideal pinhole camera model to the more complex models which take into account the intrinsic parameters of the camera such as scaling factors, focal length and principal point. Some also take into consideration extrinsic parameters, that provide the alignment between the camera frame and the world frame through rigid transformations.

The ideal pinhole model can be written as follows:

$$\begin{cases} x = f\frac{X}{Z} \\ y = f\frac{Y}{Z} \end{cases} \tag{2.1}$$

where (x,y) denote the coordinates of a 3D point in the image frame and (X,Y,Z) the coordinates of the same point in the camera frame. f represents the focal distance i.e. the distance between the origin of the camera frame and the image frame.

The simplicity of obtaining a model for a perspective camera comes from a fact that it has a single viewpoint i.e. all incoming principal light rays intersect in a single point. This may not be true when we add a mirror, creating a catadioptric system. In fact the configurations that allow this to happen are few. For those, the so-called Central Catadioptric Cameras, the projection of a 3D point is still simple

to obtain analytically, only slightly more laborious. In [8] they are derived the complete class of mirrors that can be used with a single camera to give a single viewpoint and also found the expression for the spatial resolution of a catadioptric sensor in terms of the cameras used to construct it.

These models are severely restrictive in terms of camera positioning or shape of the mirror. Slightly different configurations as having the camera placed off-axis with respect to a quadric mirror will lead to a NCCS. A projection model for this kind of system is more challenging to obtain analytically but it pays off in terms of having a more flexible implementation and better results compared with using a CCS model for approximation. The exact projection model for NCCS is studied in [11] and it will be described in more detail since it is the basis for this work.

2.2 Analytical Projection Model for Non-Central Catadioptric Cameras

Finding a projection of a 3D point in a catadioptric camera is in other words to find the image projection of a 3D point via a mirror. This problem has been densely studied for spherical mirrors and it is popularly known as the Alhazen's problem. The problem can be set as follows: Given two points in a plane of a circle find the point in the circumference where the lines drawn from the two points meet making equal angles with the normal. Other way to formulate the same problem, more related with optics would be: "Given a light source and a spherical mirror find the point on the mirror where the light will be reflected to the eye of an observer." [1] For the projection problem for NCCS using a quadric mirror and no restriction on the camera placement the analytical solution proposed by Agrawal et al. [11] can be obtained by solving an 8^{th} order equation. Let's thus describe it in more detail.

Consider a rotationally symmetric mirror described by:

$$x^2 + y^2 + Az^2 + Bz - C = 0 \tag{2.2}$$

where A,B and C are the parameters that tuned give the shape of the mirror we want to use. Let the mirror axis be aligned with the z axis. Consider a pinhole camera placed at $\mathbf{c} = [c_x, c_y, c_z]^T$ and a give 3D point $\mathbf{p} = [X, Y, Z]^T$. The problem is to find a point $\mathbf{m}$ in the surface of the mirror such that the reflection of the ray joining $\mathbf{c}$ with $\mathbf{m}$ passes through $\mathbf{p}$ i.e. the angle between the line joining $\mathbf{c}$ and $\mathbf{m}$ and the normal of the mirror surface at point $\mathbf{m}$ is the same as the angle between this normal and the line joining $\mathbf{m}$ with $\mathbf{p}$.

[1] 150 A.D., Ptolemy.

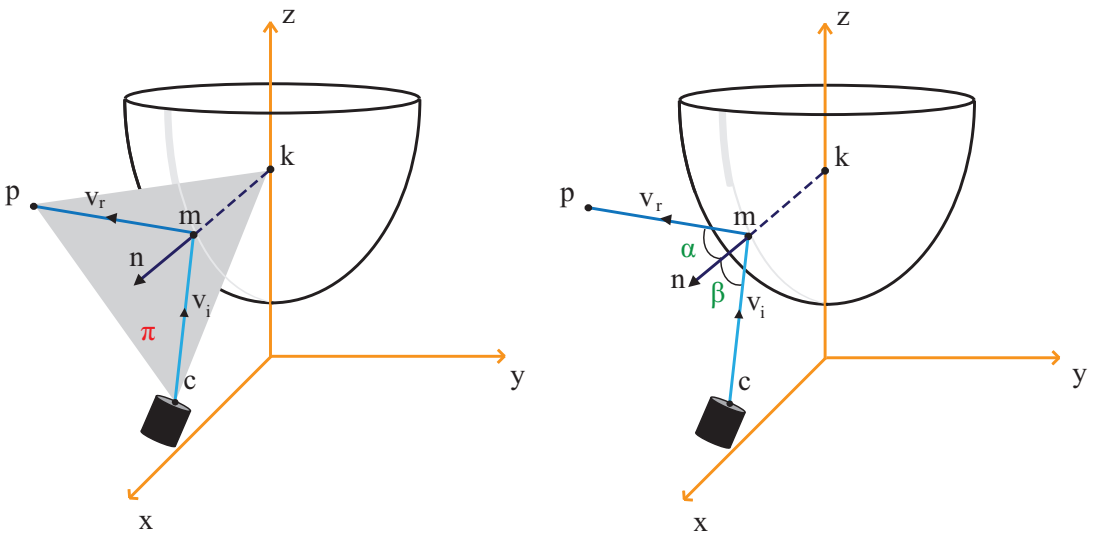


Figure 2.1: Representations of the restriction for the angle, on the right, and plane, on the left. Where $\alpha = \beta$.

Let the incoming ray be denoted by $\mathbf{v_i} = \mathbf{m} - \mathbf{c}$, $\mathbf{n}$ be the normal and $\mathbf{v_r} = \mathbf{p} - \mathbf{m}$ the reflected ray. From the law of reflection we know two things: (1) $\mathbf{v_i}$,$\mathbf{v_r}$ and $\mathbf{n}$ lie on the same plane; (2) the angle between $\mathbf{v_i}$ and $\mathbf{n}$ is the same as the angle between $\mathbf{v_r}$ and $\mathbf{n}$. The two constraints are presented in Fig. 2.1. From them its possible to deduce the following equation.

$$\mathbf{v_r} = \mathbf{v_i} - 2\mathbf{n}\frac{\mathbf{v_i}^T\mathbf{n}}{\mathbf{n}^T\mathbf{n}} \tag{2.3}$$

Besides that, the point $\mathbf{m}$ also has to satisfy the mirror equation (2.2).

To simplify the derivation of the projection equation we can take advantage of the fact that the mirror is rotationally symmetric around the z axis. Thus the entire coordinate system can be rotated in such way that the $\mathbf{c}$ is placed in the y axis. And so we can have a rotation matrix $\mathbf{R}$ such that $\mathbf{Rc} = [0, d_y, d_z]^T$. By setting x to zero the degree of the subsequent equations will be reduced. It is important to notice that by doing this the computation of the point $\mathbf{m}$ will not be affected once the rotation is independent from the point $\mathbf{p}$. So by rotating $\mathbf{p}$ we will get $\mathbf{p_R} = \mathbf{Rp}$ and we will find the mirror intersection $\mathbf{m_R}$ in this coordinate system, however by rotating the point $\mathbf{m_R}$ by $\mathbf{R}^{-1}$ we obtain $\mathbf{m}$ in the original coordinate system.

Another thing that we can do to simplify the derivation of the solution is to use the reflection plane. The reflection plane can be denoted by π and it's the plane where $\mathbf{v_i}$, $\mathbf{v_r}$ and $\mathbf{n}$ lie. Being $\mathbf{m_R} = [x, y, z]^T$ and its normal $\mathbf{n} = [x, y, Az + B/2]^T$ we can define a point $\mathbf{k}$ where the normal intersects the z axis. As the mirror is rotationally symmetric, i.e. the intersection occurs at point $(x, y) = (0, 0)$ we have $\mathbf{k} = [0, 0, z - Az - B/2]^T$. So the equation of π can be obtained using the points $\mathbf{c_R}$,$\mathbf{k}$ and $\mathbf{p_R}$. The equation will be the following:

$$c_1[z]x + c_2[z]y + c_3[z] = 0 \tag{2.4}$$

where $c_1[z], c_2[z], c_3[z]$ are linear functions of z depending on the parameters A, B, C, the coordinates of $\mathbf{c_R}$ and the coordinates of $\mathbf{p_R}$. Solving (2.4) in function of x and then substituting it in the mirror equation it is obtained:

$$(c_1^2[z] + c_2^2[z])y^2 + 2c_2[z]c_3[z]y + c_3^2[z] + c_1^2[z](Az^2 + Bz - C) = 0. \tag{2.5}$$

This equation describes a quadratic equation in y for a given value of z. Intuitively we can see that this equation describes a curve that results from the intersection between the quadric mirror with the plane π. The point $\mathbf{m_R}$ belongs to this curve. There is another constraint it has to satisfy which is the law of reflection and its by applying that, that it is possible to obtain the other equation needed.

The reflected ray has to pass through $\mathbf{p_R}$ which means $\mathbf{v_r}$ has to have the same direction as the line joining $\mathbf{m_R}$ with $\mathbf{p_R}$ i.e.:

$$\mathbf{v_r} \times (\mathbf{p_R} - \mathbf{m_R}) = 0 \tag{2.6}$$

being the incoming ray $\mathbf{v_i} = \mathbf{m_R} - \mathbf{c_R}$, by substituting $\mathbf{n}$ and $\mathbf{v_i}$ in the reflection equation (2.3) it is obtained $\mathbf{v_r}$ which substituted in (2.6) results in 3 dependent equations. These equations have coefficients depending on z and are quadratic equations regarding y. And so using one as second equation needed we have two quadratic equations regarding y with coefficients depending on z.

From there eliminating y^2 it's possible to find the value of y depending on z which replaced in one of those two equations lead us to an 8^{th} order polynomial equation in z. The coefficients of the polynomial also depend on the mirror parameters, on the known location of $\mathbf{c}$ and on the known 3D point.

2.3 3D Line Projection onto Non-Central Catadiopric Cameras

The derivation of the equation to represent the reflection of a 3D straight line on the mirror can be done analogously to the derivation of the equation for a 3D point described previously, i.e. based on the same constraints. The solution will be of the type $\gamma(x, y, z)$ in order to verify that (x, y, z) belongs to the curve, as we can see in Fig. 2.2. In this case instead of a 3D point $\mathbf{p}$ it will be considered a straight line such that $\mathbf{p}(\lambda) = \mathbf{q} + \lambda\mathbf{d}$ for some λ, where $\mathbf{p}(\lambda)$ represents any point on the line for a given λ, $\mathbf{q}$ is a 3D point of the line and $\mathbf{d}$ is the direction of the line. It will still be considered a NCCS with the camera position at $\mathbf{c} = [0, c_y, c_z]$ and a rotationally symmetric quadric mirror that can be described by (2.2). The reflection point for any point in the straight line can be represented by $\mathbf{m}(\lambda) = (x(\lambda), y(\lambda), z(\lambda))$ for some λ.

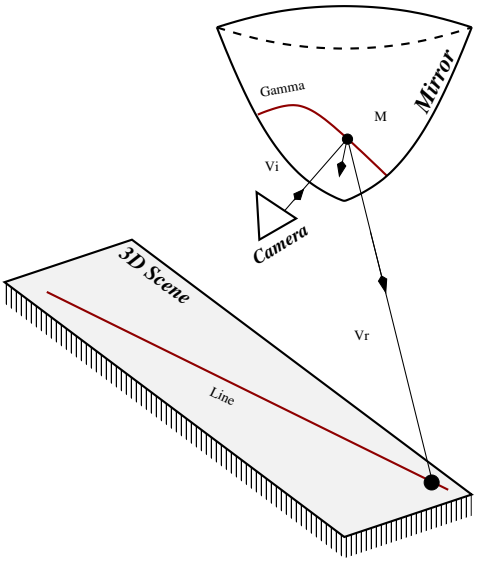


Figure 2.2: Here we can comprehend the correspondence between a line in the world and its reflection curve in the mirror. This image is reprinted from [1]

Considering the planar constraint that says that the center of the camera $\mathbf{c}$, the point in the line $\mathbf{p}(\lambda)$ and the correspondent refection point $\mathbf{m}(\lambda)$ lie on the same plane (which will be denoted by π), it is possible to find the first intermediate equation. To simplify things, instead of using $\mathbf{m}(\lambda)$ to compute the equation for the plane π, it is going to be used the point $\mathbf{k}$ such that represents the intersection of the normal $\mathbf{n}$ at the point $\mathbf{m}(\lambda)$ with the the z axis. As we have $\mathbf{n} = [x \;\; y \;\; Az + B/2]^T$ the intersection with the z axis results in $\mathbf{k} = [0 \;\; 0 \;\; Az + B/2]^T$. Computing the plane equation and solving for x we obtain

$$x = -\frac{c_3^2[y,z]\lambda + c_4^2[y,z]}{c_1^1[z]\lambda + c_2^1[z]} \tag{2.7}$$

where c_i^j is a j^{th} order polynomial equation. Replacing (2.7) in the mirror equation (2.2) we get

$$c_5^4[y,z]\lambda^2 + c_6^4[y,z]\lambda + c_7^4[y,z] = 0 \tag{2.8}$$

Eliminating λ in (2.8) makes it then possible to get an analytical equation for the projection of lines. To do that one can use the angular constraint that says the incoming ray and the reflected ray have to have the same angle with the normal at the reflection point. As we know that the reflected ray $\mathbf{v_r}(\lambda)$ has to pass through $\mathbf{p}(\lambda)$ similarly to the case for a 3D point, for a 3D line we will have

$$\mathbf{v_r}(\lambda) \times (\mathbf{p}(\lambda) - \mathbf{m}(\lambda)) = 0 \tag{2.9}$$

and also

$$\mathbf{v_i}(\lambda) = \mathbf{m}(\lambda) - \mathbf{c}. \tag{2.10}$$

Replacing (2.10) and $\mathbf{n}$ in the derived equation from Snell's law given by

$$\mathbf{v_r}(\lambda) = \mathbf{v_i}(\lambda) - 2\mathbf{n}\frac{\mathbf{v_i}(\lambda)^T\mathbf{n}}{\mathbf{n}^T\mathbf{n}} \tag{2.11}$$

and then substituting this result in (2.9) we get tree dependent equations. To simplify the one independent of x will be used. Solving this for λ we obtain

$$\lambda = -\frac{c_9^3[y,z]}{c_8^3[y,z]}. \tag{2.12}$$

Replacing λ in (2.8), we get the desired equation

$$\gamma(y,z) = c_5^4[y,z](c_9^3[y,z])^2 - c_6^4[y,z]c_8^3[y,z]c_9^3[y,z] + c_7^4[y,z](c_8^3[y,z])^2 = 0 \tag{2.13}$$

where $\gamma(y,z)$ is a 10^{th} order polynomial depending on the mirror parameters, on the known location of $\mathbf{c}$ and on the known 3D straight line.

2.4 Planar Pose using Known Coordinates of 3D lines

In this section the equation (2.13) derived in the previous section will be rewritten to obtain an error function for the estimation of absolute pose. In this case the goal is to find a planar pose as the robot is only moving in a plane constant in the z axis. So to compute the pose we need to estimate the rotation angle θ around the z axis and two translation components t_x and t_y. Being that the robot is considered to be in the camera frame, these components can be found by finding $\mathbf{R}$ and $\mathbf{t}$ that define a rigid transformation between the world frame and the camera frame. Thus the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ will be

$$\mathbf{R} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ c^{te} \end{bmatrix} \tag{2.14}$$

where c^{te} is a known constant that represents the position of the robot in the z axis. Then the unknowns we want to find to solve this problem are θ, t_x and t_y. Consider a set o N known straight lines $\mathbf{l}_i^{(W)}$, for $i = 1, ..., N$, in the world frame. Assume that the lines are not aligned with the camera coordinate system. Consider also a set of M_i pixels $\mathbf{u}_{i,j}$, for $j = 1, ..., M_i$, corresponding to the j^{th} point of the i^{th} line. The reflection points in the mirror $\mathbf{m}_{i,j}$ are obtained using the correspondent pixel $\mathbf{u}_{i,j}$ and the projection matrix of the perspective camera used in the NCCS. That's all the information needed to apply (2.13). However, this information cannot be applied directly because $\mathbf{m}_{i,j}$ points and $\mathbf{l}_i^{(W)}$ are not in the same reference frame. To do that the simplest way is by applying a rigid transformation to the lines. And so we'll have

$$\mathbf{p}(\lambda)^{(C)} = \mathbf{R}\mathbf{p}(\lambda)^{(W)} + \mathbf{t} = \lambda\mathbf{R}\mathbf{d}^{(W)} + \mathbf{R}\mathbf{q}^{(W)} + \mathbf{t} \tag{2.15}$$

where $\mathbf{p}(\lambda)^{(C)}$ represents a line in the camera frame. Now it's possible to apply (2.13). The goal with this will be to find the unknowns $\mathbf{R}$ and $\mathbf{t}$. These parameters define the transformation applied to the lines in the world frame. By replacing (2.15) in (2.13) we get

$$\gamma_r(y, z, cos(\theta), sin(\theta), t_x, t_y) = 0. \tag{2.16}$$

As we know a set of reflected points on the mirror we can set y and z and so (2.16) only depends on the rigid transformation parameters i.e. $\gamma_r(cos(\theta), sin(\theta), t_x, t_y) = 0$ given that they follow the

constraint:

$$g(cos(\theta), sin(\theta)) = cos(\theta)^2 + sin(\theta)^2 = 1. \tag{2.17}$$

The final equation for the reflection curve will then be given by

$$\gamma_r(c(\theta), s(\theta), t_x, t_y) = c_{12}^4[c(\theta), s(\theta), t_x, t_y] + c_{13}^3[c(\theta), s(\theta), t_x, t_y] + c_{14}^2[c(\theta), s(\theta), t_x, t_y] + c_{15}^1[c(\theta), s(\theta), t_x, t_y] + c_{16}^0. \tag{2.18}$$

We can formulate the absolute pose problem for NCCS as an optimization problem by taking the absolute value of the sum of the function $\gamma_r(cos(\theta), sin(\theta), t_x, t_y)$ for all matchings between 3D straight lines and the respective image pixels

$$\begin{aligned} \arg \min_{c(\theta), s(\theta), t_x, t_y} \quad & \sum_{i=1}^{N} \sum_{j=1}^{M} |\gamma_{ij}(\cos(\theta), \sin(\theta), t_x, t_y)| \\ \text{s.t.} \quad & c(cos(\theta), sin(\theta)) \quad = \quad 1 \end{aligned} . \tag{2.19}$$

The pose of the robot can be then found simply by applying the inverse rigid transformation

$$\mathbf{p}^{(W)} = \mathbf{R}^T \mathbf{p}^{(C)} - \mathbf{R}^T \mathbf{t} \tag{2.20}$$

where $\mathbf{p}^{(C)}$ represents a point in the camera frame and $\mathbf{p}^{(W)}$ the same point in the world frame. Being that it is assumed that robot center of mass in the camera frame is $(x, y, z) = (0, 0, 0)$ i.e. the same position as the center of the camera. The position of the robot in the world frame can be given by

$$\mathbf{p}^{(W)} = -\mathbf{R}^T \mathbf{t}. \tag{2.21}$$

3

Approach & Development

3.1 Problem definition and Implementation structure

This work is meant to keep developing and improve a software program that, given an image captured by a catadioptric camera mounted on top of a mobile robot, can compute its absolute pose regarding lines drawn on the floor. This program is a complex system that integrates several modules. For that reason and also because the program is meant to run close to real-time, the implementation is done in the form of a ROS package [15]. Robotic Operative System (ROS) is a set of software libraries and tools that help building robot applications. It works running separated operating system processes, so-called nodes that communicate publishing and subscribing from topics. A node can also send a request to another node and receive a response in return through a mechanism called service. There is a special node, the `roscore`, responsible to introduce the other nodes to each other. Software in ROS is organized in packages. They contain all the programs we need to constitute a useful module.

This program is divided in tree main modules. There is the first module that subscribes an image from the camera and processes it in order to detect the lines and tracking them, publishing then the points that represent each line in another topic. The second module subscribes from this topic the points of each line and uses them to solve the optimization problem described in the previous chapter computing the rigid transformation parameters and after the pose of the robot which is published in another topic. The last module is responsible for filtering the data subscribed from the topic with the pose information and present the result in a graphical way. This modularity allows us to test the different parts of the code separated and also be able to change each module completely independently from the other ones. The aim with this work will be to optimize each of the modules in order to have in the end a robust out of the box software that can be used by anybody only interested to get the

pose of the robot from a catadioptric camera without having any knowledge about the code. In the following sections it will be presented the approaches used to optimize each module and walk-through the process of their development.

3.2 Line detection and tracking

In the problem discussed in this thesis it is proposed to find a rigid transformation for 3D lines from the world frame to the camera frame. To compute that there are three things needed: the center of the camera coordinate in the camera frame (of course ideally it is $(x, y, z) = (0, 0, 0)$ but in the real world cameras do not have the optic center aligned with the camera frame). In other words we need the intrinsic parameters of the perspective camera; points of the 3D lines in the world frame; and points of the same lines reflected on the mirror in the camera frame. The first module is where the points from the reflected lines are computed. This module consists of a node that subscribes the image from the NCCS. From this image there are detected lines and selected pixels from them that correspond to reflected points of the lines in the mirror. The reflected points from the mirror are directly perceived by the perspective camera. By using the projection model for this camera, it is possible to compute the pixel in the image frame that corresponds to the point in the camera frame. On the other hand the detection of the lines requires processing the image.

There was already an implementation for this module, however it could not always guarantee the full detection of the lines. So it was tried a slightly different approach which improved the lines detection. The new implementation can be summed by the following algorithm for each image:

1. Apply color filter (for green color, the color of the lines used) to the image in order to get a binary image as in Fig. 3.2 with only the green parts of the image selected.

2. It is used erode and dilate operations to clean the image. They are use to perform the opening operation first to eliminate small objects and after the closing operation to eliminate small holes.

3. Find straight lines in the image using Hough Transform originating Fig. 3.3.

4. Use Canny edge detection to get the contours of the lines and get the pixels from the blobs that lie inside the contours obtaining Fig. 3.4.

As we are trying to track curve lines it may seem odd to use an algorithm to detect straight lines. The reason it works so well is that when we are processing an image instead of getting a nicely shaped curve of well defined pixels we get a blob that represents the line. In the worst cases, due to noise, it detects several blobs that should represent the same line, but only one is selected as a representation of the line. The main purpose of the straight line detection is to connect those blobs that should represent the same line. Note that if we zoom into a curve line we can fit small straight lines, i.e. in the limit it is a local approximation, and that was the idea behind this.

The tracking of the lines works by choosing with the mouse the lines we want to track in the first image. The lines must be selected in a specific order to associate each with the right line in the

world, that will be supplied in the next module. The program will select the big blob closer to the point selected with the mouse. For the other images the program will find the closest blobs from the blobs selected to represent each line in the previous image. For each line it is then selected randomly 75 pixels from the blob corresponding to that line. Those pixels for each line are then published in a topic. All of this processing phase is implemented in C++ using OpenCV library. The result can be seen in 3.1.

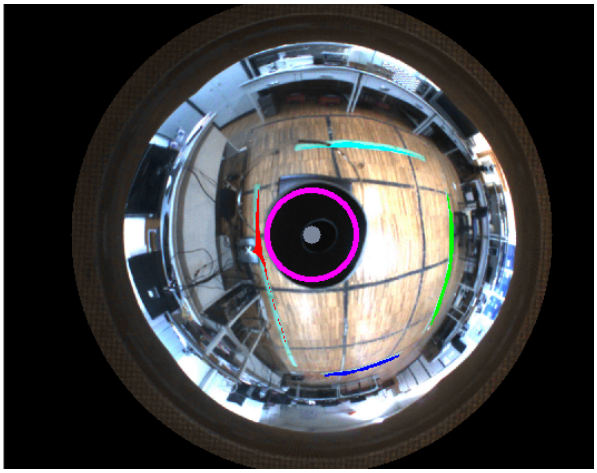


Figure 3.1: Image from the camera with the lines detected by this module.

Figure 3.2: Result of applying color filter to the raw image from the camera.

Figure 3.3: Found straight lines in 3.2 using Hough transform.

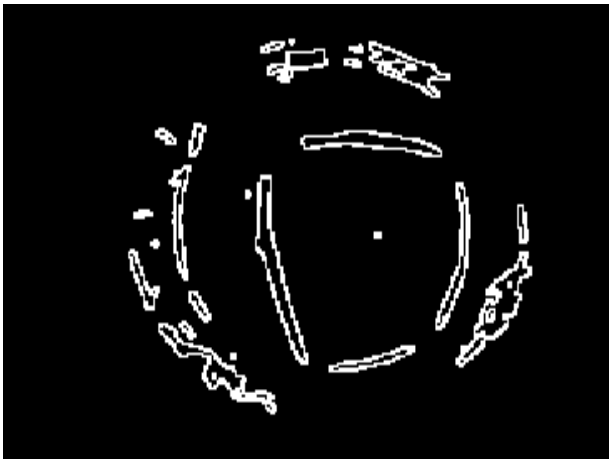


Figure 3.4: Result of applying Canny edge detection in 3.3.

3.3 Optimization Problem

This module is where the pose is computed from the reflected points of the lines detected in the previous module. It also receives as parameters: the coordinates of the camera center; the calibration parameters of the camera; the mirror parameters; and the directions $\mathbf{d}$ and the points $\mathbf{q}$ for each line in the world. The directions and points from the lines in the world together with the reflected points will be used to find λ for each point using (2.12) which will allow us to have the points in the world that correspond to the reflected points. These parameters should be supplied by the user in order to have a general program that works for all kinds of mirrors, all kinds of perspective cameras and where the lines and number of lines can be freely set. The performance for the number of lines set will be discussed in the next chapter. The points from the mirror belonging to each line are subscribed from a topic and, together with the information supplied, we have reunited all the conditions to process to the optimization.

The goal is to minimize a function $\gamma(cos(\theta), sin(\theta), t_x, t_y)$ for each point of a 3D line. Remember that a point belongs to a line if $\gamma(cos(\theta), sin(\theta), t_x, t_y) = 0$. So we want to find the rigid transformation

parameters that set this function as close to zero as possible for each pair of, point in the world, and its reflection point. During the detection of the lines errors may occur due to the noise in the image and that is why it may not be exactly zero. To overcome the noise there are used several points, 75 for each line to be more precise, in order to guarantee results more robust and as accurate as possible. So for all the points, the function we want to minimize is $\sum_{i=1}^{N}\sum_{j=1}^{M}|\gamma_{ij}(cos(\theta), sin(\theta), t_x, t_y)|$ where N is the number of lines and M is the number of points for each line. This function is a sum of several 4^{th} order polynomial with 51 monomials. The approach taken to deal with that is to calculate numerical part of the monomials for each point *a priori* to reduce the computational effort during the optimization. The objective function can be rewritten as:

$$|coef_{1,1}x_1 + ... + coef_{1,51}x_{51}| + ... + |coef_{MN,1}x_1 + ... + coef_{MN,51}x_{51}| \tag{3.1}$$

which can then be represented in a matrix form as follows:

$$\sum_{k=1}^{k=51}\left|\begin{bmatrix} coef_{1,1} & ... & coef_{1,51} \\ \vdots & \ddots & \vdots \\ coef_{MN,1} & ... & coef_{MN,51} \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ x_{51} \end{bmatrix}\right| \tag{3.2}$$

where $coef_{ij,k}$ represents the numerical part of each monomial k, where $k = 1, ..., 51$ and for each point ij where $i = 1, ..., N$ is the number of the line and $j = 1, ..., M$ is the number of the point in the line. The x_k is the literal part of each monomial i.e. a product of $(cos(\theta)^{n_1} sin(\theta)^{n_2} t_x^{n_3} t_y^{n_4})$ where $n_1, n_2, n_3, n_4 \in [0, 4]$. Note that here the module of a vector $\mathbf{v}$, denoted by $|\mathbf{v}|$, represents the module of each of its entrances. This process was previously implemented using `Matlab` to perform the optimization with `fmincon` function using the interior-point algorithm. This implementation although providing good results could not operate close to real-time and thus the need to improve this module.

The initial approach to solve this problem was to implement this module using C++ language. C++ is supposed to achieve a better performance in terms of speed once it is a lower-level language and allows code optimization in a deeper level than `Matlab`. Besides that ROS provides a `roscpp` library that enables C++ programmers to quickly interface with ROS topics, services and parameters. It is also designed to be the high-performance library for ROS. `Matlab` interface for ROS on the other hand requires a slightly more laborious setup.

For those reasons this approach was initially taken. Several C++ libraries for optimization were considered. Nlopt [16] as a free/open-source library for nonlinear optimization with an interface working with several algorithms was initially used to implement this optimization problem. Being our problem constrained and the optimization function with a gradient not trivial to compute analytically the standard algorithm provided by the Nlopt is COBYLA [17] that stands for Constrained optimization by linear approximation. This is a numerical optimization method for constrain problems where the derivative of the objective function is not given.

It works by approximating the constrained problem by a linear programming problem. For each iteration a candidate solution is found solving the approximated linear problem. This solution is then evaluated in the original constrained problem. The information obtained through evaluation is then

used to improve the approximated linear optimization problem in the next iteration. When the solution cannot be improved anymore the step-size is reduced. The algorithm finishes when the step-size is small enough. This algorithm also only searches locally being that the initial value has a lot of influence on the final result.

This implementation was densely tested for several initial values and tolerances but not even the best results can compete with the results given by `Matlab` implementation. Among others the main problem is the oscillation of results from consecutive images processed between two or more local minima which could not be solved even by selecting initial values close from the optimal value found in the previous image processed.

The ideas to deal with that at the time were to use algorithms that provide a global optimization and to use more information about the objective function like the gradient by computing it numerically or analytically. Bearing this in mind it was tried to compute the Jacobian analytically. This is not a trivial task and it has to be done by a computer due to the complexity of the function. It was done using `Matlab` symbolic calculation for the objective function of the constrained problem and so it was obtained as a result the following:

$$\mathbf{J} = \begin{bmatrix} \frac{df}{dc\theta} & \frac{df}{ds\theta} & \frac{df}{dt_x} & \frac{df}{dt_y} \end{bmatrix} \tag{3.3}$$

where f is the objective function and for each component of the jacobian it is obtain an expression like the following:

$$\sum_{ij=1}^{W} \begin{bmatrix} grad_1 \\ \vdots \\ grad_{NM} \end{bmatrix} = \sum_{w=1}^{W} \begin{bmatrix} coef_{1,1} & \dots & coef_{1,51} \\ \vdots & \ddots & \vdots \\ coef_{MN,1} & \dots & coef_{MN,W} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_W \end{bmatrix} \tag{3.4}$$

where

$$\begin{cases} grad_{ij}, \gamma_{ij} > 0 \\ -grad_{ij}, \gamma_{ij} < 0 \end{cases} \tag{3.5}$$

where W is the number of monomials for each component and the other variables represent the same as in (3.2). This gradient as it is a non continuous function which is conditioned by the objective function and for that reason although computing it this way presents accurate results it is very demanding. The number of evaluations of γ becomes huge. For each image, for each iteration of the optimization algorithm it has to evaluate $\sum_{ij=1}^{NM} |\gamma|$ one time and evaluate γ NM times i.e. the total number of points, for each evaluation of the gradient.

Using a rough approximation i.e. ignoring the condition that requires the γ evaluation it was possible to try other algorithms such as Augmented Lagrangian algorithm using ALGLIB library [18] and Interior Point algorithm using IPOPT library [19]. The results, however, were again not satisfactory. `Matlab` implementation although using the same algorithm as the IPOPT library, it does not require a gradient supplied by the user. Instead it computes an approximation for the Hessian of the Lagrangian of the barrier function computed in the interior point algorithm.

Facing bad results the approach taken from here was a little bit different. So instead of trying to find better libraries to implement this problem with better algorithms, it was tried to simplify the problem.

Simplify the problem in other words means to turn it into an unconstrained optimization problem and reduce the number of variables. An unconstrained problem has many advantages comparing to a constrained one. A clear advantage in this case is that for unconstrained problems, it is possible to have algorithms that do not rely on approximations and can work well even without the need to give the information about the objective function, like the gradient. Turn the problem into an unconstrained optimization problem can only be done due to the fact that $cos(\theta)$ and $sin(\theta)$ are functions of θ and so it can be used as a variable instead of having two variables. So we have a new formulation of the problem which is the following:

$$\arg\min_{\theta,t_x,t_y} \quad \sum_{i=1}^{N}\sum_{j=1}^{M} |\gamma_{ij}(\theta,t_x,t_y)| \ . \tag{3.6}$$

For this problem there are better algorithms available in Nlopt library than for this problem. The one providing better results was the controlled random search with local mutation [20], which was expected, once it does global optimization. The algorithm works by generating random points uniformly from a search region and store their values. The stopping rule is then based on the best and worst points, where the best point has the lowest function value and the worst point has the highest function value. The search region is updated until the difference between the function value of the best and the worst point randomly uniformly selected from that region is below a threshold. The minima is then the trial point generated by the selected points from the search area resulting from the optimization process.

The results even better than before are still not comparable with the `Matlab` implementation. In this case even though the minima found for each image is close from the expected result, it is not accurate enough. It can be justified by the fact that this algorithm is good at finding a reduced search area where the minima lies but does not seem to be able to deal with a more refined search locally. The new formulation of the optimization problem was then implemented also in `Matlab` using `fminsearch` algorithm which worked pretty well. This algorithm uses the Nelder-Mead simplex algorithm [21] which does not require any further information about the objective function.

At this point being that `Matlab` implementations provide results so superior to any C++ implementations tested, the solution proposed to have a ROS package working closer to real-time is to divide the module in two nodes. One node that subscribes the points from each image and does all the computations needed before the optimization publishing the resulting coefficients in (3.2). The second node subscribes to the topic with the coefficients message and uses them to perform the optimization. Then it publishes the result i.e. the pose of the robot in another topic. The performance of the C++ computing the coefficients is much better in terms of speed and so it is the language used to program the first node. The optimization node is implemented using a `Matlab` algorithm which is fast for the purpose.

The best qualitative results achieved were for both `Matlab` implentations: the one for the constrained problem (2.19) using `fmincon` with the interior point algorithm and for the unconstrained problem (3.6) using `fminsearch` algorithm. And thus these are the implementations whose results will be explained in more detail in the next chapter.

3.4 Filtering the Data and Graphical presentation of the Pose Estimated

The last module takes care of the presentation of the results. The results from the optimization although very good, always come with some noise associated. For this reason in this thesis it was implemented a solution to smooth the pose computed by the optimization. The solution found was to use a Kalman Filter. These filters do not only take in consideration the measurements i.e. the raw pose computed in the previous module but also the dynamical model of the robot. Although it is a well-known filter it is worth it to explain with more detail how it works for this robot and also some parameter options used.

For the Kalman filter model it is assumed that the true state at time k is evolved from the state at time $k-1$ and this behaviour can be denoted by the following equation:

$$\mathbf{x_k} = \mathbf{A_k x_{k-1}} + \mathbf{B_k u_k} + \mathbf{w_k} \tag{3.7}$$

where $\mathbf{x_k}$ represents the actual state and the state variables, $\mathbf{A_k}$ is the transition matrix i.e. the transition model that is applied to the the previous state $\mathbf{x_{k-1}}$. $\mathbf{B_k}$ is the control matrix or control model applied to the input $\mathbf{u_k}$. $\mathbf{w_k}$ is the process noise i.e. the error responsible for the robot not end up where it is sent by the controller, which is a zero mean multivariate normal distribution $\mathbf{w_k} \sim N(0, \mathbf{Q_k})$ with covariance $\mathbf{Q_k}$.

There is also an equation to predict the measurement $\mathbf{z_k}$ based on the predicted state $\mathbf{x_k}$:

$$\mathbf{z_k} = \mathbf{C_k x_k} + \mathbf{v_k} \tag{3.8}$$

where $\mathbf{C_k}$ is the observation model and $\mathbf{v_k}$ is the observation noise i.e. the error making the measurement being different from the measurement expected, which is a zero mean Gaussian white noise $\mathbf{v_k} \sim N(0, \mathbf{R_k})$ with covariance $\mathbf{R_k}$.

So the way the Kalman filter works can be explained by dividing it in two steps. The first one the prediction step and the second, the update step. During the prediction step it is computed the predicted state given the previous state $\mathbf{x_{k|k-1}}$ and the covariance matrix for the state given the covariance of the previous state $\mathbf{P_{k|k-1}}$, this covariance can be seen as an accuracy measurement for the state. The formulas are the following:

$$\mathbf{x_{k|k-1}} = \mathbf{A_k x_{k-1|k-1}} + \mathbf{B_{k-1} u_{k-1}} \tag{3.9}$$

$$\mathbf{P_{k|k-1}} = \mathbf{A_k P_{k-1|k-1} A_k^T} + \mathbf{Q_k}. \tag{3.10}$$

In the update step it is computed the Kalman gain and the updated values for the state $\mathbf{x}$ and for the covariance $\mathbf{P}$. The optimal Kalman gain is computed as the following:

$$\mathbf{K_k} = \mathbf{P_{k|k-1} C_k^T} (\mathbf{C_k P_{k|k-1} C_k^T} + \mathbf{R_k})^{-1}. \tag{3.11}$$

The term $\mathbf{C_k P_{k|k-1} C_k^T} + \mathbf{R_k}$ represents the covariance of the measurement which is a measure for the accuracy of the measurement we are making. Being that the Kalman gain is proportional to the

inverse of that term, we can verify that this gain is higher for a higher confidence in the measurement i.e. for a lower value of this term. The other updated values are computed in the following ways:

$$\mathbf{x_{k|k}} = \mathbf{x_{k|k-1}} + \mathbf{K_k}(\mathbf{z_k} - \mathbf{C_k}\mathbf{x_{k|k-1}}) \tag{3.12}$$

$$\mathbf{P_{k|k}} = (\mathbf{I} - \mathbf{K_k}\mathbf{C_k})\mathbf{P_{k|k-1}}. \tag{3.13}$$

The equation of the state can be easily understood noting that $\mathbf{C_k}\mathbf{x_{k|k-1}}$ is the prediction of the measurement and $\mathbf{z_k}$ the actual measurement. The updated state will then be the predicted state plus the difference between the measurement and the predicted measurement times the Kalman gain. When the difference tends to zero the updated value will tend to be equal to the state that was predicted. The Kalman gain will give importance to this difference depending on the confidence we have about the measurement. In the covariance equation we have the term $\mathbf{K_k}\mathbf{C_k}$, which represents how important the information about the measurement is, in the way that if it tends to zero the updated covariance of the state will simply be the same as the predicted covariance. However as it grows the effect of the error in the measurement will start to be more noticeable in the covariance of the state. One of the most important assumptions to be able to use those equations is that dynamics are linear in function of the state and control and observations in function of the state i.e. $\mathbf{x_k}$ is linear in function of $\mathbf{x_{k-1}}$ and $\mathbf{u_k}$ and $\mathbf{z_k}$ is linear in function of $\mathbf{z_k}$. As they are linear systems it is possible to apply linear properties of their covariances and get the simple equations described above. Those equations produce the exact result for covariance and its possible to compute the optimal Kalman gain with them.

Figure 3.5: Robot used in this work for the experiments with real data. Image reprinted from [2]

The system used in this work was a Pioneer 3-DX [22], as seen in Fig. 3.5. This robot is a differential-drive mobile robot. studied in this work we have the following equations:

$$\begin{bmatrix} t_{x_k} \\ t_{y_k} \\ \theta_k \end{bmatrix} = \begin{bmatrix} t_{x_{k-1}} \\ t_{y_{k-1}} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \delta t cos(\theta_{k-1}) & 0 \\ \delta t sin(\theta_{k-1}) & 0 \\ 0 & \delta t \end{bmatrix} \begin{bmatrix} v_x \\ w_z \end{bmatrix} + \mathbf{w_k} \tag{3.14}$$

$$\begin{bmatrix} \widehat{t}_{x_k} \\ \widehat{t}_{y_k} \\ \widehat{\theta}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_{x_k} \\ t_{y_k} \\ \theta_k \end{bmatrix} + \mathbf{v_k} \tag{3.15}$$

where (3.14) represents the dynamics of the system and (3.15) the equation for the observations. The state parameters $\mathbf{x_k}$ are t_{x_k}, t_{y_k} and θ_k, the control inputs $\mathbf{u_k}$ are the linear velocity in x, v_{x_k} and the angular velocity in z, w_{z_k} and the measurement $\mathbf{z_k}$ parameters are denoted by $\hat{t}_{x_k}$, $\hat{t}_{y_k}$ and

$\hat{\theta}_k$. As it is noticeable in this case the dynamics of the system are not linear in function of the state as t_{k_x} and t_{k_y} are dependent on trigonometric functions of θ_{k-1}. And for this reason the covariance equations cannot be computed using the equations for the Kalman filter. So it has to be implemented another version of the Kalman filter, the Extended Kalman Filter (EKF) which was developed to deal with systems with non-linearities like this one. EKF is not an optimal filter as its derivation is based on an approximation of $\mathbf{x_k}$ and $\mathbf{z_k}$ and so its accuracy is dependent on how good are the approximations. The equations for the EKF can be formulated as the following:

$$\mathbf{x_k} = f(\mathbf{x_{k-1}}, \mathbf{u_k}) + \mathbf{w_k} \tag{3.16}$$

$$\mathbf{z_k} = h(\mathbf{x_k}) + \mathbf{v_k} \tag{3.17}$$

where $\mathbf{w_k}$ and $\mathbf{v_k}$ represent the same as in the equations above. The functions f and h are used to compute the predicted state $\mathbf{x_k}$ from the previous estimate $\mathbf{x_{k-1}}$, and the predicted measurement $\mathbf{z_k}$ from the predicted estimate $\mathbf{x_k}$, respectively. In the EKF one or both equations are non-linear. In the case studied, only f is a non-linear function. In order to get nice equations for the covariance (3.19) and (3.17) are approximated using a first order Taylor series. With this approximation it is possible to deduce the following equations for the prediction and update steps:

$$\mathbf{x_{k|k-1}} = f(\mathbf{x_{k-1|k-1}}, \mathbf{u_k}) \tag{3.18}$$

$$\mathbf{P_{k|k-1}} = \mathbf{F_{k-1}}\mathbf{P_{k-1|k-1}}\mathbf{F_{k-1}^T} + \mathbf{Q_k}. \tag{3.19}$$

This is the prediction step where it is computed the predicted state estimate $\mathbf{x_{k|k-1}}$ and the prediction covariance estimate $\mathbf{P_{k|k-1}}$. $\mathbf{F_k}$ is the Jacobian of f in order to $\mathbf{x}$ at points $\mathbf{x_k}$ and $\mathbf{u_k}$ and for the system studied we have the following:

$$\mathbf{F_{k-1}} = \frac{\partial f}{\partial \mathbf{x}}\Big|_{\mathbf{x_{k-1|k-1}}, \mathbf{u_k}} = \begin{bmatrix} 1 & 0 & -v_{x_k} sin(\theta_{k-1|k-1})\delta t \\ 0 & 1 & v_{x_k} cos(\theta_{k-1|k-1})\delta t \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.20}$$

The covariance of the process noise and the initial covariance of state were selected as the following:

$$\mathbf{Q_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.07 \end{bmatrix} \quad \mathbf{P_{0|0}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.07 \end{bmatrix}. \tag{3.21}$$

As the expected error on the robot movement for t_x and t_y will be about 1 cm and for the θ 0.07 rad. For the update step we have the following equations:

$$\mathbf{K_k} = \mathbf{P_{k|k-1}}\mathbf{H_k^T}(\mathbf{H_k}\mathbf{P_{k|k-1}}\mathbf{H_k^T} + \mathbf{R_k})^{-1} \tag{3.22}$$

$$\mathbf{x_{k|k}} = \mathbf{x_{k|k-1}} + \mathbf{K_k}(\mathbf{z_k} - h(\mathbf{x_{k|k-1}})) \tag{3.23}$$

$$\mathbf{P_{k|k}} = (\mathbf{I} - \mathbf{K_k}\mathbf{H_k})\mathbf{P_{k|k-1}}. \tag{3.24}$$

Here $\mathbf{H_k}$ denotes the Jacobian of h in order to $\mathbf{x}$ at point $\mathbf{x_k}$. For our system it is the following:

$$\mathbf{H_k} = \frac{\partial h}{\partial \mathbf{x}}\Big|_{\mathbf{x_{k|k-1}}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.25}$$

$\mathbf{K_k}$ here denotes the Kalman gain, not optimal in the EKF case. The covariance of the measurement noise was set as follows:

$$\mathbf{R_k} = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 0.05 \end{bmatrix}. \tag{3.26}$$

As the expected error, resulting from the optimization process in t_x and t_y, is approximately 20 cm and the error in θ is about 0.7 rad.

The filter was implemented in C++ using the OpenCV library and the result of this process is showed in the next chapter. A reflection about this result will be presented in the conclusions. After this process the results are showed as a simulation as in 3.6 that was previously implemented using C++ and OpenGL.

To summarize it, this module subscribes to a topic with the pose computed in the previous module and also to another topic with the commands given to the robot. It uses both informations to filter the results and then presents them as a simulation.

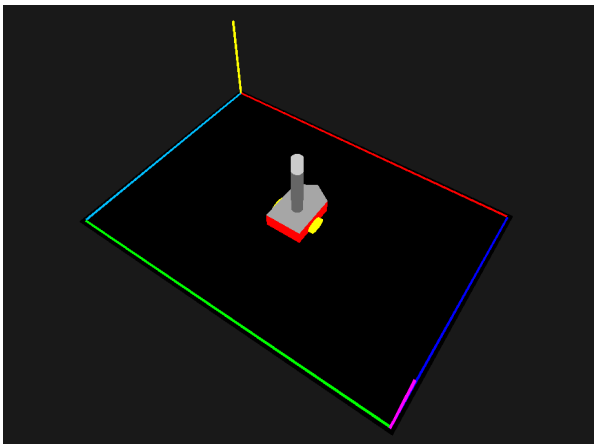


Figure 3.6: Graphical simulation of the robot.

4

Results

4.1 Qualitative results

All the work done in this thesis had the main goal of developing a ROS package capable of applying the vision-based method studied for pose estimation as accurate as possible and as close as possible to real-time processing. The starting point for this was the previous implementation which had the outcome expressed in Figure 4.1 for a trajectory of the robot. The real data experiment was made using a bag file i.e. a record of the topics publications for the raw images coming from a catadioptric camera while the robot performs a trajectory and also the commands sent to the robot. All tests were made using the same bag file in order to be able to analyse and compare the results qualitatively. Unfortunately, as there was no ground truth available, it is not possible to quantify the accuracy for the experiment using this bag file.

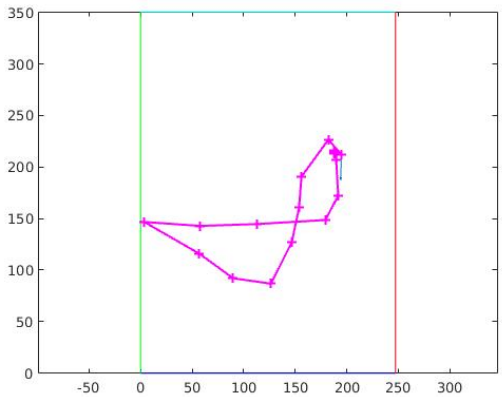


Figure 4.1: Outcome of the pose estimation for the trajectory the robot realised in the bag file.

Several improvements were made in all the modules.

The most critical was definitely the optimization module and the results discussed in this chapter will concern mostly the outcome of the new implementation developed for this module. The solution obtained to improve the performance of the program was to divide the optimization module in two nodes. This improvement by itself results in the outcome express in Figure 4.2. As it can be seen in the figure the new implementation allows to process each image much faster resulting in much more points obtained for the pose estimation. Having more points it is only natural that the accuracy improves even if it is not possible to measure it for this experiment. Note that the optimization algorithm is still the same as before, so the pose computed must be the same. It just takes much less time to run the method and it is possible to get much more measurements.

Other methods were tried in the attempt of producing similar or better results using a C++ implementation. This did not happen and to give a better understanding of what are considered bad results in Fig. 4.3 we can see examples of bad outcomes that were considered not worth it of further analysis. There was however a good result obtained from a new formulation of the problem but still using a `matlab` implementation. As from this experiment it is only possible to give a qualitative analysis, other kind of experiments using synthetic data were made in order to compare the performance of the implementation using the new formulation and the one used before. The quantitative results for the performance of the selected algorithms used to apply the method studied are presented in the next section.

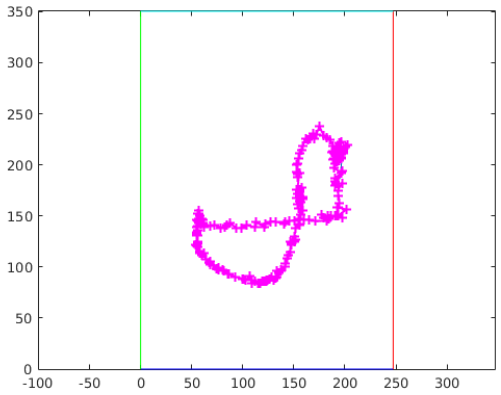
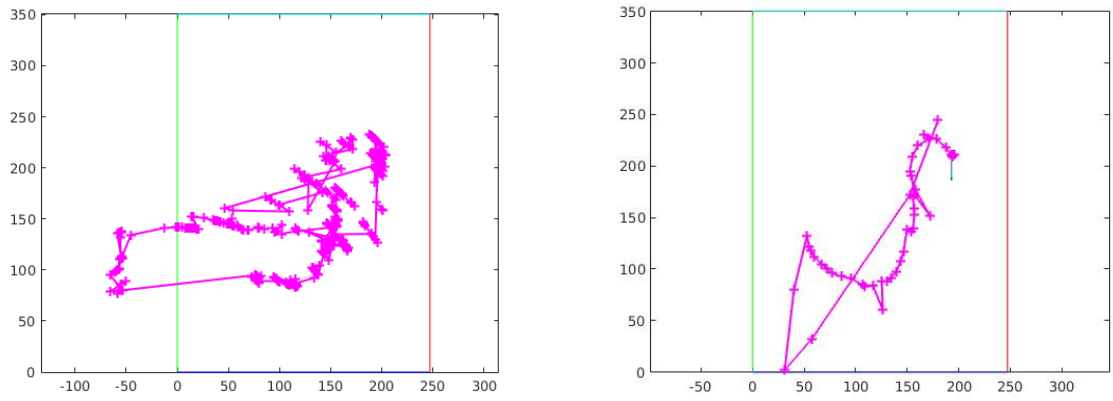


Figure 4.2: Outcome resulting from diving the module in 2 nodes.



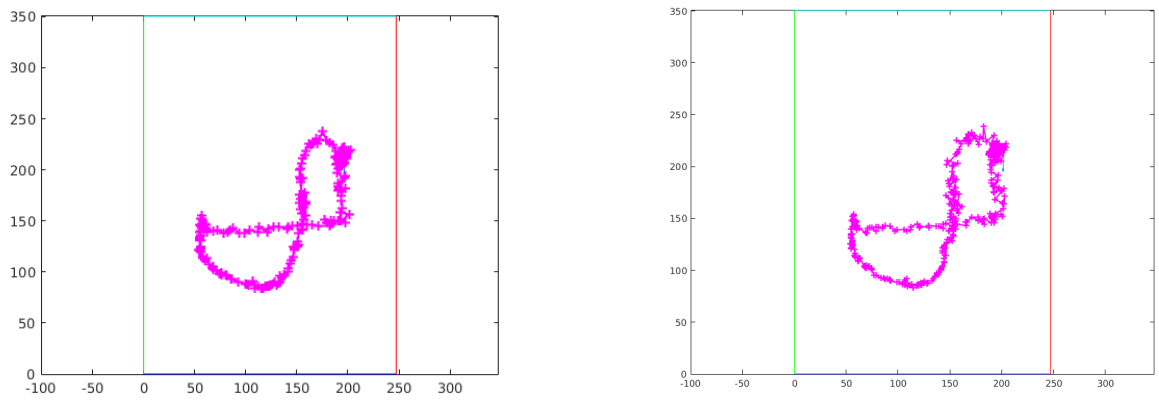
(a) **(b)**

Figure 4.3: Example of bad outcomes produced by some optimization algorithms tried.

4.2 `fmincon` vs `fminsearch`

The algorithms that revealed the best qualitative results, seen in Fig. 4.4, are here compared with detail regarding several parameters in order to do a quantitative analysis of how good they are in respect to the same parameters. The performance was tested regarding:

- How the algorithms perform depending on the number of lines used to estimate the pose.
- How the algorithms deal with errors made in the detection of the pixels in the image.
- How the algorithms deal with errors made in the world coordinates.
- The time they take to compute the pose.



(a) `fmincon` **(b)** `fminsearch`

Figure 4.4: Outcomes of the two best algorithm tried.

For the synthetic data tests it was selected N 3D straight lines randomly generated $\mathbf{p}_i(\lambda)$. Those lines where selected by generating unit length vectors for the points $\mathbf{q}_i$ and directions $\mathbf{d}_i$ and then randomly generate 3D rigid transformations different for each trial, defined by the rotation matrices $\mathbf{R_1}$ and $\mathbf{R_2}$ and the translation vector $\mathbf{t}$ independently generated such that the straight line can be defined by:

$$\mathbf{p}_i^{(W)}(\lambda) = \mathbf{R_1}\mathbf{q}_i^{(W)} + \lambda\mathbf{R_2}\mathbf{d}_i^{(W)} + \mathbf{t}. \tag{4.1}$$

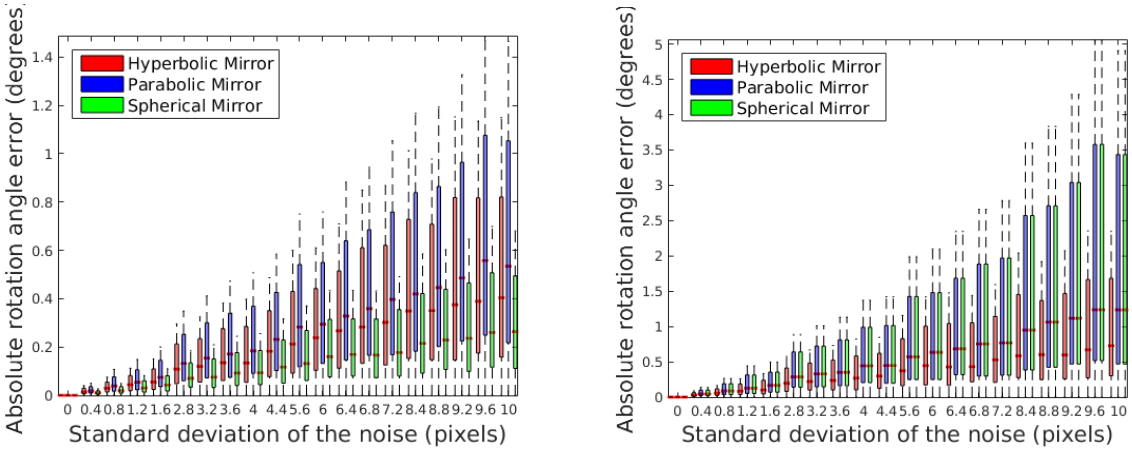
These lines are then transformed by a ground truth rotation and translation parameters. From each line that results from this transformation, there are selected M points to be projected to the mirror using the method (2.18) and optimizing to find the transformation parameters. The transformation parameters obtained are then compared with the ground truth transformation parameters. This comparison will be expressed as the norm of the angles difference and the norm of translation difference, representing them the angular error and translation error, respectively. For these experiments there are used 3 kinds of mirrors: hyperbolic, spheric and parabolic with the parameters in Table 4.1.

Table 4.1: Mirror parameters used for the synthetic data experiments and respective camera optical center used in the same run.

	A	B	C	$\mathbf{c}(x, y, z)$
Hyperbolic	-1.2	3.4	-33.2	(0,25,25)
Parabolic	0	20.4	53.2	(0, 30, 20)
Spheric	1	0	900	(0, -15, 55)

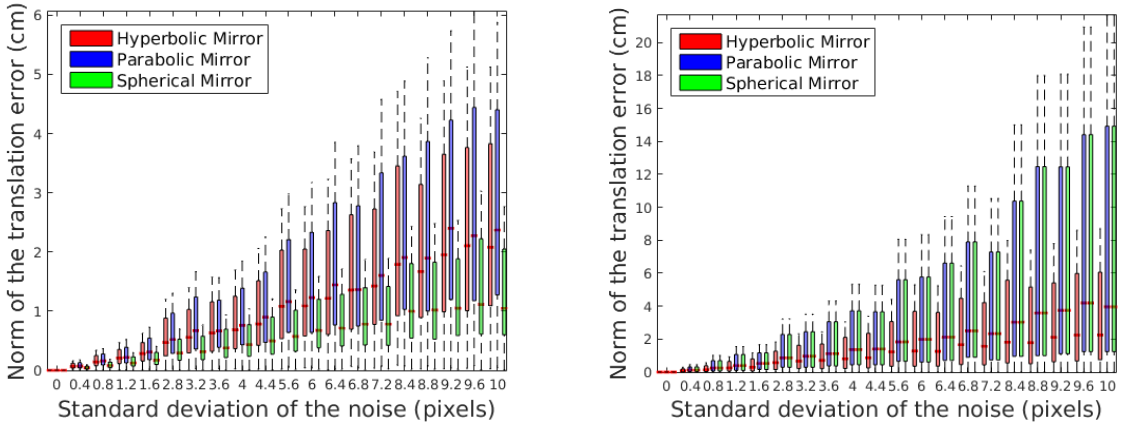
4.2.1 Robustness facing errors in the pixel detection

In order to test the effect of noise in the pixels it was projected the reflected points $\mathbf{m}_{i,j}$ into the image plane and added a normal distribution to the data with zero mean and standard deviation ranging from 0 to 10, with a constant number of 5 lines. The pixels are then back-projected into the mirror originating the reflected points $\mathbf{m}_{i,j}$ to be used in the method. Applying the method for this points there are obtained the estimated transformation parameters, that are going to be compared with ground truth transformation parameters, to estimate the error. The results obtained are expressed in the Figures 4.5 and 4.6 for the 3 kinds of mirror and for both optimization algorithms we want to compare. They are computed 1000 trials for each trial, being that a trial is the execution of the method for the data-set generated.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.5: Angle error originated by errors in the pixels detected.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.6: Translation error originated by errors in the pixels detected.

In Tab. 4.2 it is shown the average of the difference of the errors between the two optimization algorithms. This average was computed as follows:

$$\mu = avg(\boldsymbol{\mu}_{e_{\texttt{fmincon}}} - \boldsymbol{\mu}_{e_{\texttt{fminsearch}}}) \tag{4.2}$$

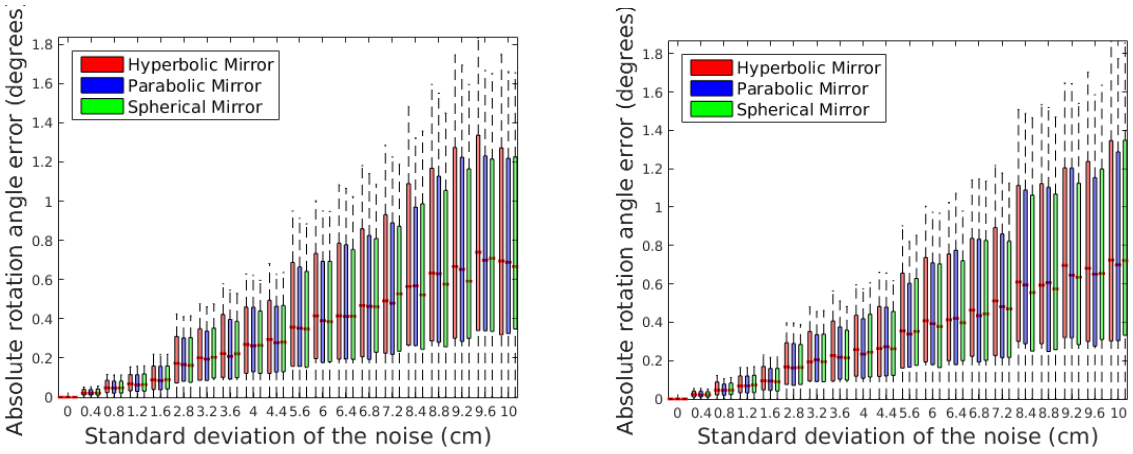
where $\boldsymbol{\mu}_{e_{\texttt{fminsearch}}}$ and $\boldsymbol{\mu}_{e_{\texttt{fminsearch}}}$ are the vectors with the mean value for the trials, using the same the same standard deviation, for each algorithm. μ is the average after making the difference between the two vectors. This average is computed for translation and rotation parameters separately and for each type of mirror.

Table 4.2: Average difference of the errors between the 2 algorithms regarding the noise in the pixels.

	Mean of Angle difference (degrees)	Mean of Translation difference (cm)
Hyperbolic	1.1378	3.7606
Parabolic	2.3134	7.5650
Spherical	2.4407	8.1968

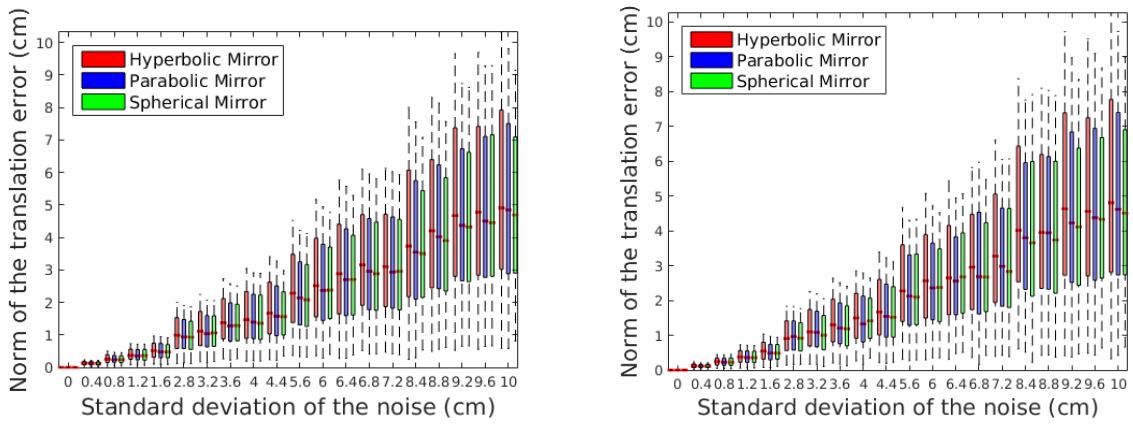
4.2.2 Robustness facing errors in the points in the world

To test the effect of the noise on the coordinates of the points in the world frame it is added a normal distribution with zero mean to the straight lines 4.1 and a standard deviation ranging from 0 to 10, with a constant number of 5 lines. Using the method there are computed 1000 trials . The results originated can be found in Fig. 4.7 and Fig. 4.8 for 3 kinds of mirror and both optimization algorithms.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.7: Angle error originated by an error in the points in the world frame.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.8: Translation error originated by an error in the points in the world frame.

In Tab. 4.3 we can find the average of the difference of the errors between the two optimization

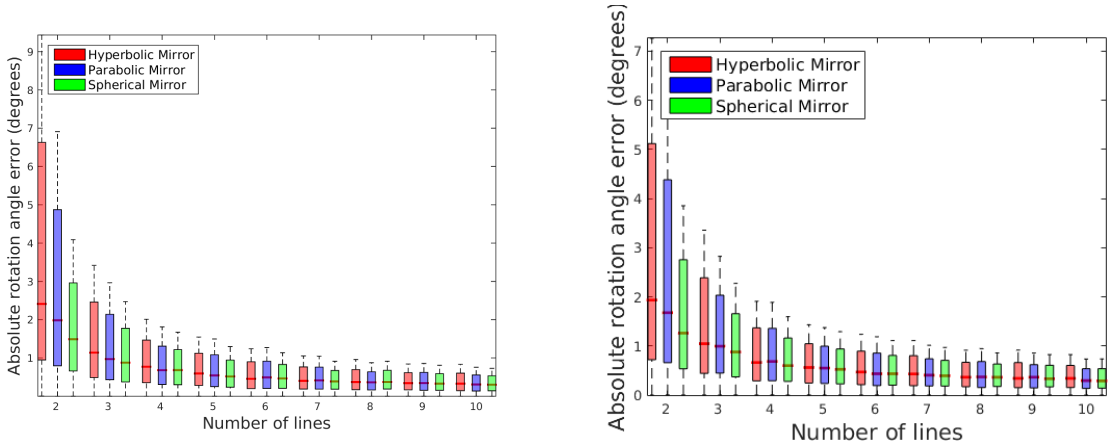
algorithms computed as in (4.2) for the 3 kinds of mirror tested and for the angle and translation.

Table 4.3: Average difference of the errors between the 2 algorithms regarding the noise in the world coordinates.

	Mean of Angle difference (degrees)	Mean of Translation difference (cm)
Hyperbolic	0.5016	-0.0474
Parabolic	0.5577	-0.0367
Spherical	0.5045	-0.0432

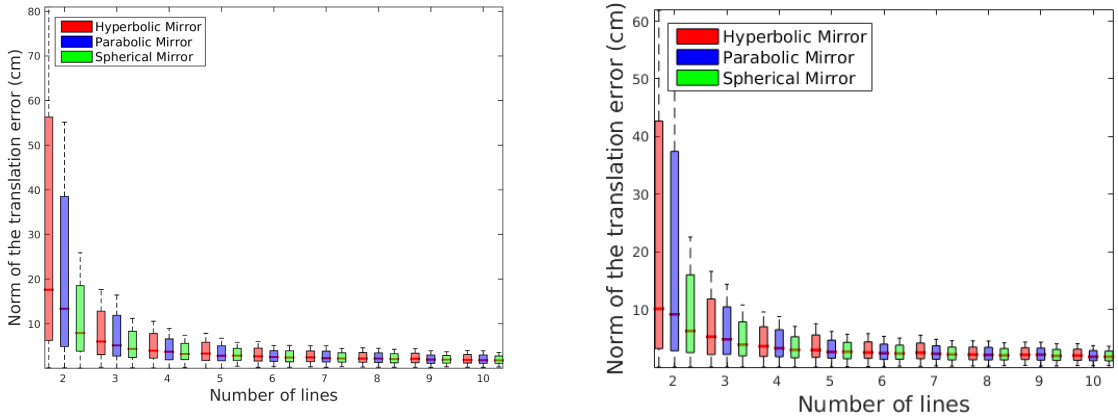
4.2.3 Performance regarding the number of lines

To test the performance as a function of the number of lines used, the synthetic data-set is generated as in the previous section, but the standard deviation is fixed at 5 cm and what varies between trials is the number of lines. The results can be seen in Figures 4.9 and 4.10.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.9: Angle error regarding the number of lines used.



(a) `fminsearch` **(b)** `fmincon`

Figure 4.10: Translation error regarding the number of lines used.

In Tab. 4.4 we can find the average of the difference of the errors between the two optimization

algorithms, for the 3 kinds of mirror tested, and for the angle and translation, computed as in (4.2) but this time $\mu_{e_{\texttt{fminsearch}}}$ is the mean value of the trials using the same number of lines.

Table 4.4: Average difference of the errors between the 2 algorithms regarding the number of lines used.

	Mean of Angle difference (degrees)	Mean of Translation difference (cm)
Hyperbolic	0.2326	-2.2380
Parabolic	0.4383	-0.5201
Spherical	0.0557	-0.1348

4.2.4 Computational time comparison

One of the biggest goals of this work was to make the implementation as close as possible to real-time. To be able to analyse how that worked out, it was computed the computational times for both nodes in the optimization module in Tab. 4.5. It is the result from the times of the optimization algorithm for both optimization algorithms tested during the tests done for robustness. For each test it was computed the average time value for each algorithm. In Tab. 4.6 it is compared the computational time between computing the coefficients in (3.2) using `Matlab` and using C++.

Table 4.5: Comparison between the computational time for both optimization algorithms tested.

	Time `fmincon` (s)	Time `fminsearch` (s)	Difference (s)
Pixel error test	0.0728	0.0589	0.0139
World coordinate error test	0.0679	0.0638	0.0041
N lines test	0.0780	0.0784	-3.8925e-04

Table 4.6: Comparison between the computational time between using C++ and `Matlab` to compute the coefficients in 3.2

	Time C++	Time `Matlab`
Coefficients computation	0.0195	1.2790

4.3 EKF filter applied to pose estimation

Using the real data from the bag file available it was possible to test the implementation of the EKF filter. It was used with the parameters chosen as presented in Section 3.4. The result is shown in the Figures 4.11 and 4.12 where we can see a comparison between the raw pose estimation (in pink) and the filtered pose estimation (in blue).

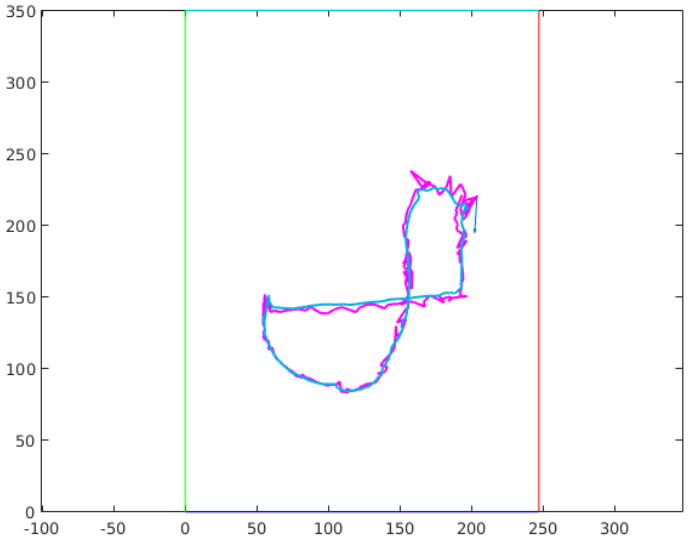


Figure 4.11: Result of the applying the kalman filter to the robot pose. In pink it is represented the trajectory that results from the optimization and in blue the trajectory after applying the filter.

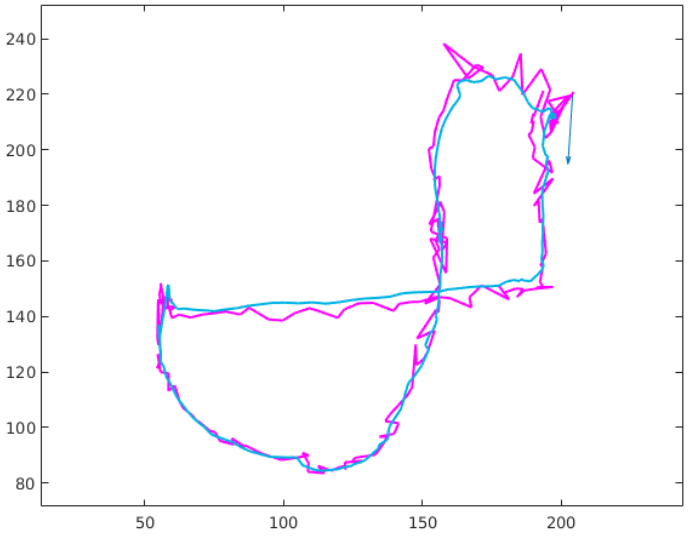


Figure 4.12: Zoom of 4.11.

4.4 Analysis of the results

For the robustness tests it was tested the influence of the noise in the pixels and in the world coordinates in the output error. Regardless the algorithm it can be seen in the graphics that the trend is the output error to increase almost linearly with the errors in the method inputs. Regarding the number of lines it can be seen that for a bigger number of lines the same error in the input originates less error in the output as it was expected. Once we have more observations without increasing the noise it makes sense that the measurement is more accurate.

Specifically for each test we can see that for the world coordinate errors the `fmincon` algorithm has better performance than `fminsearch` in the estimation of the translation but slightly worse for the rotation parameter estimation, on average, regardless the kind of mirror. As the `fminsearch` optimizes directly the angle unlike the `fmincon` there is less error associated. `fmincon` optimizes the variables $cos(\theta)$ and $sin(\theta)$ and thus, in order to get an estimate of θ it has to use the function $atan$. By doing that the error may be increased due to a computation of an approximated value by this function.

For the pixel errors it can be seen that, on average, `fminsearch` algorithm produces less error in both estimations, rotation and translation, for all mirrors.

For the computation time, for the coefficients computation it is clear that the C++ implementation is much faster than the `Matlab` implementation and it was the best result taken from this work.

Besides that, comparing the algorithms for optimization, `fminsearch` in general and, on average, it is also slightly faster than the other.

If we compare the times between the implementation used before and the new one we see that before we had $avg(\boldsymbol{t}_{\texttt{fmincon}}) + t_{\texttt{Matlab}} = 0.0729 + 1.279 = 1.3519$ s and for the new one we have $avg(\boldsymbol{t}_{\texttt{fminsearch}}) + t_{\texttt{C++}} = 0.067 + 0.0195 = 0.0865$ s. t in this case represents time. This means that now we have a pose estimation each 0.0865 seconds while before it was each 1.3519 seconds. Considering the the robot going at its top speed using in the experiment (0.194 m/s) we have approximately a measurement each 1 cm while before it was every 25 cm.

5

Conclusions and Future Work

5.0.1 Achievements

In this thesis it was developed three modules, that together, implement the visual-based method for absolute planar pose estimation described in Chapter 2. Considering the implementation done before, there are some improvements in each module. The improvements are a result of a recursive process of research, experiment and analysis of results. For the first module we had already a good implementation before. It could recognize most of the times at least 3 lines or parts of lines. The introduction of a slightly different algorithm to detect small straight lines within the blobs, and between blobs close to each other, improved the robustness of detecting the full lines. For the bag file tested the improved implementation can detect 4 lines correctly at all times and most of the times the full lines. As proved by the synthetic results, for the test where it is varied the number of lines, detecting more lines correctly increases the accuracy of the pose estimation. Thus it is concluded that this improvement has a direct impact on the final result.

The optimization module was the most challenging to work on. The function it is tried to optimized is highly non-linear, in the sense that trying to approximate it by a linear function would not produce good results. Also its gradient is a non continuous function. Optimizing this function can be highly demanding to compute depending on the algorithm tried to use. From all the algorithms tested that did not perform very well there are a few conclusions to take. The first one is that algorithms that require analytic computation of the gradient take a lot of computational effort. The algorithms that try to approximate the constrained problem with linear programming problems, without using gradient or Hessian, did not produce acceptable results. And the algorithms tested based on search area could not search well locally. The best results were found using `Matlab` algorithms for the constrained and unconstrained optimization problems. These algorithms proved to be very efficient and robust. The two best implementations were tested for noise robustness and computational time. On average

the best results were found for the algorithm that solves the unconstrained optimization problem. It only performed worse for the noise in the world coordinates, presenting a slightly higher error for the translation norm. Overall this lead us to conclude that in general `fmincon` would be the best to use. The computational time was also better on average. The use of two nodes in this module ended up being the best result of this thesis, as the overall problem can be solved much faster.

The last module also suffered an improvement regarding the smoothness of the final result for the estimated trajectory of the robot. The implementation of the EKF, although can not guarantee the optimal result, as it is based on an approximation, presents an output that looks a lot like the real trajectory of the robot.

In sum it up, there is now a software that can estimate the pose approximately each 0.08 seconds, which is an excellent improvement. Besides that, the software was improved in terms of robustness regarding errors mainly in the pixels detection and the results obtained are now processed in order to have a smooth outcome for the pose.

5.0.2 Future Work

The new implementation for the image processing phase was only tested for one experiment, it should in the future be tested for different environments in order to tune the parameters as well as possible in order to behave in a robust way in any general situation.

Although it was possible to test robustness with synthetic data, it was not possible to measure the difference between the pose estimated and the ground truth, since there was not one available. This should be the next step, to test the accuracy of the pose estimation regarding the true position and orientation of the robot. For this reason it was also not possible to test quantitatively how well does the EKF behaves for the pose estimation.

Using a ground truth could be also possible to test more computational expensive methods for filtering, like Monte Carlo, and compare performances. The resulting software performs as desired. The setup still involves using `Matlab` being that part less user friendly. However using `fminsearch` there is a chance to create a `Matlab` function callable from C and maybe in the future it can be possible to make the optimization module using just one node and the same results presented in this thesis.

Bibliography

[1] A. Mateus, P. Miraldo, and P. U. Lima, “Non-central catadioptric cameras pose estimation using 3d lines,” *CoRR*, vol. abs/1607.02290, 2016. [Online]. Available: http://arxiv.org/abs/1607.02290

[2] “Pioneer 3 - dx,” http://cbslab.kocaeli.edu.tr/RRC-Lab/pioneer.php, accessed: 2016-10-15.

[3] H. Araujo, R. L. Carceroni, and C. M. Brown, “A fully projective formulation to improve the accuracy of lowe’s pose-estimation algorithm,” *Comput. Vis. Image Underst.*, vol. 70, no. 2, pp. 227–238, May 1998. [Online]. Available: http://dx.doi.org/10.1006/cviu.1997.0632

[4] Z. Zhang, H. Rebecq, C. Forster, and D. Scaramuzza, “Benefit of large field-of-view cameras for visual odometry.”

[5] H. Araújo, R. L. Carceroni, and C. M. Brown, “A fully projective formulation to improve the accuracy of lowe’s pose-estimation algorithm,” *Computer Vision and Image Understanding*, vol. 70, no. 2, pp. 227–238, 1998.

[6] A. Ansar and K. Daniilidis, “Linear pose estimation from points or lines,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 578–589, 2003.

[7] B. M. Haralick, C.-N. Lee, K. Ottenberg, and M. Nölle, “Review and analysis of solutions of the three point perspective pose estimation problem,” *International journal of computer vision*, vol. 13, no. 3, pp. 331–356, 1994.

[8] S. Baker and S. K. Nayar, “A theory of single-viewpoint catadioptric image formation,” *International Journal of Computer Vision*, vol. 35, no. 2, pp. 175–196, 1999.

[9] C. Geyer and K. Daniilidis, “A unifying theory for central panoramic systems and practical implications,” in *European conference on computer vision*. Springer, 2000, pp. 445–461.

[10] N. Gonçalves, “On the reflection point where light reflects to a known destination on quadratic surfaces,” *Optics letters*, vol. 35, no. 2, pp. 100–102, 2010.

[11] A. Agrawal, Y. Taguchi, and S. Ramalingam, “Beyond alhazen’s problem: Analytical projection model for non-central catadioptric cameras with quadric mirrors,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 2993–3000.

[12] C.-S. Chen and W.-Y. Chang, “Pose estimation for generalized imaging device via solving non-perspective n point problem,” in *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, vol. 3. IEEE, 2002, pp. 2931–2937.

[13] G. Schweighofer and A. Pinz, “Globally optimal o (n) solution to the pnp problem for general camera models.” in *BMVC*, 2008, pp. 1–10.

[14] D. Nistér and H. Stewénius, “A minimal solution to the generalised 3-point pose problem,” *Journal of Mathematical Imaging and Vision*, vol. 27, no. 1, pp. 67–79, 2007.

[15] 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.

[16] S. G. Johnson, “The nlopt nonlinear-optimization package.” [Online]. Available: http://ab-initio.mit.edu/nlopt

[17] M. J. Powell, “A direct search optimization method that models the objective and constraint functions by linear interpolation,” in *Advances in optimization and numerical analysis*. Springer, 1994, pp. 51–67.

[18] S. Bochkanov, “Alglib,” 2010, http://www.alglib.net.

[19] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10107-004-0559-y

[20] P. Kaelo and M. Ali, “Some variants of the controlled random search algorithm for global optimization,” *Journal of optimization theory and applications*, vol. 130, no. 2, pp. 253–264, 2006.

[21] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, “Convergence properties of the nelder–mead simplex method in low dimensions,” *SIAM Journal on optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[22] I. MobileRobots, “Pioneer 3-dx.” [Online]. Available: http://www.mobilerobots.com/MobileRobots.aspx