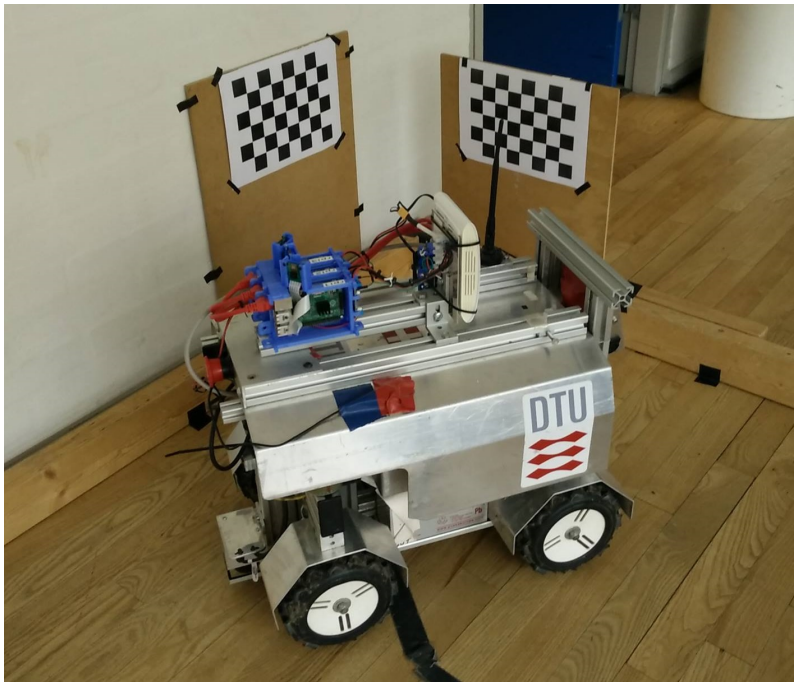
TECHNICAL UNIVERSITY OF DENMARK
DEPARTMENT OF ELECTRICAL ENGINEERING

NAVIGATION FOR MOBILE ROBOTS USING INERTIAL MEASUREMENTS AND SPARSE LANDMARKS

Martin Davidsen Kirkegaard s123674
Javier Fuentes Alonso s161815
Omar Gallardo Puertas s170052
Jonas Kohlschmidt s170045

Wednesday 28th June, 2017



DTU Electrical Engineering
Department of Electrical Engineering

CONTENTS

1 Introduction

1.1 Motivation and Goals

In the following years the use of mobile robots for various tasks will increase sharply [1]. Autonomous ground vehicle robots are used in factory environments to optimize the transportation of goods, in warehouses [2] or in agricultural environments automating field work to reduce physical work and increase harvest outcome [3]. Amazon announced to use autonomous drones to deliver goods to its customers to increase shipment time drastically [4].
In all of these cases the robot has to deal with changing environments, which makes localization using optical sensors more difficult as fixed structures won't be detectable all the time. Due to this and the fact that the robot has to travel long distances in the mentioned examples, the vehicle must rely on it's odometry for longer runs.
To increase the range of odometry travel, the exact starting position of the robot needs to be known. Even a small offset in the beginning, especially in the angular position of the robot, will result in great position errors after relatively short movement.

Therefore a high precision initial position estimation is crucial for lengthy odometry travels without deviating from the intended path. Improvement of the position estimation was the task of this project, with the goal of getting an accurate navigation using the odometry of the robot and sparse landmarks only.

1.2 Robot in use

The vehicle used to carry out this project was the DTU field robot. The four-wheel robot with Ackermann steering is equipped with a gyroscope to measure angles and a laser sensor to scan and measure the distance to surrounding objects. Additionally, three cameras are mounted on top of the vehicle to take pictures in the robot's field of view.

1.3 Approaches and Strategy

Three approaches to increase odometry travel have been investigated and partly implemented in this project.

The first approach used the robot's cameras and computer vision to identify different added landmarks at the starting point. The distance and angular relationship of the identified landmarks and the precise knowledge of their position was then used to determine the exact angular position of the robot.
For the second approach the robot's laser sensor was investigated, as the laser scan is used for the localization. The option of averaging multiple scans to gain more precision was evaluated, offsets in the laser scan were found and a method for compensating these will be introduced.
The third approach analyzed the robot's odometry measurements experimentally to find offsets and compensate them to improve odometry runs apart from increasing initial position estimation.

2 VISUAL POSE ESTIMATION

In this chapter we are going to explain how we have approached the problem from a computer vision perspective. This problem is commonly refer to by the computer vision community as the 'Pose estimation problem' and lately it has attracted the interest of the research community due to its importance in augmented reality, 3D reconstruction, robotics and other novel technologies.

Due to this growing interest a lot of different approaches have appeared in the specialized journals. For example, in [5] the author uses sift descriptors and a previously computed 3D model to calculate accurately the 3D pose of the object. In a similar approach from what we will do, we have [6], in which a simple iterative method is presented. Finally, in [7] a completely different way of tackle the problem is presented, using a genetic algorithm.

In our case we have decided to base our solution in landmarks as a way of modeling the 3D space considering that the initial position of the robot will not always be the same. Thus, the landmarks allow to easily export this model to any initial position changing only the position of two points in the code. For the 2D-3D matching different algorithms have been studied, as we will see in section 2.3.

Our solution starts by determining the system in which we are going to implement our algorithm. Next, and as a first stage of our vision pipeline, the determination of 2D points in the image for different landmarks has been addressed. The next step in our pipeline is the matching of 3D and 2D points, which will result in the camera rotation and translation matrices. Finally, after applying dome frame transformations based on the previous results we were able to give the localization of the robot and its orientation.

2.1 SYSTEM CHARACTERIZATION

In order to get a accurate final result, it is critical to describe the system that we are working with as detailed as possible. In this chapter two coordinate systems will be presented. one is the world coordinate system. The aim is to find the robots position in world coordinates. The other coordinate system is the robots camera coordinate system. The camera coordinate system is used for our vision based localization.

2.1.1 CAMERA CALIBRATION

Due to limited information available for the camera sensors on the robot, it was necessary to perform a camera calibration.

There are three similar camera sensors on the robot. one looking straight ahead, and one in each side looking at an angle from the robot, see figure 2.1). We chose to focus on the one looking straight ahead. The sensor being used is an Omnivision OV5647. Its pixel size is 1.4 microns and it has a maximum resolution of 2592x1944 pixels. Under normal circumstances, with the data sheet of the sensor and known characteristics of the lens on top of the sensor, it would be possible to calculate the camera matrix, A, describing the focal length of the camera, the offset of the optical axis on to the bare image sensor, and the affine offset in x and y directions. In our case, the images provided by the ucamserver in the robot has a resolution of 1024x768. This resolution comes from binning and cropping the original image. We have no knowledge of how this crop was placed and therefore cannot calculate the center of the optical axis in the image. Also we have no information on the lens and therefore we cannot compute the focal length. In terms of affine offset, this is the only parameter we can say something about, as we assume that it can be neglected. With this assumption the camera matrix boils down to the following:

$$A = \begin{bmatrix} f & 0 & dx \\ 0 & f & dy \\ 0 & 0 & 1 \end{bmatrix}$$

The parameters we need to estimate via calibration is thus, the focal length and the offset in x and y directions of the optical axis (with the image origin in the upper left corners, and the x-axis pointing left, and the y axis pointing down). Due to complexity we choose not to take lens distortion into account.

For Calibrating the camera we used the computer vision package in Matlab. It comes with a ready made calibration tool that uses images of a chessboard in different positions, taken with the camera that is being calibrated. Examples of these images taken with the robot camera can be seen in figure 2.2). The images were loaded into the camera calibration tool and the calibration were performed. The tool works by detecting the corners of the checkers in the chessboard. A user types in the physical size of the checkers, and from this it can compute the 3D position of the chessboard while estimating the camera parameters as well. A screenshot of the Matlab Gui can be seen in figure 2.3).

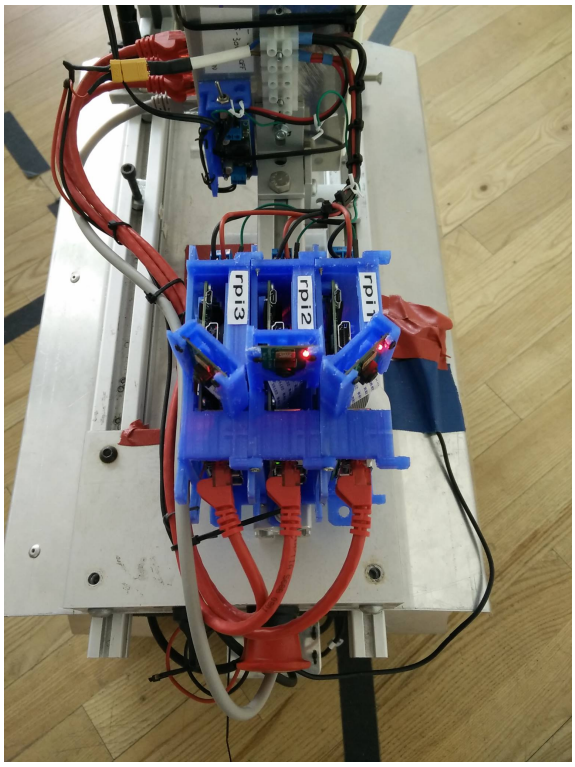


Figure 2.1: Cameras places on top op the robot. Middle one pointing straight and the two others pointing at an angle to each side

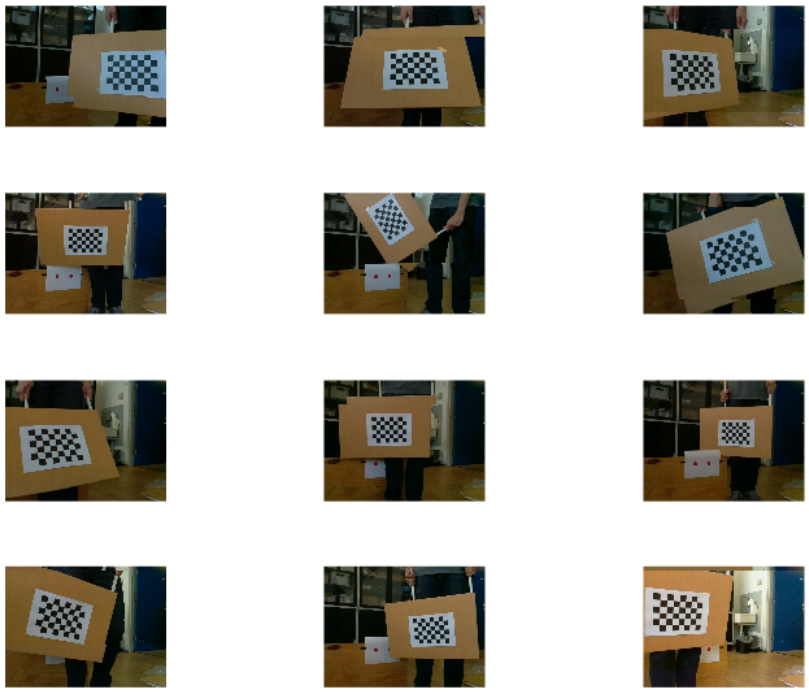


Figure 2.2: Example images used for camera calibration via matlab

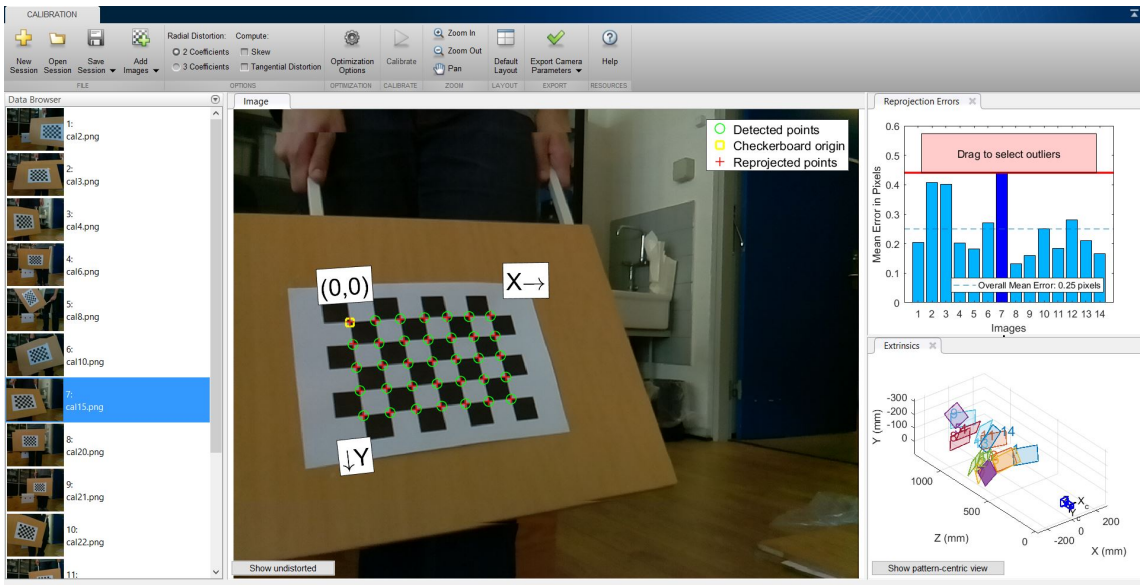


Figure 2.3: Angular range of the laser sensor

The estimated focal length and offset of the optical axes were found and put into the camera matrix, A, and can be seen below

$$A = \begin{bmatrix} f & 0 & dx \\ 0 & f & dy \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1000 & 0 & 511.13 \\ 0 & 1000 & 371.68 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

2.1.2 Camera to robot frame transformation

As mentioned in the previous chapter the camera sits on top of the robot. It is important to know the position of the robot relative to the camera coordinate system. With this knowledge it will be possible to transform the robot position to world coordinates from the estimated location of the camera by the vision algorithms.

At first we investigated if the optical axis of the camera were aligned with the orientation of the robot frame. A laser pointer were placed on each side of the robot pointing in the same direction as the robot and the robot were then placed perpendicular to a wall. Two images were taken, at different distances to the wall, see figure 2.4. From each image it is possible to calculate the ratio of the distances from the laser points to the optical axis of the image sensor. If the camera is angled compared to the robot frame, the ratios will change. An illustration of the problem can be seen in figure 2.5 and the data from the calculations can be seen in table 2.1.

(a) Highlighted Laser points on wall taken at 125 cm distance

(b) Highlighted Laser points on wall taken at 175 cm distance

Figure 2.4: Pictures taken, to calculate alignment between camera and robot frame

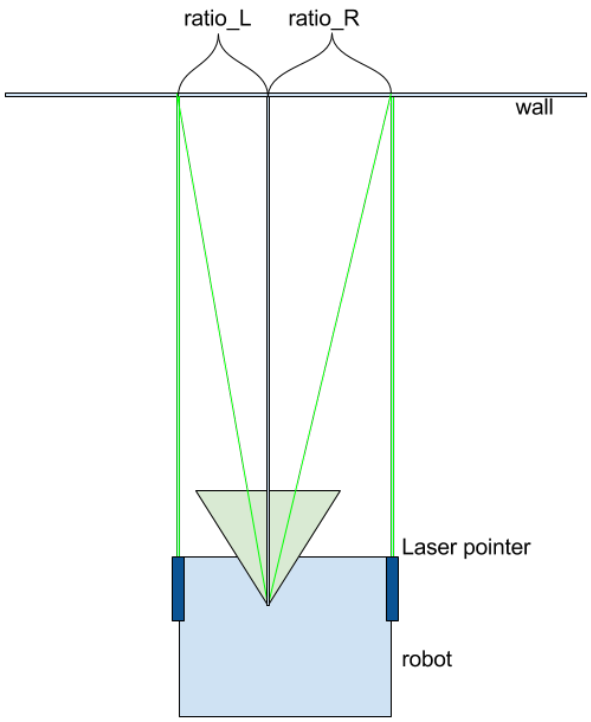


Figure 2.5: Setup for testing the alignment of the optical axis with the robot frame

Table 2.1: Optical axis offset measurement data

Image	Dist to wall	Left dist to optical axis	Right dist to optical axis	Dist between laser points	Ratio_L	Ratio_R
1	125 cm	51.13 pixels	97.87 pixels	149 pixels	34.32 %	65.68 %
2	175 cm	35.13 pixels	68.87 pixels	104 pixels	33.78 %	66.22 %

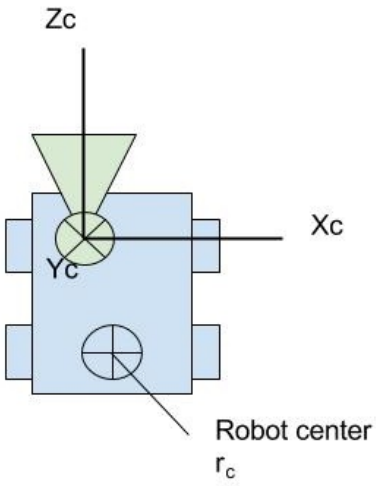


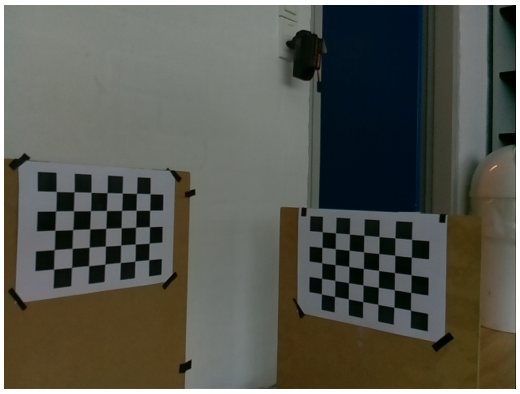
Figure 2.6: Top down view of the robot, with the camera coordinate system and robot center marked

It can be seen that the left and right ratio is almost the same for both images, thus we can conclude that the camera must be almost perfectly aligned with the robot frame. The slight difference of 0.54% is so small, that it can be caused by measurement errors or other uncertainties and it was, therefore, not taken into consideration.
These measurements only looked at the horizontal alignment. We decided to neglect the vertical offset, as simple tests (undocumented) showed us that it would be very small and thus wouldn't have any effect on the final results, as we consider the robot from a top-down perspective, as depicted in figure 2.6. If there had been a horizontal misalignment between camera and robot, this 'error' would get translated directly to angular estimate of the robot in the world if it were not taken into account.

The only thing left to define is the translation of the robot center in camera coordinates. We only consider x and z offset (in camera coordinates, cf. figure 2.6) as this is all we need to compute the robots position in the world. We will denote this translation, in homogeneous coordinates r_c and it was found to be:

$$r_c = \begin{bmatrix} -0.035 \\ 0 \\ -0.315 \\ 1 \end{bmatrix}$$

(a) Circles grid landmark as seen from the camera of the robot



(b) Chess board landmark as seen from the camera of the robot

Figure 2.7: Selected landmarks to test

2.1.3 Landmarks selection

Due to the lack of constraints in terms of selection of landmarks, different options have been considered. The main factors when choosing this landmarks have been accuracy, precision and easy implementation.

Circles grid landmark As our first approach to this problem, we created a landmark composed by 4 red circles (see figure 2.7a). The color was chosen for easy segmentation based on the environment where the tests were conducted.

This landmark was good enough for the initial prototyping phase given its easy point extraction and limited number of correspondences to address. However, it is not the appropriate when looking for accuracy and precision, as it is difficult to determine the actual center of the circle, especially when not facing the landmarks directly. Therefore another solution were needed

Chess board landmark As we have mentioned before, our initial choice for the landmark lacked the accuracy and precision required for this project, which will lead to the decision to use landmarks more appropriate for accurate localization. The choice was to use landmarks similar to the ones seen in section 2.1.1, where the camera calibration problem was tackled. This choice is motivated by its proven efficacy in determining precisely and accurately the intersections of the squares. The increased number of points

will add to the robustness of the system. We decided to use two 8x6 chessboards giving a total of 70 points to be used for position estimation

2.2 IMAGE PROCESSING: 2D POINT EXTRACTION

The first step in our position estimation pipeline is to determine the exact location of the 2D points from the landmarks.

2.2.1 PROBLEM DESCRIPTION

Given an image where two determined landmarks are completely visible determine the n key points p_c, where $p_c = \begin{pmatrix} x_c & y_c \end{pmatrix}$ being x_c and y_c the coordinates in the camera frame of said landmark points.

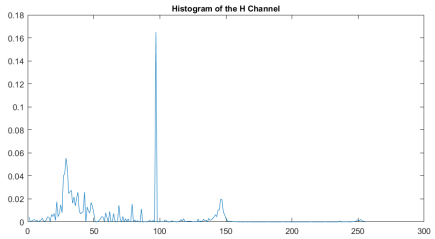
2.2.2 PROPOSED SOLUTIONS

Considering that we have worked with two completely different landmarks the solutions vary greatly.

CIRCLES GRID LANDMARK In this landmark the key point is the center of each circle. In order to calculate that, we will pass through the following steps:

- *Step 1: HSV conversion* By transforming into HSV color space we are going to be able to threshold the image more accurately. In this format the a certain color has a constant value in the Hue channel, and the changes in the luminosity of the scene is shown as changes in the other channels.

- *Step 2: Thresholding* The first step for thresholding is to examine the histogram of the image. It is possible to see the histogram of the hue channel in figure 2.8a, where the values close to 256 represent the red of the landmarks.Thus, we have decided to accept only the pixels which hue channel is in the top 5%, which gives us the results seeing in the binary image of figure 2.8b.



(a) H channel histogram from figure 2.7a

(b) Thresholded binary image with $th = 0.95$

Figure 2.8: Circles grid landmark solution

- *Step 3: Blob selection and centroid determination*In an iterative manner, we are going to detect the biggest blob in the image and then we are going to calculate the centroid of said blob by computing the center of mass of this object. After each iteration we are going to subtract the biggest computed blob so that in the next loop it detect a different blob.

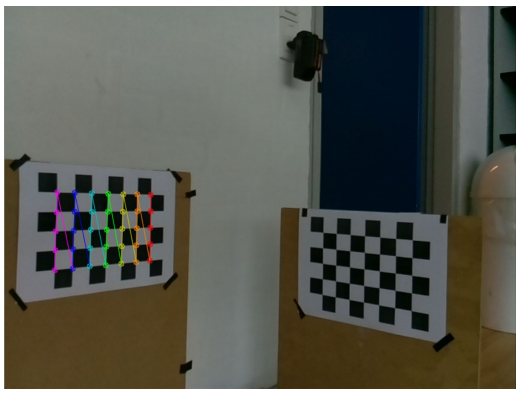
- *Step 4: Point sorting* Once we have all the key points of the landmarks, it is necessary to sort them so that they are in the same order as the physically measured 3D points.

CHESS BOARD LANDMARK The key points in this case are the intersections at the corner of the squares.The process has been the following

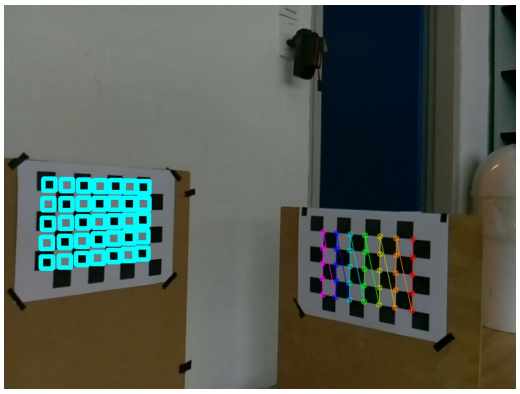
- *Step 1: Chessboard detection* Firstly, we try to detect any of the chessboards. As there are two different ones in the image the algorithm chose one based on how clear it is. This points can be seen in figure 2.9a.

- *Step 2: First chessboard occlusion* As we want to detect also the other chessboard using the same algorithm as before, we are going to draw a random shape over the previously detected corners. This will hide the first chessboard.

- *Step 3: Second chessboard detection* In the same way that we detected the first chessboard we are going to scan the image out from step 2 for a chessboard. The results are shown in figure 2.9b.



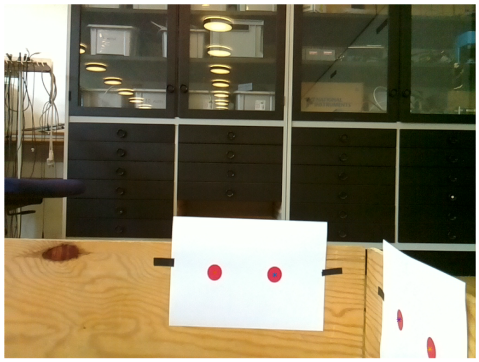
(a) Points detected in the first chessboard



(b) Points detected in the second chessboard after occlusion of the first one

Figure 2.9: Detection of key points using the chessboard landmark

(a) Points detected facing the landmarks



(b) Points detected parallel to one of the landmarks

Figure 2.10: Detection of key points using the circles landmark

- *Step 4: Point sorting* Finally, and as we did for the other landmark, we are going to sort the points so that it matches with the 3D points.

2.2.3 Implementation and results

The implementation of this algorithms, and as well as all the other parts of the pipeline has been done in two steps: a prototyping phase in Matlab and an implementation phase in C++ using OpeCV libraries.

The results using the first landmark can be seeing in figure 2.10. As we can note, this points are not always perfectly aligned with the center of the circle, and in situations like the one presented in 2.10b, the algorithm struggles to give an accurate position of the center.

On the other hand, the results with the chessboard are remarkable, as seen in figure

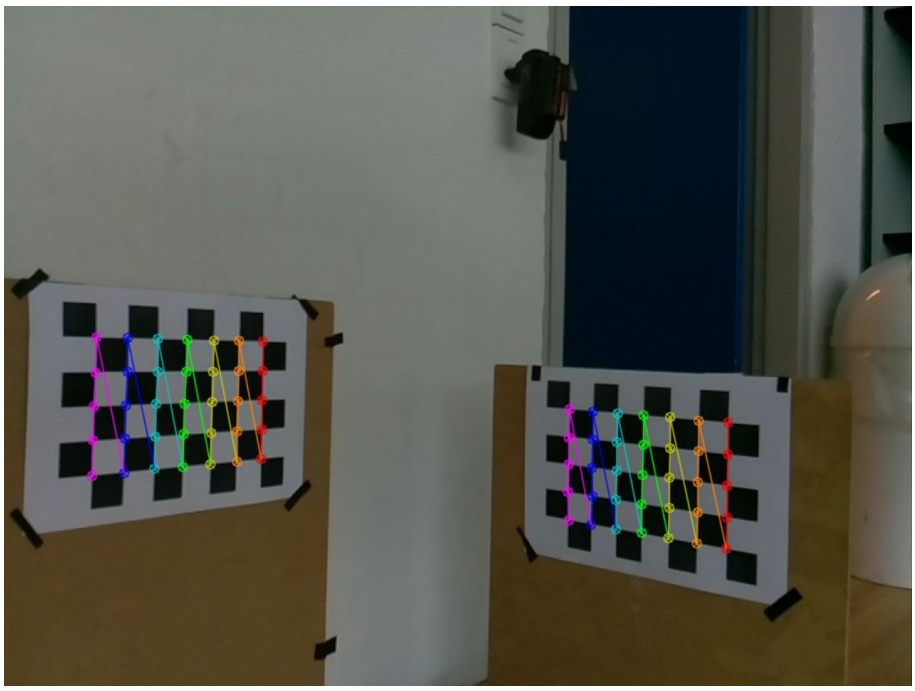


Figure 2.11: Key points extracted from chessboard landmarks

2.11. Due to this results, we have decided to complete the final results of the project using this kind of landmarks.

2.3 PnP:3D-2D matching

Once all the correspondences are known, it is possible to calculate the rotation and translation of the camera by solving the known as 'Perspective-n-Point' problem or 'PnP'.

2.3.1 Problem description

Given a set of n 3D points in world coordinates and their correspondences in the 2D image plane as well as the intrinsic camera parameters matrix, calculate the rotation matrix R and the translation matrix t so that:

$$p_c = A \begin{bmatrix} R & t \end{bmatrix} p_w \tag{2.2}$$

being $p_w = \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix}^T$ in homogeneous coordinates, $p_c = \begin{bmatrix} x_c & y_c & 1 \end{bmatrix}^T$ and the A matrix is the internal camera parameters matrix as seen in 2.2.

2.3.2 PROPOSED SOLUTIONS

As we have mentioned previously, the pose estimation problem has attracted the interest of the research community for a few years, and with it the PnP problem has being a central part of it.

Within the PnP problem, there some specific cases like the P3P problem like [8], but given that we are not constrained in the number of points we are going to explore further only the PnP case.

The solutions of this problem can be divided into two different approaches: the iterative method and the non-iterative methods. The iterative methods, as the one presented in [9], gives really accurate results. However, you still need to determine an initial pose, which adds complexity to the problem. Moreover, when trying to implement this kind of solutions we have noted that if the initial pose estimation is not close enough to the actual position the algorithm returns completely wrong values.

On the other hand, the non-iterative methods have been increasing in popularity due to its low complexity and computational speed, which makes it perfect for solving the visual odometry problem, between others. Among this solutions the most accepted one is the one presented in [10], which is the one that we are going to use for our problem.

This algorithm, the so called 'EPnP algorithm', has became the state of the art due to its advantages to their competitors in terms of computational time, offering a solution with $O(n)$ in comparison with other state of the art solutions which are $O(n^5)$ or even $O(n^8)$. The EPnP algorithm achieves this results basing the algorithm in the assumption that all the reference points can be expressed as the weighted sum of 4 control points. This problem formulation reduces the complexity to the estimation of this control points in the camera frame, and not all the individual reference points.

Furthermore, as we have mentioned before, this algorithm is optimal for visual odometry, which is the natural next step to keep optimizing the solution of the whole problem. Thus, this step of the pipeline can be easily reused in the upcoming projects.

2.3.3 IMPLEMENTATION AND RESULTS

The implementation of this algorithm was based on the code provided by the authors of the previously cited paper, which ensures an optimal performance both in Matlab and C++. The original code can be found via this website: `http://cvlab.epfl.ch/EPnP/index.php`

The results out of this algorithm are the rotation matrix and the translation vector used to transform between world coordinates and camera coordinates i.e. the matrix R and vector t in equation 2.2.

In order to validate this results, we are going to reproject this points again into the image, showing a really small reprojection error, which ensures the trustworthiness of the algorithm.

2.4 Frame transformations: camera to robot position

Once the transformation between world and camera is known and the transformation between camera and robot frame is also established, it is possible to forecast the localization of the robot.

2.4.1 Problem description

Given a rotation matrix R and a translation matrix t that maps the world points into camera coordinates as expressed in 2.2 determine the position x_w and y_w as well as the angle θ_w of the robot where the camera is mounted.

2.4.2 Proposed solutions

In this case, the proposed solution relies on linear algebra to compute all the transformations.

Firstly, we are going to calculate the position of the robot in world coordinates. Given the matrix $\begin{bmatrix} R & t \end{bmatrix}_{w2c}$ to transform between world to camera coordinates it is trivial to calculate the transformation between camera to world coordinates. It is defined by the inverse, as we see in 2.3.

$$\begin{bmatrix} R & t \end{bmatrix}_{w2c}^{-1} = \begin{bmatrix} R & t \end{bmatrix}_{c2w} \tag{2.3}$$

Now, given this transformation and the translation vector r_c between camera and robot origo presented in 2.1.2we calculate the point in world coordinates as follows:

$$r_w = A \begin{bmatrix} R & t \end{bmatrix}_{c2w} r_c \tag{2.4}$$

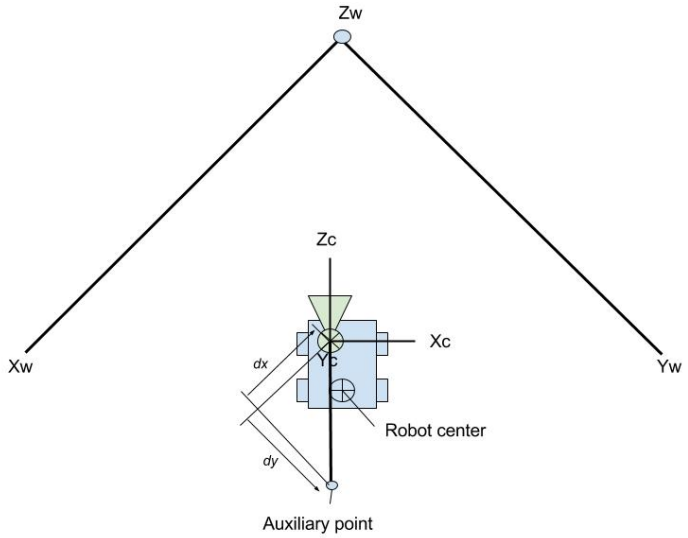


Figure 2.12: Robot angle calculation sketch

Once the position is established it is time to consider the angle of the robot θ. Our approach, based on the fact that camera and robot are aligned, can be seen in figure 2.12. We start by expressing in world coordinates the camera origin and the auxiliary point, which is a random point in the z axis of the camera. After this calculation, it is possible to determine the components of the derivative of the line between this two points as seen in 2.5 and 2.7.

$$d_x = xAux_w - xCam_w \tag{2.5}$$

$$d_y = yAux_w - yCam_w \tag{2.6}$$

With this values, and using the well known function $atan2$ we can calculate the angle as follows:

$$\theta_r = atan2(d_y, d_x) - 180; \tag{2.7}$$

2.4.3 Implementation and results

The implementation on the robor and the results from the vision algorithm will be described in the next chapter. and final testing can be found in the chapter on validation of

the results

2.5 PLUGIN TO SMRCL

This section will describe briefly how the vision algorithms were implemented on the robot.

2.5.1 GENERATING THE PLUGIN

In order for everything to run in the field robot, it was necessary to make a C++ plugin, that would be executable from a smr-cl script while the robot was running.

We decided to base the plugin on the already existing ballfinder plugin, used in DTU course advanced autonomous robots. This provided us with a framework to call the function from smr-cl, read images into the plugin, perform our needed calculations, and return values back to the robot.

The whole function is build as a class that inherits all needed functionality. Whenever a message is send from smr-cl to the vision server with the tag ”VisLoc” the plugin will run. The image is received in the plugin and is transferred to a temporary buffer. From here all the computations as described in the previous chapters are run. When all this is done the plugin will return 4 values to the robot in the variables $vis1 - $vis4. $vis1 - $vis2 contains the estimated x and y position of the robot in the world, $vis3 contains the estimated angle of the robot in world, and $vis4 is a flag indicating if the calculations were performed successfully.

2.5.2 TESTING THE PLUGIN

To test the plugin the robot were placed in a 90° corner with a guidemark on each side, as illustrated in figure 2.13. The position of the robot were roughly estimated to be: x = 0.94 , y = 0.965 , $\theta = -135°$.

An image were then taken and loaded into the plugin. The image can be seen in figure 2.14. and the commands and return values from the Linux terminal can be seen in 2.15. It can be seen that the plugin estimates to robot to be at position x = 0.98 , y = 0.916 , $\theta = -136.9°$. These initial results were satisfactory, as the purpose were to see if the localizing plugin returned values more or less similar to the measured ones. Due to uncertenties when measuring the robors position in the world, it makes no sense to compare the

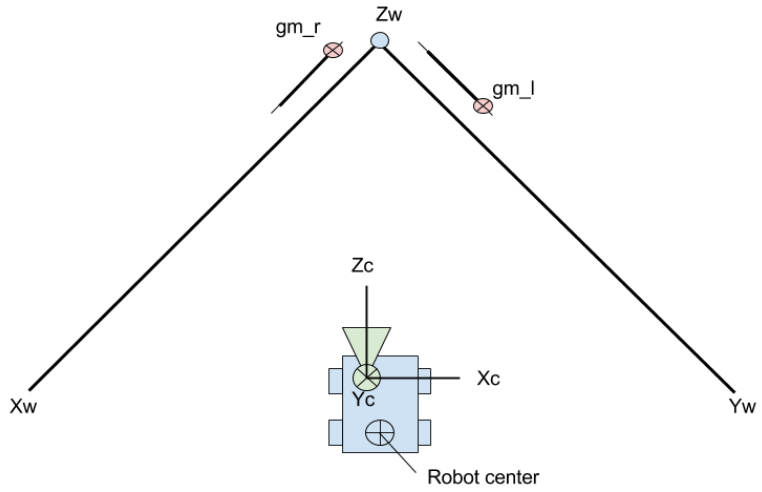


Figure 2.13: Illustration of robot in the world for localization testing using vision

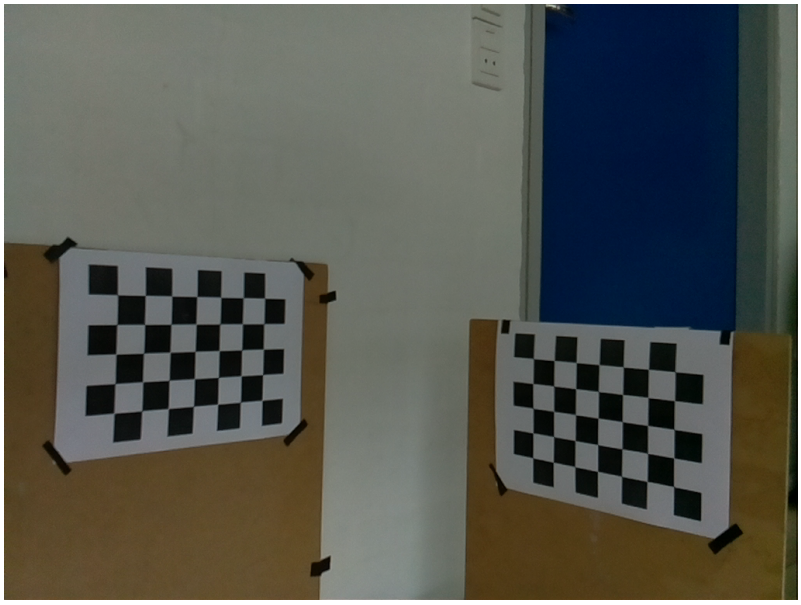


Figure 2.14: Image taken by the robot of the chessboard guidemarks

results further, instead more extensive test results are presented in the chapter regarding validation of our results.

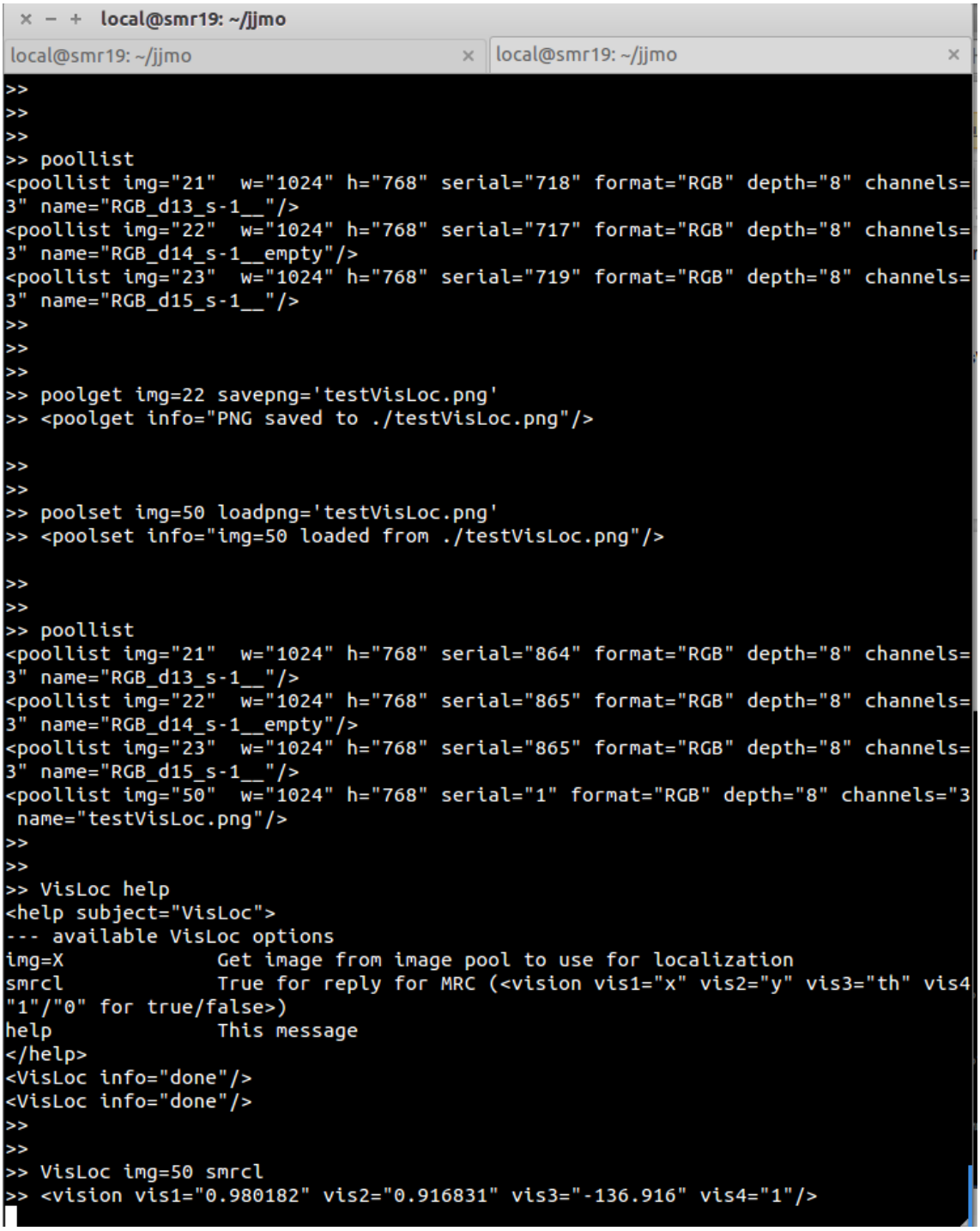


Figure 2.15: Result using the plugin

3 Laser Sensor

This chapter states the approach and results of investigating the robots laser sensor.
The benefits of averaging multiple scans as well as the accuracy of the laser sensor and the line estimation using the scan results were examined.

3.1 Overview Laser Sensor and Measurement Environment

The mobile field robot was enhanced with a laser sensor, which is mounted to the bottom front of the vehicle. The active, exteroceptive time-of-flight sensor measures the distance to its surrounding objects and has an angular range from $-90°$ to $+90°$. The angular resolution is $0.3516°$. This results in 512 distances for each laser scan within the $180°$ angular range (see figure 3.1).

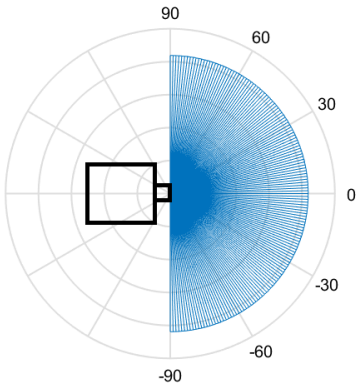


Figure 3.1: Angular range of the laser sensor

The measurement environment consisted of a square box with an edge length of $180cm$. The robot was placed inside this box to carry out the laser sensor measurements (see figure 3.2). This environment ensured precise knowledge of the robots surrounding to detect offsets and irregularities within the laser scan.

Figure 3.2: Measurement Environment

3.2 Averaging of Multiple Scans

As described above, the laser sensor returns multiple distances for each scan. Due to the imperfect precision of the sensor, these distances differ among different scans from the same position. Therefore it was investigated if averaging multiple scan results from the same position result in higher precision.
Apriori, the scan results were checked for gaussianity. Multiple laser scans were carried out from the same position to obtain multiple results for each scan angle. The array of results for each angle was then checked for gaussianity using the Kolmogorov-Smirnov test in MATLAB. This test revealed normal distribution for each scan angle.
As the results are normally distributed over multiple scans, the mean of several results will be more precise. The increase of precision was determined next.

To ensure the scan results are not falsified by possible offsets of the laser sensor or a longer distance to the object for some scan angles (which will result in less precision), the following test procedure was chosen:
The robot was placed in the middle of the square box, performing 50 scans facing one wall. Eventually, the robot was turned 5° and another set of scans was performed. This was repeated 19 times until the robot turned 90°, facing the next wall of the box.
The gained scan data was then transformed from the robots coordinate system to Cartesian world coordinates and the average of the 50 distances per scan angle was determined.

The averaged scan results in the world coordinates and the real walls are shown in 3.3. It can be seen that the scan results show a light bend. This will be dealt with in section 3.3. It can also be seen that the distance to the right and to the lower wall is greater than the distance to the upper and left wall. This can be affiliated to a slightly incorrect determination of the midpoint of the square.

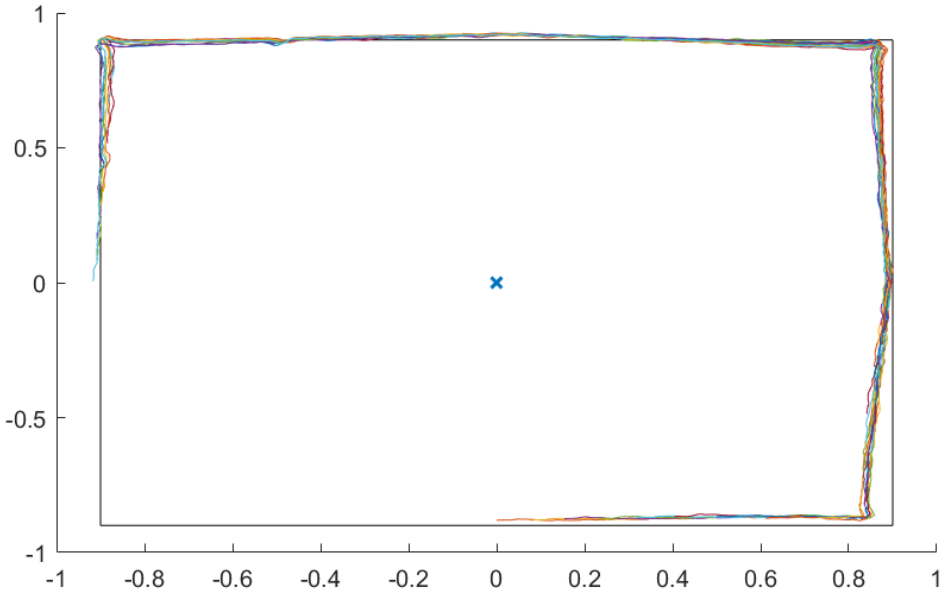


Figure 3.3: Averaged laser scan results for different angles in world coordinates

Subsequently, the distance from each averaged result to the closest wall was determined in MATLAB. Additionally, the distance from a not-averaged data set to the wall was determined.

Knowing the precise position of the robot during the scan and the precise position of the walls, the distance from the scan data to the walls describes the error of the laser sensor. The error of the averaged scan data to the not-averaged scan data was then compared. This is expressed in 3.4. The graph shows the offset of the averaged and not-averaged data for each scan angle.

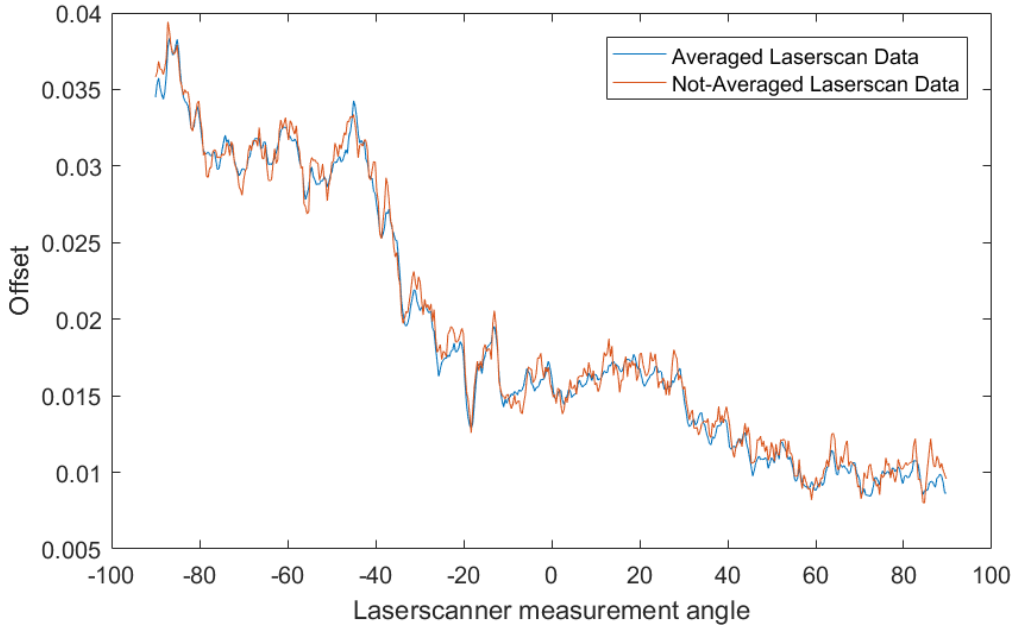


Figure 3.4: Offset from laser sensor scan data to actual walls

The graph shows almost no advantage of using averaged scan data over using not-averaged scan data. The gain of precision is as little as 1.588%.
Therefore the use of averaged scan data will not further be pursued in this report.

It can also be observed, that the offset for angles from $-90°$ to $0°$ is greater than from $0°$ to $+90°$. This can be explained with the slightly incorrect position of the squares midpoint, as the higher offset corresponds to the lower and right walls. Nevertheless, the different offset for different scan angles was further pursued and will be described in detail in section 3.4.

3.3 Offset in Line Estimation

The data obtained by a laser scan is further used to localize the robots position. A RANSAC algorithm [11] is used to estimate lines from the scan data and these lines are subsequently matched with the walls and surroundings known by the robot.
As seen in figure 3.3, the laser sensor does not return straight lines. Hence, it was of interest to investigate the accuracy of the line estimation to obtain information about the localization precision. This approach is covered in this section.

To examine the line estimation, the same scan data as in section 3.2 was used.
A MATLAB script was developed to separate the scan data for each angle that correspond to a wall for every scan. Following, a RANSAC line estimation was implemented in MATLAB. The scan data corresponding to a certain wall was then fed into the RANSAC algorithm to estimate a line. This estimated line was subsequently compared to the real walls to determine the accuracy of the localization.
Figure 3.5 shows the estimated lines (coloured lines) compared to the real walls. It can be seen that the error derived from the laser scan certainly affects the line estimation. The estimated walls cross the actual wall in one point, but show an obvious angular error.
This behaviour can be explained by the bended lines obtained from the laser scan and the principle of the RANSAC algorithm. As RANSAC aims for the highest number of points within the threshold of a random line, the estimated line along a bended graph will always go along the right or the left side of the graphs turning point. This is visualized in figure 3.6.

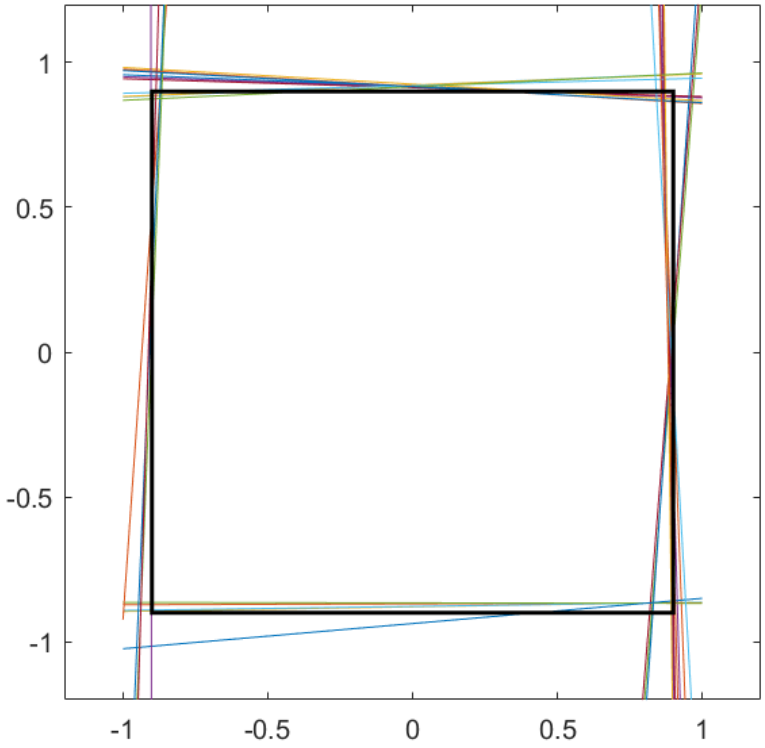


Figure 3.5: Estimated lines using RANSAC compared to walls

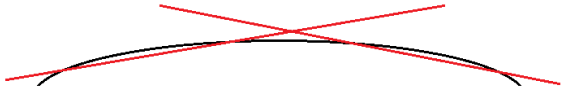


Figure 3.6: Symbolic RANSAC line estimation along a bended graph

The use of RANSAC displayed a striking problem when it comes to line estimation and therefore a problem for high precision localization. Especially the initial angular position of the robot is important for long odometry travel, and this cannot be determined as precise as needed with the current laser sensor data. Hence, further investigation of the laser sensor will be carried out in section 3.4.

3.4 Offset per Scan Angle

After revealing the impact from the error of the laser sensor data to the line estimation, this section will go further into detail on the laser scan offset. The goal was to obtain general statements about the offset per scan angle of the laser sensor in order to compensate these errors.

To gather more information about the error, a new set of measurements was carried out in the measurements environment. The test procedure is depicted in figure 3.7 and 3.8.
Nine scan positions with different distances to the facing wall and the perpendicular walls were chosen. Additionally, the angle in each scan position was varied by $+20°$ and $-20°$ with respect to the position facing the wall directly. This setup covered multiple distances to the surrounding walls as well as multiple angles for each position to give evidence of potential sensor offsets, for a total of 27 scans.

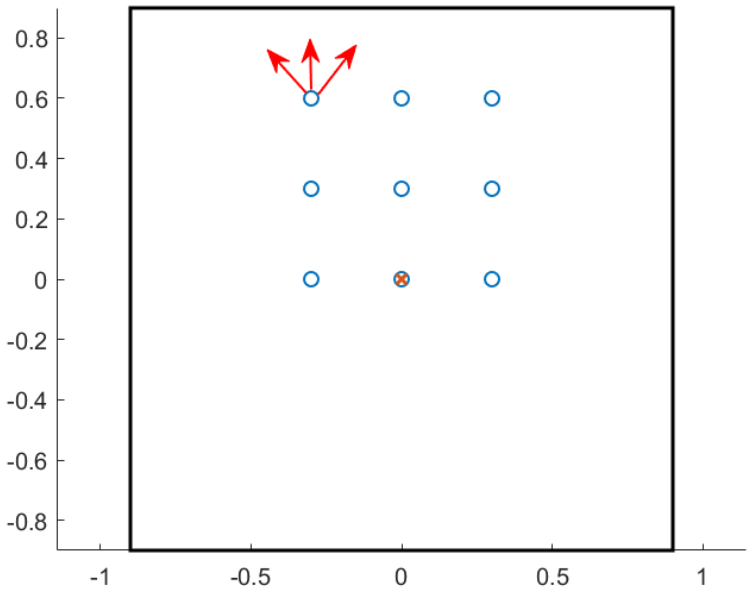


Figure 3.7: Symbolic depicition of the measurement positions

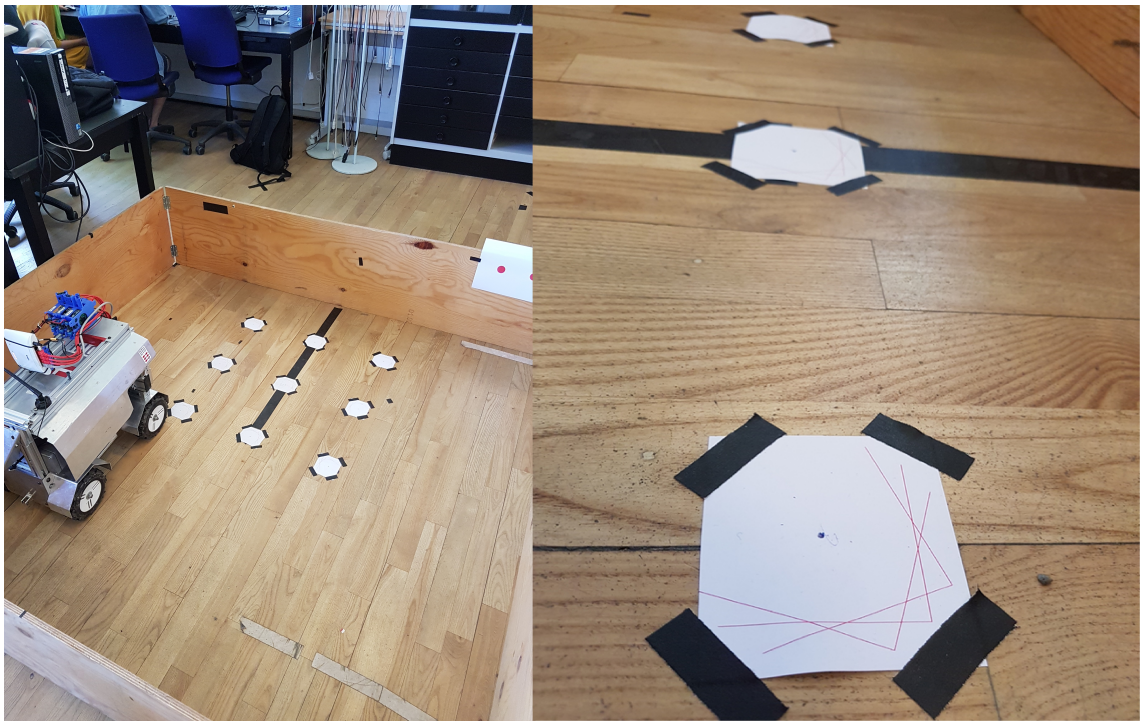


Figure 3.8: Real depicition of the measurement positions

Similar to section 3.2, the scan results were transformed to Cartesian world coordinates and the distance of each scan and each scan angle to the closest actual wall was determined. With the so generated data, the offset for different scan position could be described graphically and mathematically. Figure 3.9 shows the offset from the laser sensor to the real wall for all scans carried out in this test averaged.

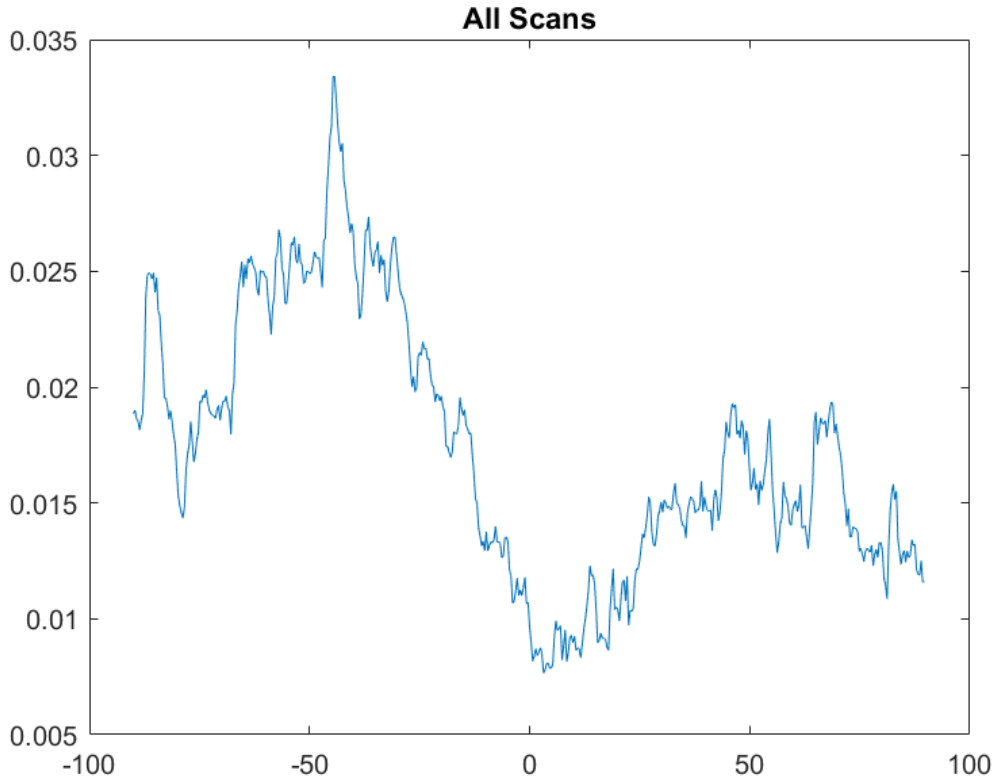


Figure 3.9: All scans: offset from laser sensor to real walls

It can be seen that the laser scanner indeed has a higher offset for the angles from $-90°$ to $0°$ than from $0°$ to $+90°$. Analogous graphs have been created only taking certain scan positions into account (e.g. short distance, long distance, facing left, facing right). These graphs can be found in the appendix (section 7). Striking about all graphs, beside the significantly higher offset for one side, is that the most exact measurement carried out by the laser sensor is done at $0°$.

The graph combining all scan positions (figure 3.9) is similar to the graphs taking only certain positions into account. Therefore it is representative for the whole test. It provides foundation to make further efforts to compensate this error mathematically. An outlook for a compensation will be discussed in section 3.5.

3.5 Mathematical Compensation

Due to the short time period of this project, a mathematical compensation could not be carried out. Nevertheless, this section gives an outlook on how the presented data about the laser sensor error can be used to compensate the offset.

The distances determined and shown in figure 3.9 and 3.4 are the shortest distances from the result of the laser scan to the closest wall. This does not represent the distances that should be compensated for. It provides the foundation for further trigonometrical calculations to determine the distances to compensate for. This is pictured in figure 3.10.

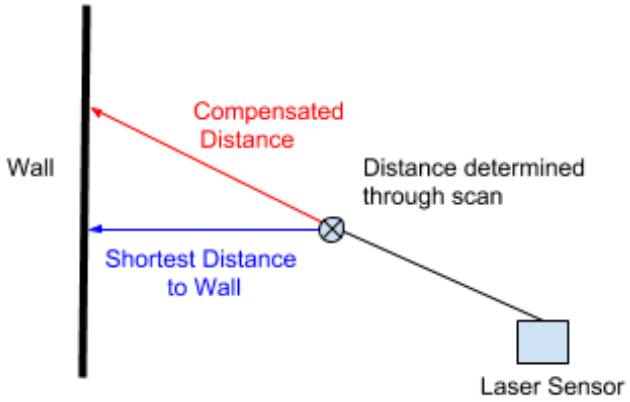


Figure 3.10: Compensation Distance

A proposed method for offset compensation is the fixed value compensation. As described in section 3.4, the offset from laser sensor to the walls for all scans (figure 3.9) is representative for particular scans carried out with different distances and angles to the walls. Therefore the compensation can be calculated based on these offsets per angle and simply be added to any future scans.
This will not provide the best solution but can be implemented fairly easily.

Another proposed method for offset compensation is the flexible value compensation. A deeper look can be taken into the determined offsets for each distance and angle of the ”nine point” scans. Based on this a function depending on distance on angle of the robot to its surrounding objects can be developed.
This method will provide a better compensation for the laser offset than the fixed value compensation but is significantly more complex to implement.

To prove the effect of the compensation, the RANSAC line estimation can be used again. The estimated line for compensated and not compensated can subsequently be compared to the real walls.

4 ODOMETRY

Improving the odometry of the robot was not into the objectives, but after making some measurements it was seen that it needed to be compensated at some way.

4.1 ODOMETRY MEASUREMENTS

The measurements that were done are the next:

- 7 meters forward. 10 measurements were taken, with an offset average of -18.1cm in the x axe where all of them were within a range of $\pm$1.2cm.
- 7 meters backward. 10 measurements were taken, with an offset average of 11.1cm in the x axe where all of them were within a range of $\pm$1.1cm.

Although it can look like unusual, there was a difference in the offsets between going forward and backward, which could be caused by a little deviation in the front wheels.
In the case of the y axe, the measurements are less precise, which is because, to do it properly, it would be needed a perfect environment. For example, if the robot is initially positioned 0.5° off then, after 7 meters, the difference in the y direction will be 6cm. Therefore, no compensation was made for this axe. It was appreciated, in the taken measurements, that the offset in this y varies within the range of $\pm$7cm.

4.2 ODOMETRY COMPENSATION

Due to the odometry of the robot cannot be changed, it is necessary to compensate these errors at a high level, directly in the 'mrc' scripts.
To do this, it has been calculated the percentages of the obtained offsets and added to the distances that the robot has to run. It will not change the odometry, but the robot will go over the desired distance.
Then, using the compensation, the distance used in the script to run 7 meters forward will be $dist = 7 + (7 * 0.0258)$, where 0.0258 correspond to the percentage error.
In the backward case, the needed compensation is lower than before, so the distance will be $dist = 7 + (7 * 0.01587)$.

4.3 RESULTS AND CONCLUSIONS

Doing the measurements again, now with the correct parameters, the obtained results have been very good. The average offset for both forward and backward are almost zero and, as expected, the range keeps being the same. Hence, it is possible to say that now, for the chosen distance, it is not only precise, it is also very accurate.
Then, it was decided to make it run further in order to see if the compensation is valid for any distance. Two different distances was chosen:

- 30 meters forward. 2 measurements were taken and an offset average of -24cm was obtained.
- 24 meters forward. 2 measurements were taken and an offset average of -22cm was obtained.

Looking at these new measurements, it is clear that the compensation cannot be made for every distance, but without it, the offsets in these two cases would have been around -70cm.

Therefore, if a good accuracy for longer distances is desired, there are two ways of making it. Whether dividing the distance in more than one path, or changing the compensation percentage in the script.

5 VALIDATION OF THE RESULTS

Once implemented the visual pose estimation and with the accurate odometry, three tests have been made in order to see the improvement of the navigation. First, only using the laser scanner to obtain the initial position; second, only using the visual pose estimation and third, mixing both of them.

The environment that has been used to make the tests (figure 5.1) is very simple, and consists in:

- A 90° corner where the two guidemarks are positioned in each of its sides.
- The point that corresponds to the initial position of the robot, which is, in the world coordinates, (0.9 , 0.9).
- The point that corresponds to the final position of the robot, which is in the coordinates (9.9 , 0.9).

Figure 5.1: Tests environment

The robot has been positioned in the initial point facing to the corner, with an approximate angle of -135°. From that position the localization is made, using the laser scanner, the camera, or both of them. Once the localization returns the exact coordinates and angle of the robot, it uses this information to drive to the goal. Then, it is in the final position where the measurements are made to see the error between the theoretical position and the one obtained.

5.1 LASER SCANNER POSE ESTIMATION MEASUREMENTS

The first test was made using only the laser scanner to estimate the initial position. In the Table 5.1 it can be seen that the x axe is very accurate, but in the y the offsets are much bigger and with the robot always tending to the right.

That is due to a small difference between the real angle of the robot and the one obtained after the scanner. For example, just 1° will make an offset, after driving 9m, of 15cm in the y direction. This is why it is very important to be very accurate in the estimation of the initial angle.

Table 5.1: Laser pose estimator measurements

Offset		**Initial position**		
x (cm)	y (cm)	x (cm)	y (cm)	th (deg)
+0.1	-8.2	0.903	0.928	-132.24
-0.1	+4.7	0.902	0.935	-130.28
0	-23.7	0.890	0.919	-131.64
0	-15.4	0.906	0.926	-133.98
+1.1	-27.8	0.911	0.929	-133.13
-3.4	-56.1	0.863	0.949	-134.41
-0.2	-25.6	0.882	0.920	-131.24
-0.9	-18.5	0.893	0.912	-131.39

5.2 VISUAL POSE ESTIMATION MEASUREMENTS

The second test was made using only the new implemented visual part to obtain the initial position. In the Table 5.2 it can be seen that, as in the previous test, the x axe is very accurate, but now, in the case of the y, the robot always tends to the left.

Table 5.2: Visual pose estimation measurements

Offset		**Visual Initial position**		
x (cm)	y (cm)	x (cm)	y (cm)	th (deg)
-3.8	+25.3	0.928	0.915	-135.73
+6.2	+41.1	0.860	1.192	-121.86
-0.8	+18.2	0.873	0.877	-133.46
-4.5	+28.2	0.747	0.997	-123.57
+1.1	+32.8	0.914	0.904	-138.82
-2.5	+38.8	0.924	0.901	-140.15
-0.8	+26.7	0.923	0.877	-131.23
-5.2	+31.1	0.912	0.892	-134.72

5.3 Laser and Visual pose estimation measurements

The third test was made using the two techniques which have been talked about above. Due to the individual offsets obtained before in the y axe are similar, but in the opposite direction, calculating the mean between both initial angles it is possible to compensate them in order to get accurate results.

In the Table 5.3. it can be seen the big improvement in the y axe while the x keeps being very accurate.

Table 5.3: Laser and Visual pose estimation measurements

Offset		**Laser Initial position**			**Visual Initial position**		
x (cm)	y (cm)	x (cm)	y (cm)	th (deg)	x (cm)	y (cm)	th (deg)
+0.2	-6.3	0.909	0.885	-133.05	0.927	0.805	-136.73
+0.4	+5.4	0.919	0.933	-134.21	0.930	0.857	-136.6
+0.1	+11.8	0.883	0.941	-132.16	0.906	0.863	-135.22
+2.1	+11.6	0.903	0.962	-131.54	0.924	0.886	-135.18
-1.5	-6.2	0.887	0.903	-133.85	0.911	0.820	-138.08
-0.4	0	0.887	0.944	-133.76	0.901	0.879	-136.48
-3.2	-11.1	0.931	0.947	-135.52	0.957	0.960	-140.10
-0.2	-5.9	1.000	0.954	-132.53	1.015	0.894	-135.42

5.4 Offsets representation

In the figure 5.1 is shown the points that correspond to the offsets displayed in the previous tables. With the graphical representation it is easier to realize about the improvement that has been got because of implementing the localization through the guidemarks.

The green points indicate the final position of the robot using only the laser scanner and, as it was explained before, it tended to go to the right.

On the other hand, the blue points indicate the final position of the robot using only the vision pose estimation and, as it was explained before, it tended to go to the left, compensating the offsets caused by the localization from the laser scanner.

Hence, thanks to this compensation, the final results are more than satisfactory. As it is possible to appreciate in the figure 5.2 all the point are, more or less, within the 10cm radius circumference. This means that the estimation of the initial angle are almost perfect, because 10cm after driving 9m corresponds to an angle of 0.66°. Therefore, the estimation error of the initial angle will be [-0.66 , 0.66], but, being aware of the odometry errors, it is possible to say that the obtained angle is even more accurate than 0.66°.

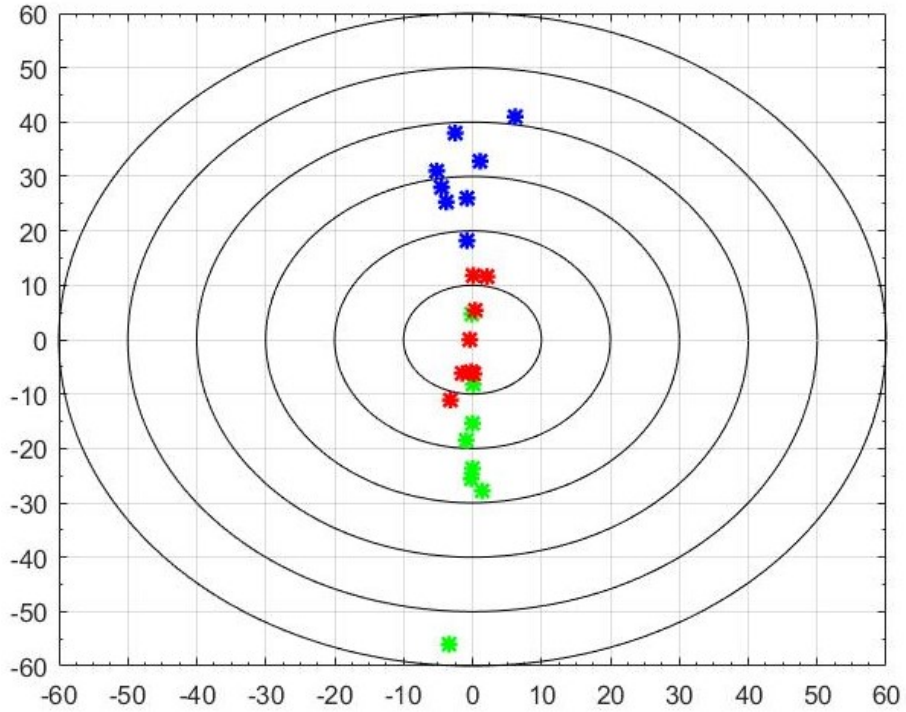


Figure 5.2: Offsets representation

5.5 Loop navigation

Once all the measurements have been done and with the good results shown in the section 5.3, it was the moment of testing the robot running in a loop, where it had to go from the initial point to the final and come back during many times. It is important to say that the only place where localization was used is in the first point, therefore every time the robot came back to the initial position it calculated its angle and coordinates and then, it started running again.

The results of the tests were good, the script was programmed to make the loop 10 times and the robot usually reached them without any problem, although in some cases it did not finished, whether because of a failure in the robot (broken pipe), or because in the moment to localize with the camera, it did not get the image of the two guidemarks.

This last problem can be easily solved by getting the guidemarks closer to the corner and also, setting the initial position a little further in order to gain more range in the camera vision.

The final test was made with the presence of the professor Nils to verify the proper functioning of the task. In this case, the robot ran around 7 times, keeping the correct path until in one of the moments of localizing with the camera it did not get the image of the two guidemarks and stopped.

6 Conclusion and Outlook

Looking at the visual localization, it can be concluded that for a first implementation it works quite good. The biggest issues for this type of localization is to get all the measurements of the camera on robot frame correct, as well as the camera calibration. Combining this with the fact that we did not have a 100% static test environment, i.e. the environment were wood planks and paper guidemarks positioned in the world with rulers and fixed with tape, the overall performance was satisfying.

As it were seen, half of the localization runs with pure vision, where we had offset to one side. This speaks to the fact that there could have been a measurement error somewhere in the system. If the time had allowed it, this would have been needed to be investigated. Also, taking the odometry error into consideration, the results could have been improved slightly.

In general, when localizing and driving a robot with only sparse landmarks, the overall precision of the system is limited to how well the landmarks position can be determined and the precision of the odometry. The natural next step for an implementation like this, would be to implement visual odometry and combining it with the inertial odometry that we use now. Visual odometry does not need a static environment to work.

There are three main recommendations to take into account for possible future improvements:

- Improve the visual localization until the current offset is fixed. It should not be difficult because, according to the measurements, it is seen that the estimation of the initial positions are quite accurate (all the final position obtained are close to each other).

  If the vision localization is improved, then, it would let us localize without the laser scanner and use only the vision part. This would be useful in environments where the laser is not available.

- Now, as it has been commented before, the vision range of the camera is not very wide, which can carry out with problems when it tries to get the picture of the two guidemarks. If one of the guidemarks is not taken completely in the mentioned picture, then it will not able to localize.

A solution for this would be to make use of the three cameras of the field robot, which would allow us to cover a range of around 180°.

- Currently, the laser scanner is not precise enough, as it has been explained in the chapter 3. Therefore, a solution for this would be to make a plugin where it is implemented the improvements we have made in Matlab and, because of lack of time, it has not been possible to try in the robot.

  Then, as in the case of the visual localization, if the precision of the laser scanner is good enough, it would let us to navigate only with the laser in environments where it is not possible to fix the guidemarks.

Additionally, a combination of visual and laser localization can be implemented. This would provide two high-precision localization methods for finding the precise start position.

REFERENCES

[1] *Global Autonomous Mobile Robots Market 2017-2021*. URL: https://www.researchandmarkets.com/research/pxl836/global_autonomous.

[2] Peter Buxbaum. *Robotics and drones are coming to the warehouse*. URL: https://www.ajot.com/premium/ajot-robotics-and-drones-are-coming-to-the-warehouse.

[3] *Field to be farmed exclusively by robots – a world-first*. URL: https://harper-adams.ac.uk/news/202923/field-to-be-farmed-exclusively-by-robots--a-worldfirst.

[4] *Amazon Prime Air*. URL: https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011.

[5] Manolis Lourakis and Xenophon Zabulis. "Model-Based Pose Estimation for Rigid Objects". In: *Lecture Notes in Computer Science: Computer Vision Systems. ICVS 2013* 7963 (), pp. 83–93. DOI: https://link.springer.com/chapter/10.1007/978-3-642-39402-7_9.

[6] Daniel F. Dementon. *Model-Based Object Pose in 25 Lines of Code*.

[7] Kin Hong Wong Ying Kin Yu* and Michael Ming Yuen Chang. *Pose Estimation for Augmented Reality Applications Using Genetic Algorithm*. URL: http://www.cs.cuhk.hk/~khwong/j2005_smc_b_GA_Pose_draft.pdf.

[8] Xiao-Shan Gao et al. "Complete solution classification for the perspective-three-point problem". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.8 (Aug. 2003), pp. 930–943. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2003.1217599.

[9] C. P. Lu, G. D. Hager, and E. Mjolsness. "Fast and globally convergent pose estimation from video images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.6 (June 2000), pp. 610–622. ISSN: 0162-8828. DOI: 10.1109/34.862199.

[10] V. Lepetit, F.Moreno-Noguer, and P.Fua. "EPnP: An Accurate O(n) Solution to the PnP Problem". In: *International Journal Computer Vision* 81.2 (2009).

[11] Robert Collins. *Robust Estimation : RANSAC*. URL: http://www.cse.psu.edu/~rtc12/CSE486/lecture15.pdf.

7 Appendix

7.1 Laser Sensor

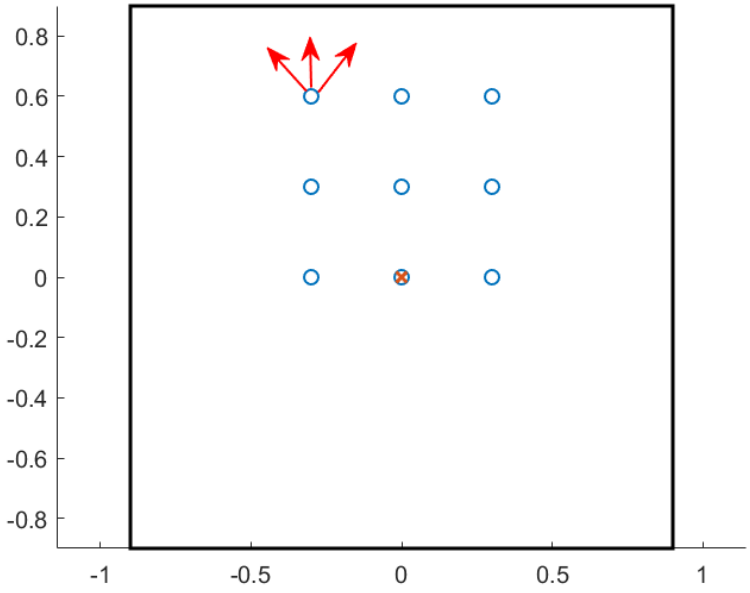


Figure 7.1: Symbolic depicition of the measurement positions

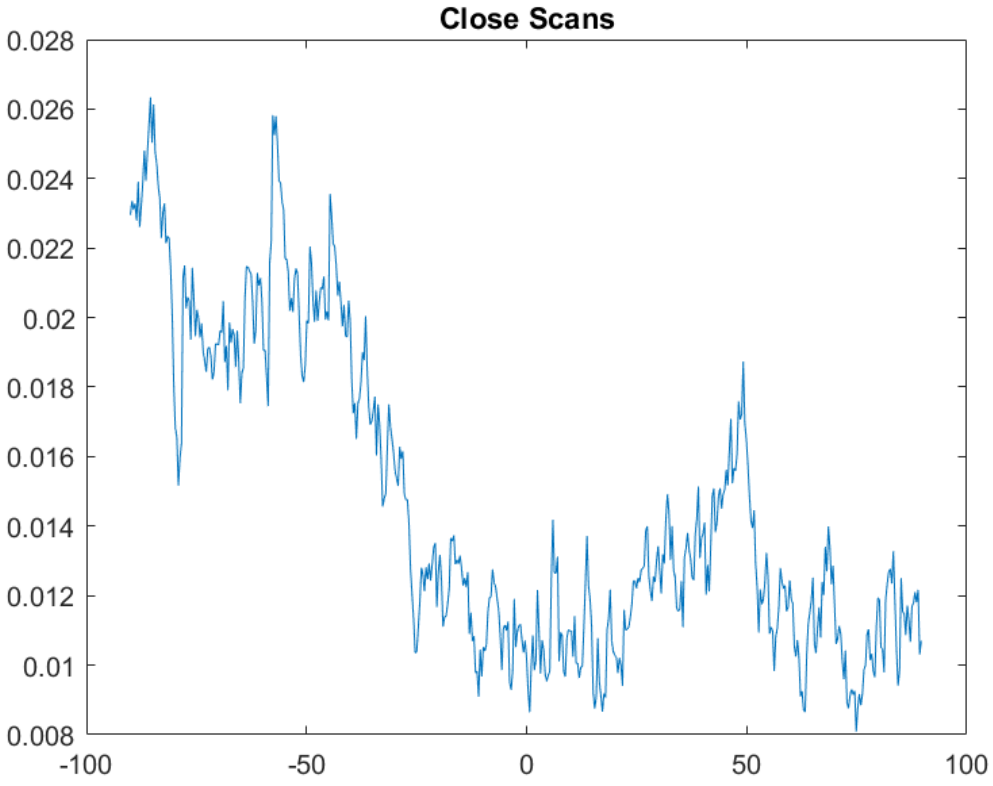


Figure 7.2: Offset close scans

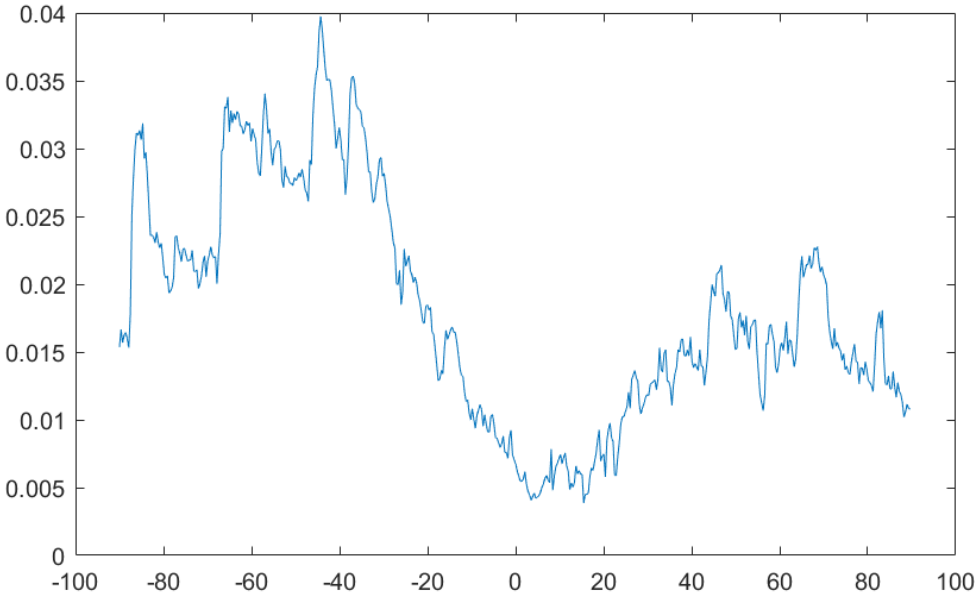


Figure 7.3: Offset medium distance scans

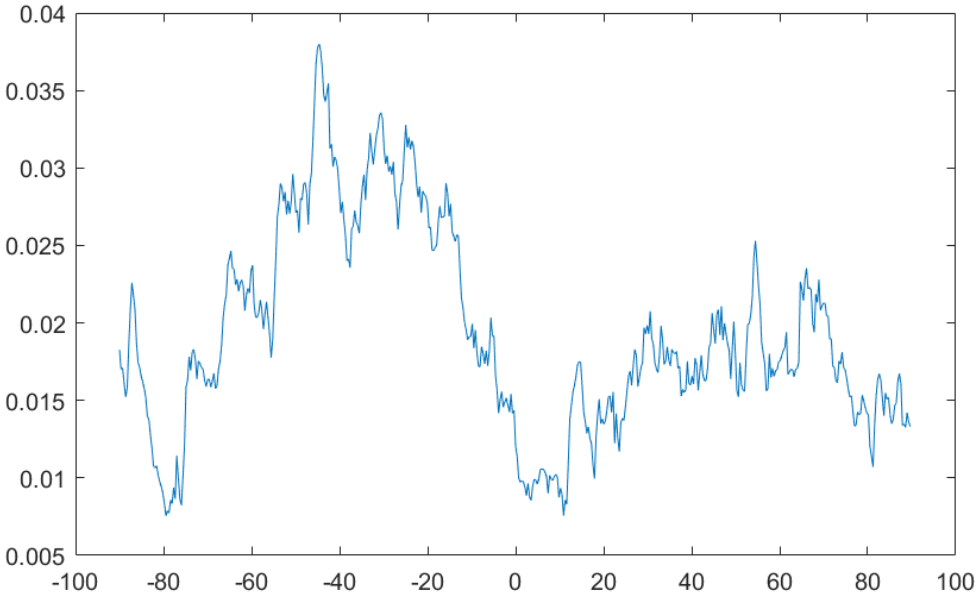


Figure 7.4: Offset distant scans

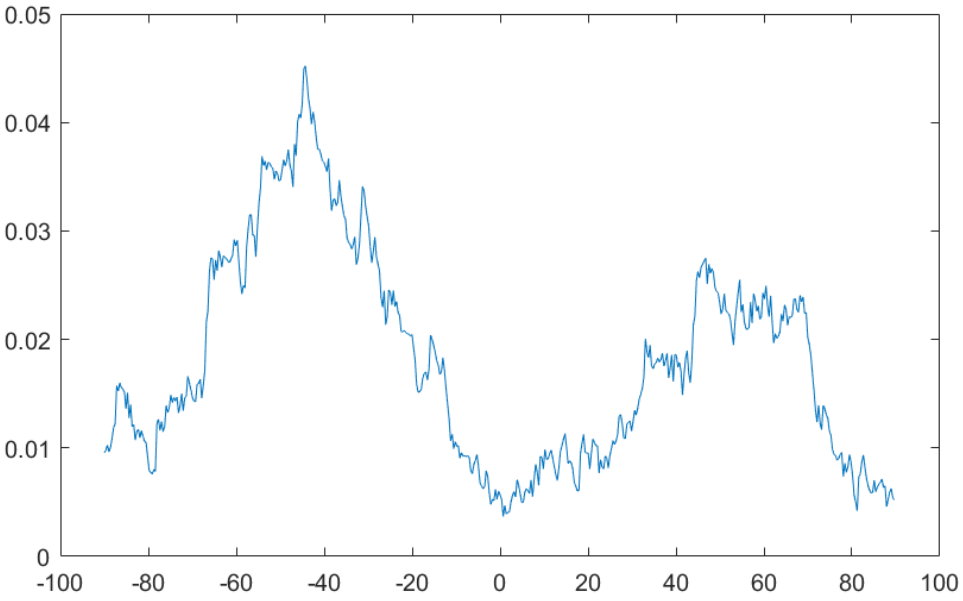


Figure 7.5: Offset left facing scans

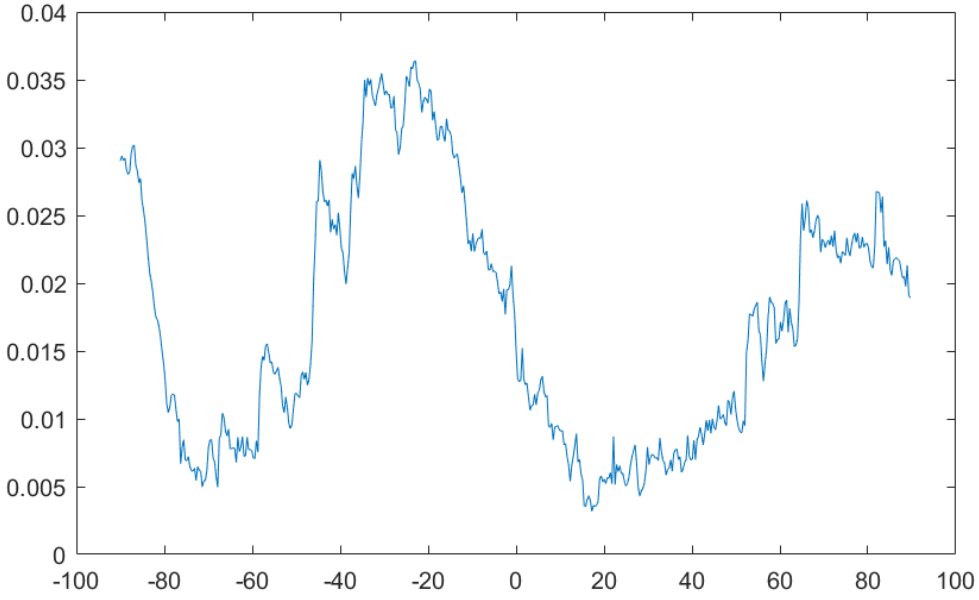


Figure 7.6: Offset right facing scans