Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Industriales

Máster en Automática y Robótica

Robot Guidance and Navigation

Robotic Navigation Strategies for a Nocturnal Surveillance Application

Group 7

Alberto Ibernón Jiménez (23079)
David Redondo Quintero (23147)
Josep María Barberá Civera (17048)

January 8, 2024

Table of Contents

CHAPTER 1

Introduction

This work is part of the course of *Robot Guidance and Navigation* of the Master in Automatics and Robotics of the Polytechnic University of Madrid (UPM). It tries to familiarize the student with the task of **programming the navigation of a ground robot in the presence of uncertainties**. First, a brief introduction to the selected robotic application is given. Next, the project's goals and their organization in the following chapters are explained.

1.1 Project Context and Motivation

The problem selected is that of **a night watchman robot**. It is suggested as a potential system to be implemented in the *Museo Nacional del Prado* (in Madrid). It can, in our opinion, improve security and make the job of human guards easier. Depending on the number of robots used, a swarm of robots could provide continuous, complete, or nearly complete coverage given the size of the proposed museum.

The project will use *Matlab* for coding and programming, as well as *Apolo*, a simulator developed by the Robotics and Automation Center associated with the UPM. The simulators provides the possibility to be connected with Matlab and command a robot. *Marvin*, a Pioneer 3-AT type robot will be used for the simulator (see Figure 1b), its locomotion system actually being that of a differential robot. Our research will only focus on a single robot, with the procedure being the same for all other swarm agents. Moreover, it will only be able to monitor one museum floor, with no way to go down or upstairs. Regarding the exteroceptive perception systems, ultrasonic sensors has been selected for reactive control and a laser (LIDAR) in point cloud mode for the localization, as seen in Figure 1a. An additional proprioceptive system that is available is the robot's own odometry. The robot's localization will be made possible by the odometry and laser working together.

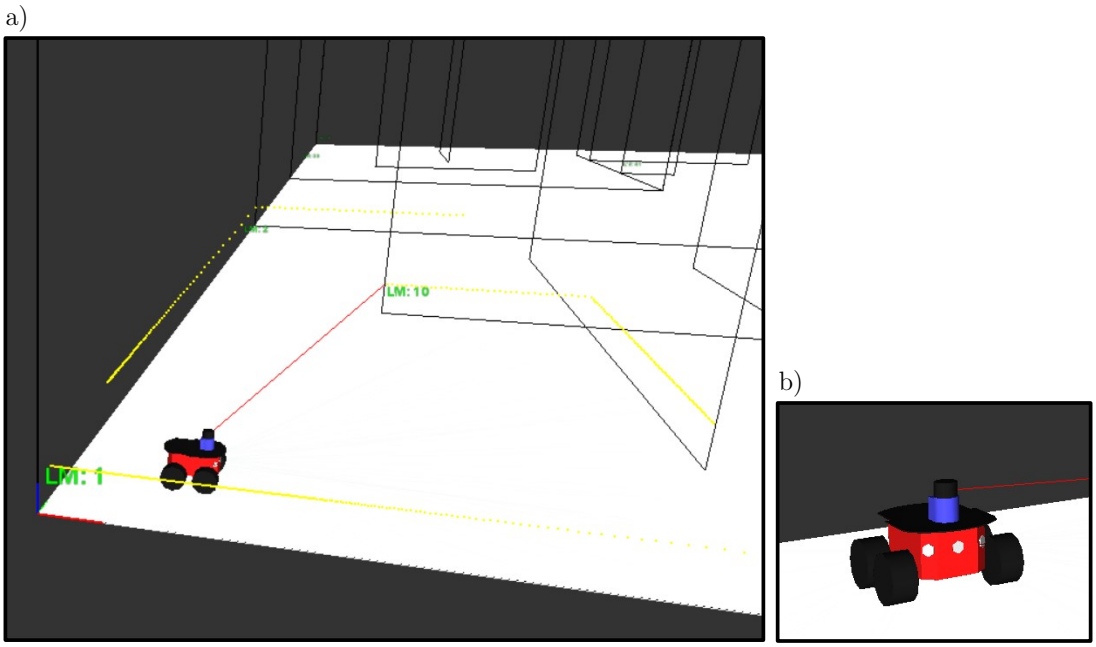

Figure 1: *Apollo simulations. In (a) Marvin is in the initial position with the laser activated while (b) shows Marvin closer and the laser beacon sensor can be identified.*

This selection and the assumptions about the application have conditioned the map (the robot's navigation environment). As can be seen in Figure 2, it presents areas with narrow corridors, small rooms, but also wide spaces and oblique walls. In this way, we have tried to generalize the possibilities to be found in a real museum.

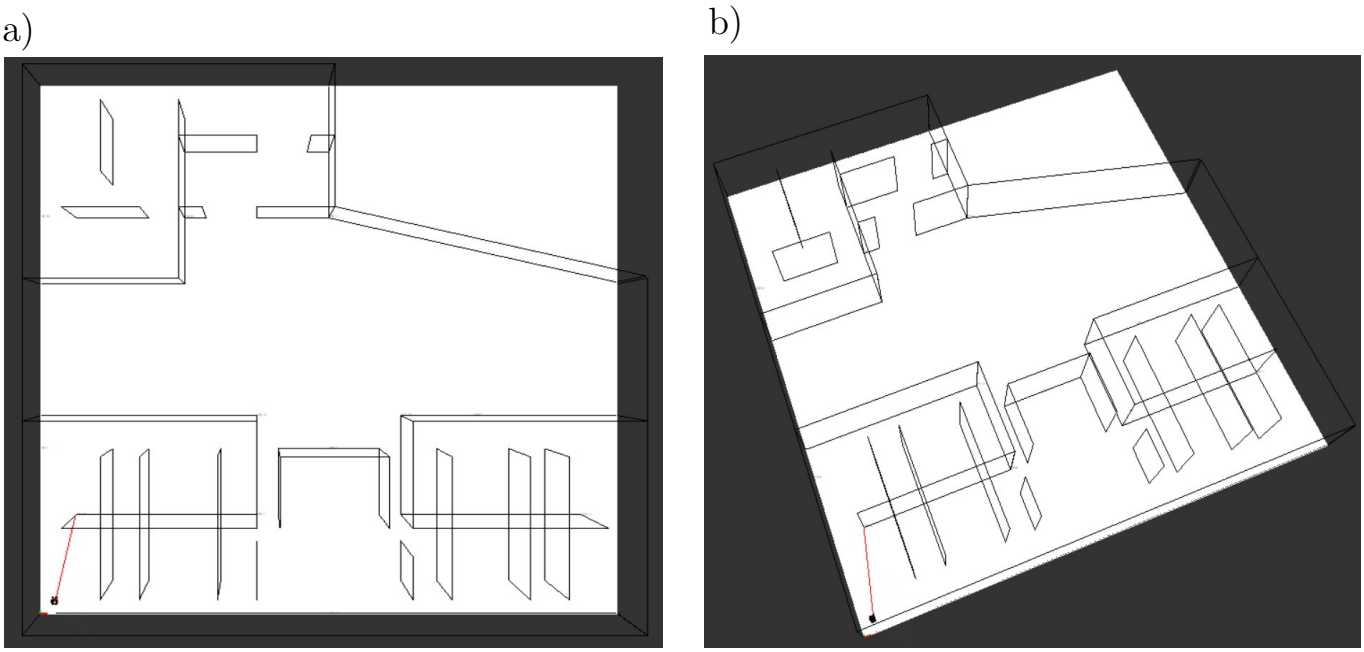

Figure 2: *Visualization of the map in Apolo. (a) Top view. (b) Perspective view.*

1.2 Objectives of the Project

Once the application/problem requiring the navigation of a terrestrial robot has been chosen and the map of the environment to be worked on has been defined, we can establish the following objectives for the present work:

- By using a differential type locomotion system: determine the **uncertainty associated with the kinematic model** (Chapter 2).
- Chosen the **perception system**: **determine its uncertainty** by means of calibration (Chapter 2).
- For the **localization**: implement an algorithm that allows to obtain at any time the absolute position of the robot **using the Extended Kalman Filter**. Test and experiment with this algorithm to **analyze the influence of the parameters** and the goodness of the estimation achieved (Chapter 3).
- For the movement: **program a controller** that allows the robot to follow a defined trajectory. **Include a reactive control** that allows to avoid obstacles or to navigate through narrow places (Chapter 4).
- For trajectory generation: **integrate a planner** (Chapter 4).
- **Encapsulate everything** as a navigation algorithm as a proposal to the proposed application (Chapter 5).

1.3 Software Architecture

The Robotic Navigation and Control has been implemented following a modular methodology that allows to perform individual regression tests to ease the validation process. It permits to distinguish the several parts involved in the project. It is structured in the following modules:

- **Init.m:** responsible for reading the initial and goal poses, inputs of the simulator. Then, the module places the robot real pose in the initial pose and resets the odometry. It also defines the sensors noise and estimation covariances. Finally, it initialize the time variables and the store matrices.
- **Planificador-global.m:** once the initial and goal poses are identified, the path planning module is executed to generate the waypoints that connect both points, based on RRT-star. This planification process is done off-line, previous to the running of the pseudo real-time (in-the-loop) code. The output of module is a matrix with the waypoints coordinates.
- **Planificador-local.m:** responsible to generate the intermediate reference points that the robot needs to chase between two waypoints during the in-the-loop execution. It permits path re-planning from the last estimated position to the next waypoint once the previous sequence of reference points has been completed, which enhances the success of the path chaser task. The module also extracts the derived forward and rotation velocities commands, computed when the reference points are correlated with the time. These commands are considered as an active control.
- **Navegacion.m:** responsible for taking the information from the proprioceptive (odometry) and exteroceptive (landmarks) sensors which are integrated in the Expanded Kalman Filter (EKF) to obtain the estimation pose and covariance matrices.
- **Control-reactivo.m:** responsible for taking the reference pose and the estimated pose, computing the error between them and extract reactive velocity commands, from a PI controller, which are added to the active commands. It also identifies the present of a close wall or obstacle from ultrasonic sensors and generates a robot rotation, proportional to the distance to the objects, to move away from them. Once, the objects are not detected, the robot performs a replanning, in case of needing.
- **Dke.m:** the simulator takes the control velocity commands and executes them in the Dynamics and Kinematic Environment (DKE), although only kinematic effects are considered for this approach. In this way, the real position and orientation are obtained.

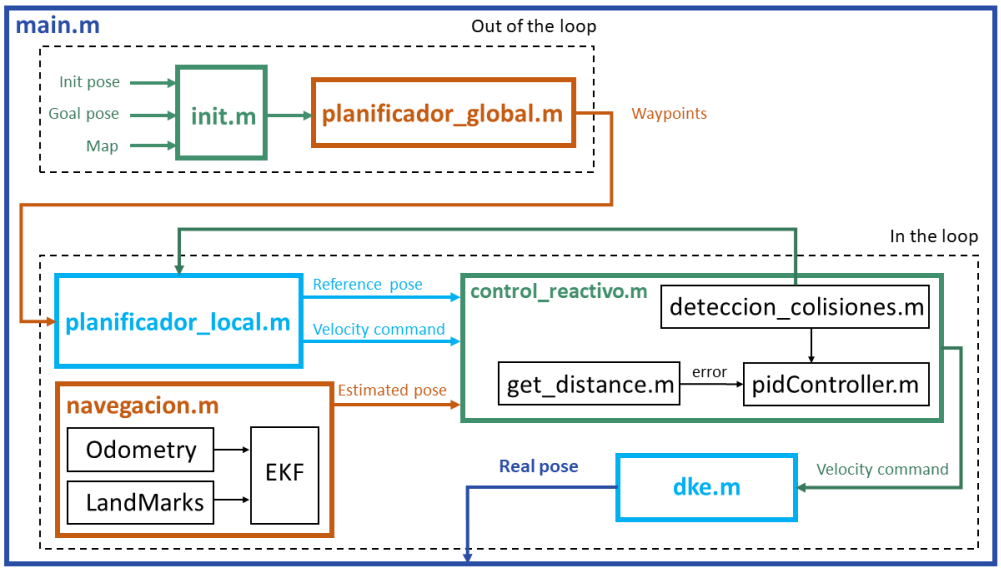

Figure 3: *Robotic Navigation and Control software architecture*

CHAPTER 2

Locomotion and Perception System

In this chapter, our focus is on measuring the uncertainty linked to the differential locomotion system of our robot using the functions offered by the Apollo simulator. Additionally, we determine the uncertainty of the selected perception system through small simulations and multiple measurements.

2.1 Differential Kinematic Model

The simplified equations describing the odometry model in a differential robot is as follows:

$$x(k+1) = x(k) + \Delta d(k) \cos\left(\beta(k) + \frac{\Delta\beta(k)}{2}\right) \tag{1}$$

$$y(k+1) = y(k) + \Delta d(k) \sin\left(\beta(k) + \frac{\Delta\beta(k)}{2}\right) \tag{2}$$

$$\beta(k+1) = \beta(k) + \Delta\beta(k) \tag{3}$$

Apolo provides functions to move the robot and to determine both the actual pose and the one provided by the odometry model. These are detailed below:

- **apoloMoveMRobot**: given the robot name, the desired speed (forward motion and rotation) and the time of the movement, it returns 1 if the robot has moved correctly and 0 if it hasn't.
- **apoloGetLocationMRobot**: given the robot name, it returns a vector of doubles: $\begin{bmatrix} x & y & z & \theta \end{bmatrix}$ where θ is the rotation angle at z.
- **apoloGetOdometry**: given the robot name, it returns a vector of doubles: $\begin{bmatrix} x & y & \theta \end{bmatrix}$ indicating the distance in meters calculated by the robot from the last reset or startup until the present moment.

2.1.1 Model Uncertainty

Thanks to the aforementioned functions provided by *Apolo* the error variance associated with the odometry is calculated as follows:

1. The robot is commanded to move with a constant velocity in the same direction for a fixed amount of time.
2. During this movement, the actual location of the robot is stored in a vector of measurements, as well as the odometry, both provided by the *apoloGetLocationMRobot* and the *apoloGetOdometry*, respectively.
3. The difference between these values (actual and odometry) is then calculated and saved.
4. Finally, the variance of the error in the measurements is calculated as the square of the standard deviation of the error vector.

We have performed four simulations, three consisting of rectilinear motion in three different directions: horizontal, vertical and diagonal, as shown in Figure 4, and the fourth simulation consisting of rotating the robot in a fixed position.

Figure 4: *Flat terrain designed for odometry calibration, with sufficient dimensions to accommodate the simulation duration (50 s) and the speed (1 m/s). x refers to the horizontal direction, y to the vertical and xy to the diagonal.*

As usual, we are considering that the forward movement and the rotation are not correlated so the covariance in not calculated and is assumed to be negligible. From the results obtained from the four simulations, it can be concluded that the variance in the advance error is similar regardless of the direction followed. On the other hand, it should be noted that in the non-horizontal movements (Figures 6 and 7) there is an offset in relation to the real measurements and the position estimation of 0.1 meters. With respect to the error in the rotation (see Figure 8), this presents a much higher variance than that of the advance. Finally, the variance and covariance matrix is as follows:

$$Q = \begin{bmatrix} 1.6072 \cdot 10^{-5}\ m^2 & 0 \\ 0 & 0.01334\ rad^2 \end{bmatrix}$$

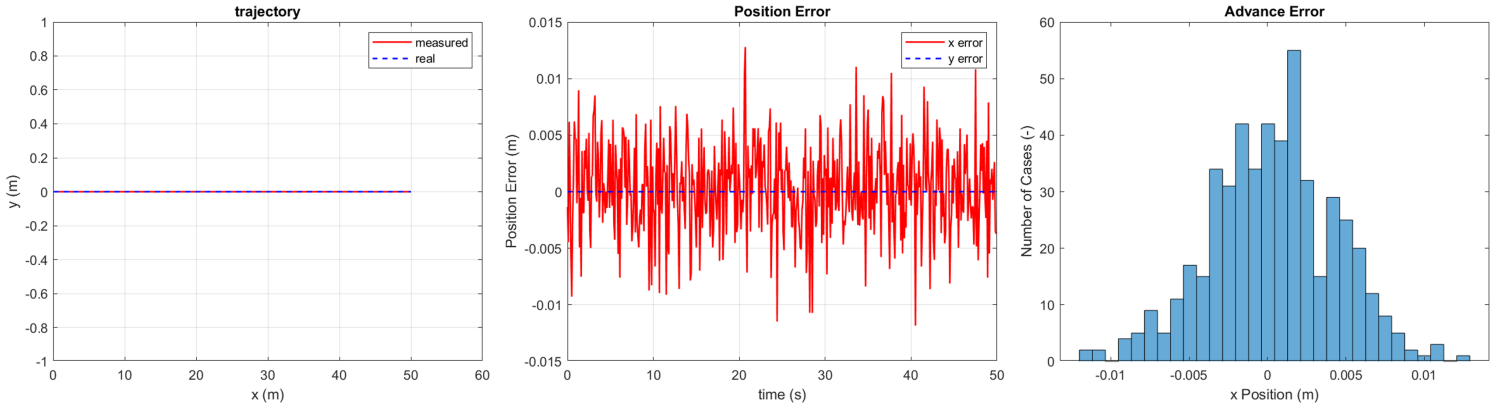

Figure 5: *Plots for the **horizontal** simulation. In the left, the trajectory. In the middle, the position error along this trajectory, it is centered in zero with a variance of $1.6072 \cdot 10^{-5}\ m^2$. In the right, a histogram of the error shows the expected Gaussian behavior.*

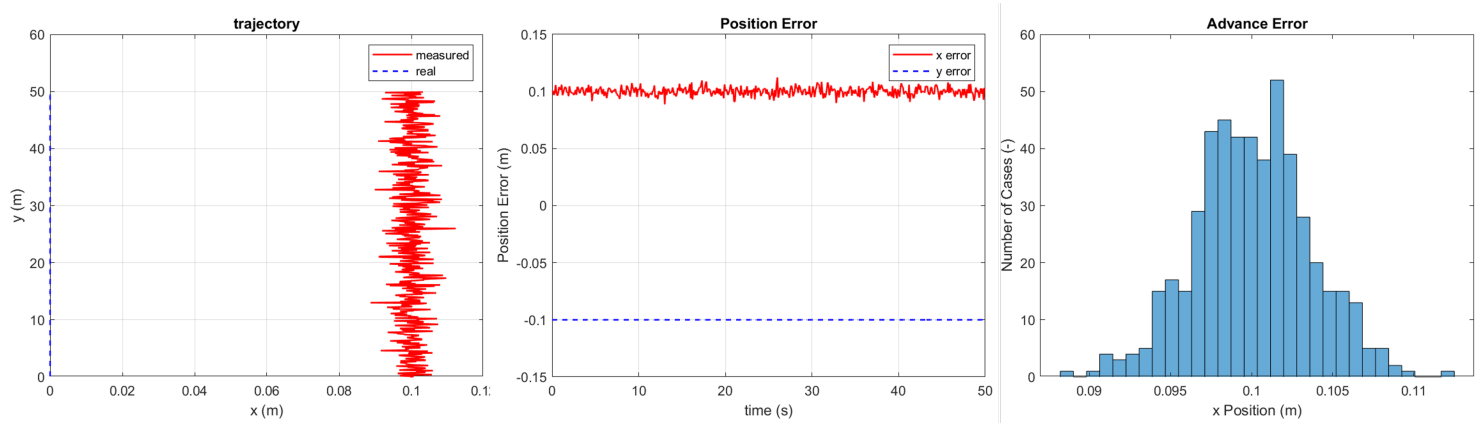

Figure 6: *Plots for the **vertical** simulation. In the left, the trajectory with an offset of* 0.1 *meters. In the middle, the position error along this trajectory, it is not centered in zero but in* 0.1 *with a variance of* $1.2719 \cdot 10^{-5}$ m^2. *In the right, a histogram of the error shows the expected Gaussian behavior.*

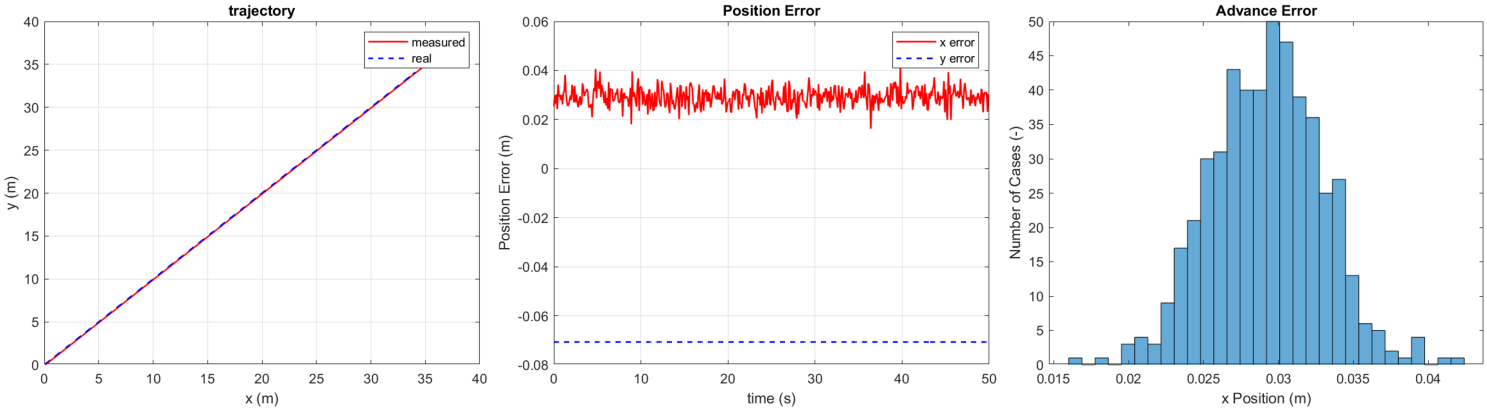

Figure 7: *Plots for the **diagonal** simulation. In the left, the trajectory. In the middle, the position error along this trajectory, it is not centered in zero but in* 0.1 *with a variance of* $1.4067 \cdot 10^{-5}$ m^2. *In the right, a histogram of the error shows the expected Gaussian behavior.*

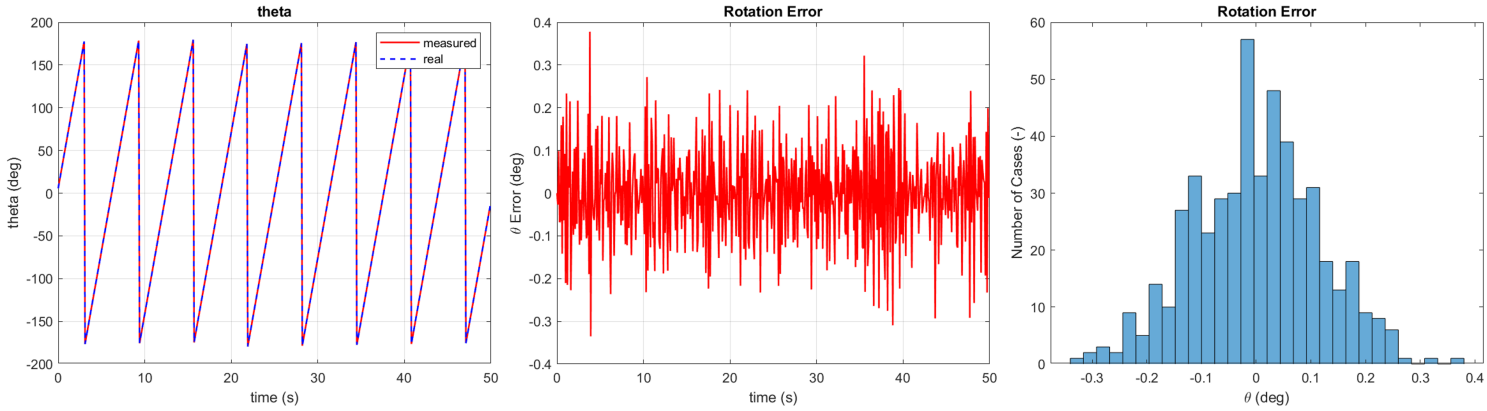

Figure 8: *Plots for the **rotation** simulation. In the left, the change in the angle of rotation (θ). In the middle, the rotation error, it is centered in zero with a variance of* 0.01334 rad^2. *In the right, a histogram of the error shows the expected Gaussian behavior.*

2.2 Sensor Selection

As commented in the introduction, regarding the exteroceptive perception systems, ultrasonic sensors has been selected for reactive control and a laser (LIDAR) in point cloud mode for the localization.

The Apolo simulator also offers a laser beacon sensor. The three are studied and calibrated, since all of them can be useful later on, if required.

For the calibration, we will use a square room ($8 \times 8\, m^2$) with the robot in the middle (see next Figure 9).

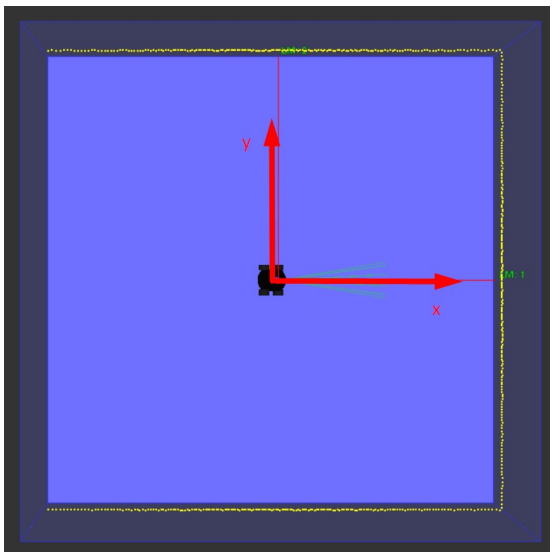

Figure 9: *Square room in Apolo for the exteroceptive calibration. Marvin is placed in the center facing forward along the x-axis.*

2.2.1 Calibration for Uncertainty Assessment

In the **LIDAR with point cloud mode**, we obtain the distances from the robot to all points on the wall by sweeping an angle of 270 degrees, which generates a total of 539 measurements, what means a resolution of $\sim$ 2 measurement/degree. It is essential to calculate the variance in the distance measurements. Due to the symmetry of the room, the 270 measurement in the output vector of the point cloud represents the desired distance. By repeating this measurement 1000 times, the resulting variance is

$$R = \left[9.62015 \cdot 10^{-6}\right] \; m^2$$

where R is the variance and covariance matrix of measurement noise, since we only have one sensor, the dimension of the matrix is 1×1.

The **ultrasonic sensor** does not present noise in the distance measurement. We have checked this with Apolo placing the robot two meters from the wall and tanking 1000 measurements (Figure 10.a). In the plot (see Figure 10.b) it is shown how the same distance is obtained for all the measures. For this reason we will just use the ultrasonic sensor for the reactive control.

Finally, the **laser beacon sensor** is calibrated as it may be necessary in subsequent tests or simulations. The same square room as before is used, with one landmark placed at the same height of the laser and in front of it, in the x direction. In Figure 10.a, the landmarks on the walls and the red laser beam are visible. We take 1000 measures and compute the variance in the angle and in the distance, obtaining the next matrix of measurement noise:

$$R = \begin{bmatrix} 3.4665\, m^2 & 0 \\ 0 & 1.1771\, rad^2 \end{bmatrix} \cdot 10^{-4} \tag{4}$$

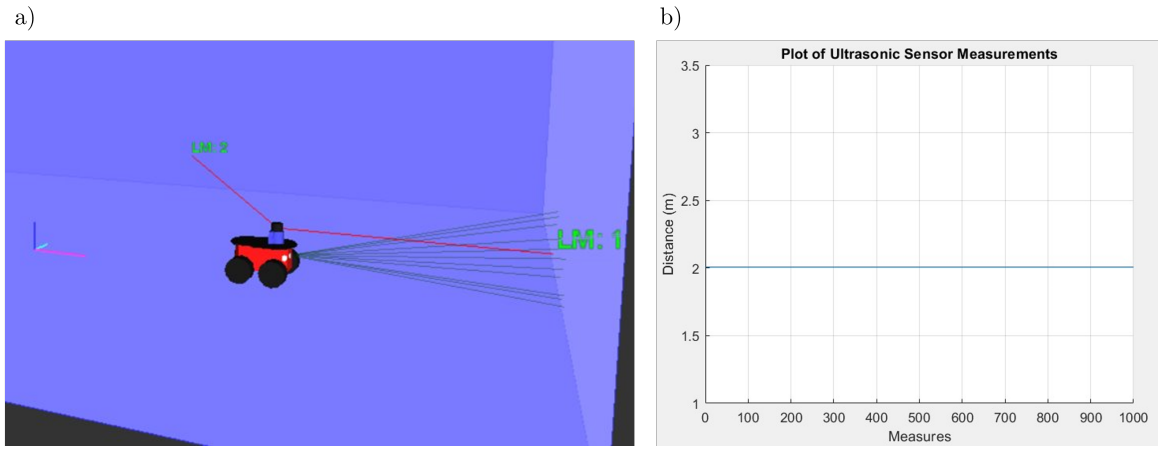

Figure 10: *Ultrasonic sensor test. In (a) Marvin is placed two meters from the wall while in (b) each of the measures is plot and we can see how the distance never changes.*

If we plot the probability of the correct measurement in distance and angle we see that both follow a normal distribution with variances of eq.(4) and centered on the mean which is the expected measurement (refer to Figure 11). We get 0 radians for the angle (as the robot is measuring in front) and $3.8\,m$ for the distance, because the sensor is $0.1\,m$ ahead and the beacon must be in front of the wall to be measured (another $0.1\,m$ closer to the robot) giving a total of $3.8\,m$.

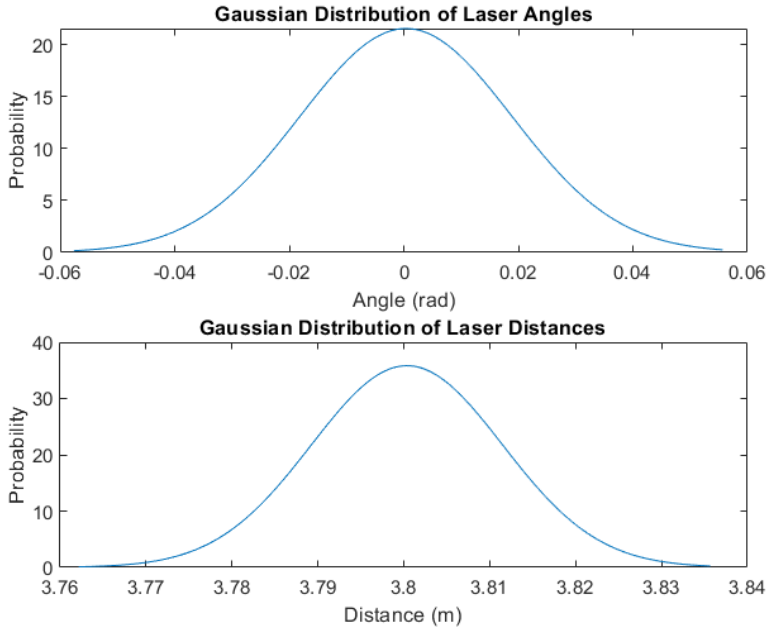

Figure 11: *Gaussian distribution of the measurement in distance and angle.*

CHAPTER 3

Localization and Estimation

3.1 Details of the Programmed Localization Algorithm

Given the highly structured environment, a localization EKF based on the detection of walls and the distance to them was chosen. The main advantage of this approach is that it dispenses with the use of landmarks along the map. However, as it will be seen throughout this section, the designed algorithm was not able to maintain a correct estimation of the robot's position. For this reason, another EKF based on landmarks was implemented.

3.2 Localization algorithm based on the distance to walls

A quick explanation of the algorithm is given. More details in subsequent sections:

- The LiDAR generates a sampling of the nearby walls.
- The points are classified by proximity to group them into sets.
- Then the RANSAC algorithm is used to analyze the point sets and detect the most probable straight lines. Distances are calculated to walls and compared with the estimated distances.
- The matching process consist in selecting those distances that best match with the expected distances.

3.2.1 Clustering

Before analyzing the points cloud, a filter is applied in order to remove the points further than 19.9 meters, because they do not represent any wall.

Utilizing the linkage function, "clusterdata" constructs an agglomerative hierarchical tree based on the input data. This hierarchical tree elucidates the degrees of similarity among entities with related elements grouped on proximal branches. In this certain process, the square euclidean distance was performed to determinate the similarity degree. This hierarchical tree is then truncated at a predefined level, determined by the cutoff value. The resulting clusters consist of groups of entities that persist after this segmentation. The function "clusterdata" provides cluster indices for each entity within the initial data set, thus providing a structured representation of the clusters formed.

The result of applying this process can be seen in the (Figure 12). Clusters with less than 10 points are removed in order to obtain only the most representative groups.

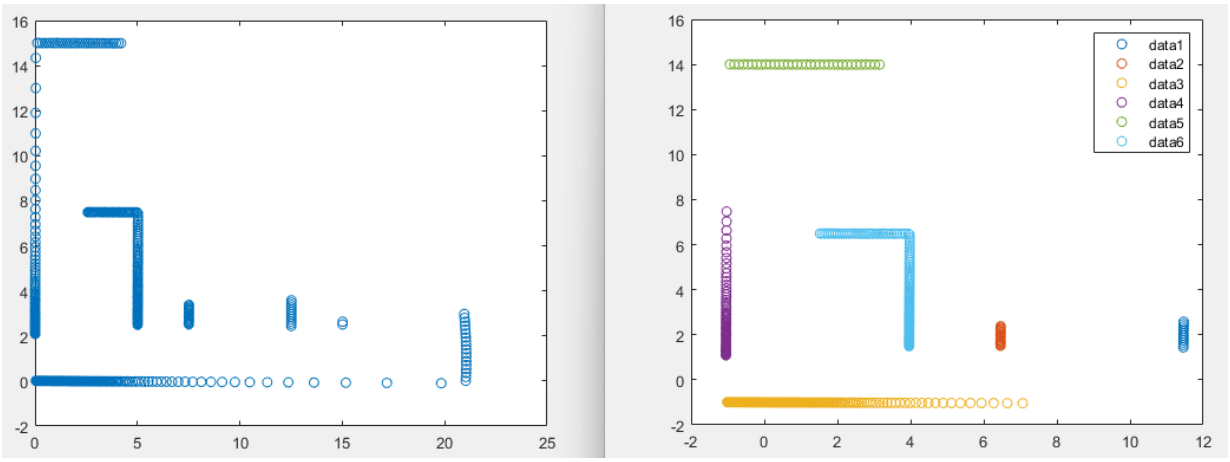

Figure 12: *Clustering Process. Blue points represents LiDAR measurements. Colored points represents each cluster.*

3.2.2 RANSAC and Distance Computing

Once the clusters are defined, the RANSAC algorithm is used to get the straight lines which represents the walls. RANSAC randomly selects two points from each set. These samples are then used to estimate an initial model. This randomness is pivotal for the method's robustness, allowing it to explore different data subsets and potential models. With the estimated model in hand, RANSAC evaluates the entire data-set by assigning each point the label of "inlier" or "outlier" based on whether a point fits the model within a predefined threshold. The number of inliers is then counted. This process repeats several times, each time with a different set of random samples. The model that yields the highest number of inliers across these iterations is chosen as the final model. In the Figure 13, is shown the result of applying RANSAC to to each cluster separately.

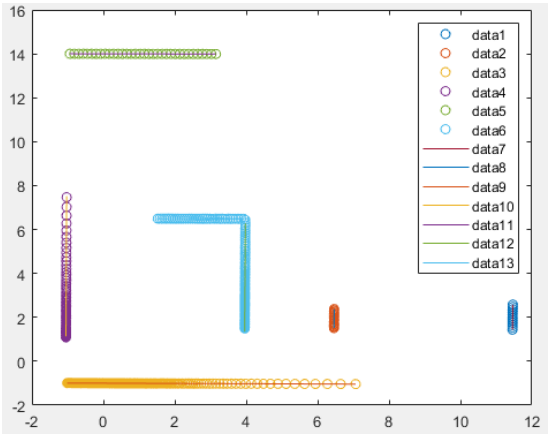

Figure 13: *RANSAC algorithm applied to the clustering process output*

The two extreme inliers points of the straight line are used to obtain the general equation of each of the infinite lines. Subsequently, the shortest distance of the infinite line is computed. In addition, the intersection point on the infinite line is determined. This point is used to determine whether the line is with respect to the robot, above, below, to the right or to the left.

This step allows to refine the matching process with the robot's simulations. Thus, if the intersection point is on the right, only the actual distance is compared with the estimated distances to walls that are at the right side of the robot seen from the map view. This filter avoids incorrect associations due to map symmetries.

This simple filtering method is feasible because of the map structure, since except for 1 wall, the rest are vertical or horizontal. So it is enough to know the "x" and "y" coordinates of the robot to check the position of the walls regarding the robot. There is no need to take into account its orientation.

3.2.3 Matching Process

The matching process involves two main steps: estimating distances to walls and generating an error matrix.

The estimation utilizes a wall map containing geometric extreme points. In this process, infinities lines and minimum distances from the robot to the walls are calculated based on the map. With 32 walls in the map, 32 distance estimates are generated. A Nx32 matrix is then formed, representing the absolute error between real (N infinite lines detected) and estimated distances.

Error measurements are computed only for distances corresponding to the position of the wall with respect to the robot, reducing incorrect associations. Non-matching distances have a value of 99. Effectively eliminating them during matching.

Once the error matrix is created, indices with the smallest values for each measured distance are obtained in order to reach the non-absolute errors of the distances. The rows in the error matrix represent sensed distances, and the columns represent the robot distance to each of the 32 walls in the map.

$$ErrorMatrix = \begin{bmatrix} 99 & 99 & 99 & 99 & 99 & 99 & ... \\ 99 & 99 & 99 & 99 & 99 & 99 & ... \\ 99 & 99 & 99 & 99 & 99 & 99 & ... \\ 99 & 99 & 99 & 99 & 99 & 99 & ... \\ 7.5 & 0.000117 & 0.000117 & 7.50 & 2.50 & 9.99 & ... \\ 99 & 99 & 99 & 99 & 99 & 99 & ... \\ 5.2837e-05 & 7.5 & 7.5 & 5.2837e-05 & 5 & 17.5 & ... \end{bmatrix} meters$$

Please note that some errors are repeated for the same detection (row), this is due to the fact that many walls of the map are contiguous to others and by working with infinite straight lines, the distance to both lines is the same. This particularity does not suppose any problem, because in the end, they are the same line.

3.2.4 Jacobians

The calculation of the Jacobian consists of taking the general equation of a straight line and deriving it partially with respect to $[x, y, \theta]$. The equation of an straight line is chosen, because the distance to infinite lines is being calculated.

$$0 = Ax + By + C \tag{5}$$

Where A, B and C are the parameters of the straight lines gotten from the map "x" and "y" variables are the estimated pose of the robot. The aspect of the $h(\hat{x}_{k+1|k})$ matrix and Jacobian H_{k+1} are the following:

$$h(\hat{x}_{k+1|k}) = \begin{bmatrix} \frac{Ax+By+C}{\sqrt{A^2+B^2}} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}; H_{k+1} = \begin{bmatrix} \frac{A_1}{\sqrt{A_1^2+B_1^2}} & \frac{B_1}{\sqrt{A_1^2+B_1^2}} & 0 \\ \frac{A_2}{\sqrt{A_2^2+B_2^2}} & \frac{B_2}{\sqrt{A_2^2+B_2^2}} & 0 \\ \frac{A_3}{\sqrt{A_3^2+B_3^2}} & \frac{B_3}{\sqrt{A_3^2+B_3^2}} & 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

Please, note that the size of matrices varies dynamically with the number of detected walls. $h(\hat{x}_{k+1|k})$ and H_{k+1} are dimension Nx1 and Nx3 respectively. N is the number of detections.

3.2.5 EKF implementation

A regular EKF coupled with the Mahalanobis distance has been applied as a method to select the best matches. The application of the Mahalanobis distance is as follows:

$$y_{k+1} = z_{k+1} - h_{k+1}(\hat{x}_{k+1|k}) \text{(Innovation)} \tag{6}$$
$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1} \text{(Innovation covariance)} \tag{7}$$
$$D_{k+1} = y_{k+1}^T S_{k+1}^{-1} y_{k+1} \text{(Mahalanobis Distance)} \tag{8}$$
$$D_{k+1} \leq \chi^2_{1-\alpha,m} \text{(Chi-squared test)} \tag{9}$$

On one hand it is obtained the innovation (6). On the other hand, the innovation covariance is obtained as equation (7). The Mahalanobis distance is calculated as the matrix product of equation (8). The result of these matrix operations is known as the squared Mahalanobis distance. Using the chi-squared table (9) , based on the one degree of innovation (error of distance to lines) and a cumulative probability of 0.95, the threshold used is 0.0039.

3.2.6 Performance and results

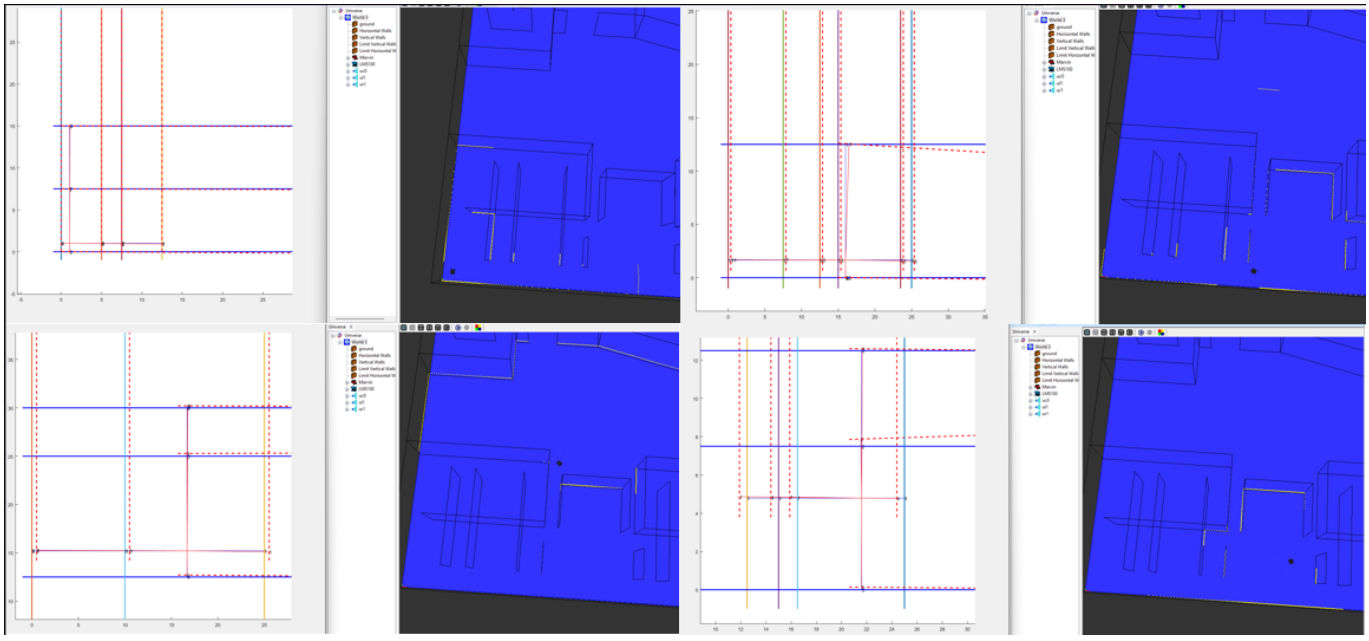

Figure 14: *The plane walls are shown as continuous lines and the detected ones as dashed. Distances to straight lines obtained by RANSAC are shown as red, while distances to plane walls are blue. The numbers indicate the ID of the distance to the wall and the point where it ends.*

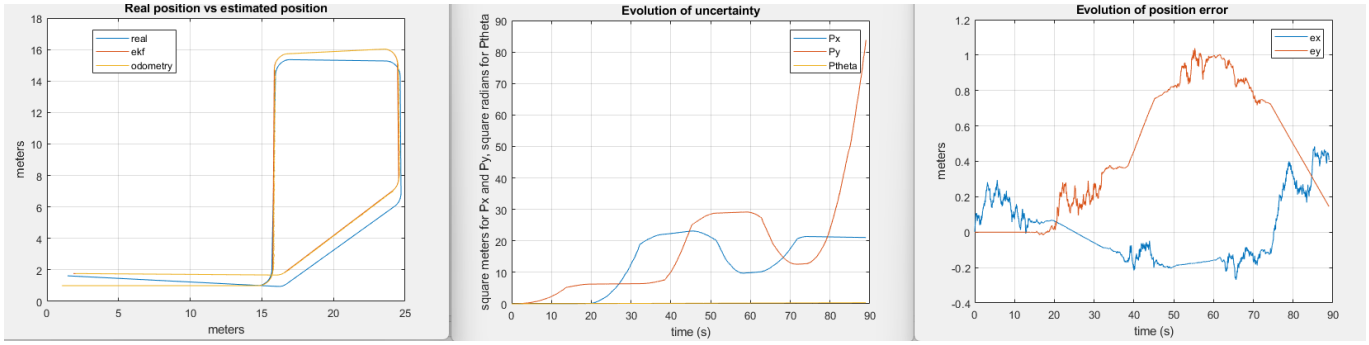

Figure 15: *Results of test with Mahalanobis distance* $= 0.0039$, $R = 0.01$.

As the robot advance over the route pre-programmed, the estimated walls and their distances are displaced because of the error estimation. The designed filter is unable to correct the estimation of the robot. In the Figure 15, it can be seen that Mahalanobis distance does not introduce any of the measurements. Therefore, a softer criterion is used, 0.1 instead of 0.0039.

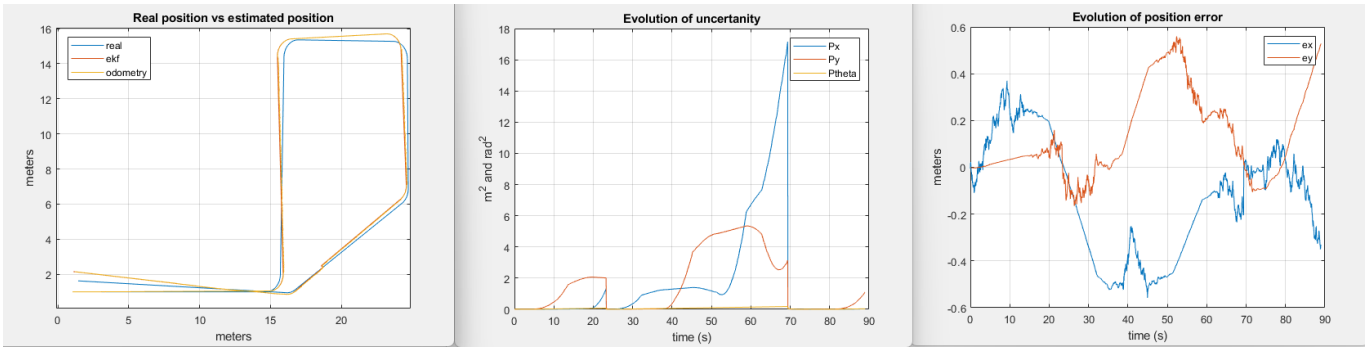

Figure 16: *Results of test with Mahalanobis distance* $= 0.1$, $R = 0.5$.

Despite increasing the variance of the exteroceptive measurements to 0.5 m^2 and making the Mahalanobis filter less aggressive, the performance of the method has not improved too much. Figure 16 shows the results. It can be seen that the Mahalanobis filter lets more measurements through, but throughout the test the algorithm diverges again. So it is tested to increase the variance up to 1 and threshold for the Mahalanobis distance up to 1.

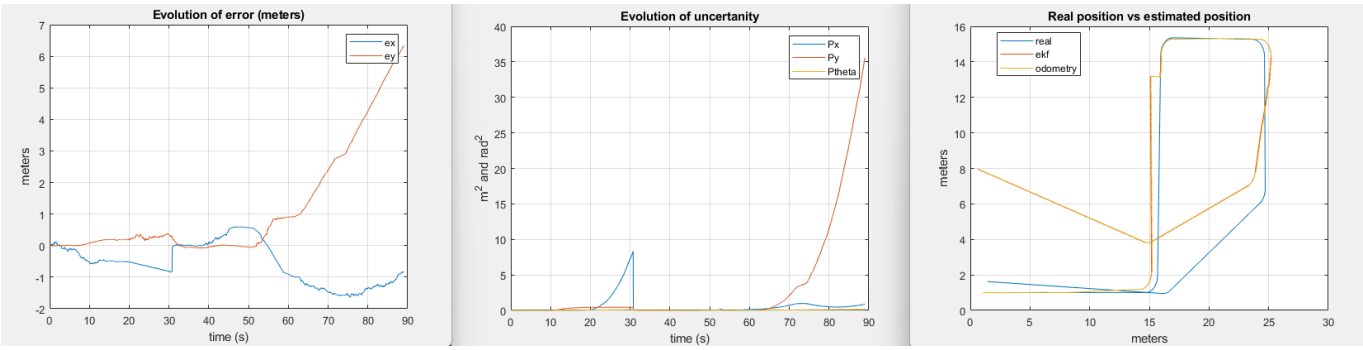

Figure 17: *Results of test with Mahalanobis distance* $= 1$, $R = 1$.

Figure 17, the filter allows more exteroceptives measurements, but at the end of the test, the EKF diverges rapidly. The next test sets the R variance to 5 m^2 and the threshold to 2.

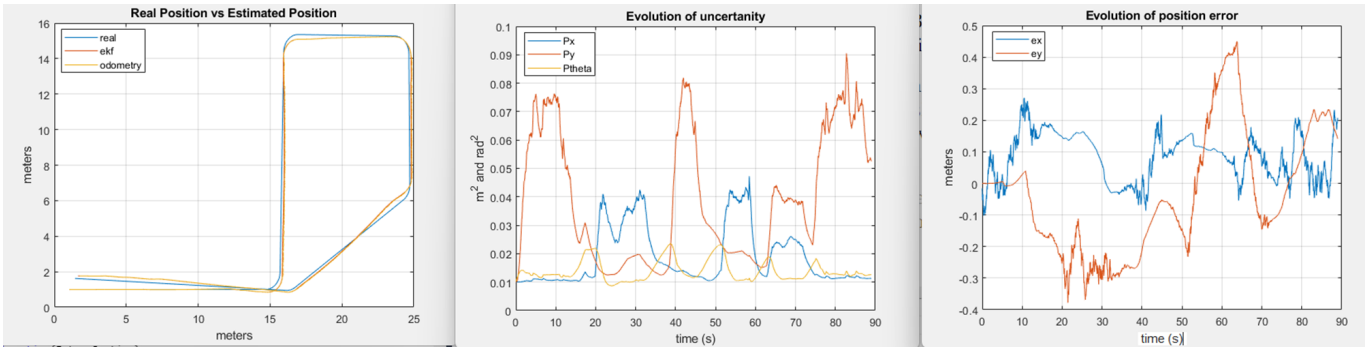

Figure 18: *Results of test with Mahalanobis distance* $= 2$, $R = 5$.

Figure 18, this test apparently shows a good performance compared to the others, but if repeated several more tests with the same configuration, it is clear that this particular test was extremely fortunate.

Figure 19, to analyze the perform along the trajectory, turns can help to know where the robot is at any time. The first turn happens from time 15 secs to 20 secs, the second one from time 32 to

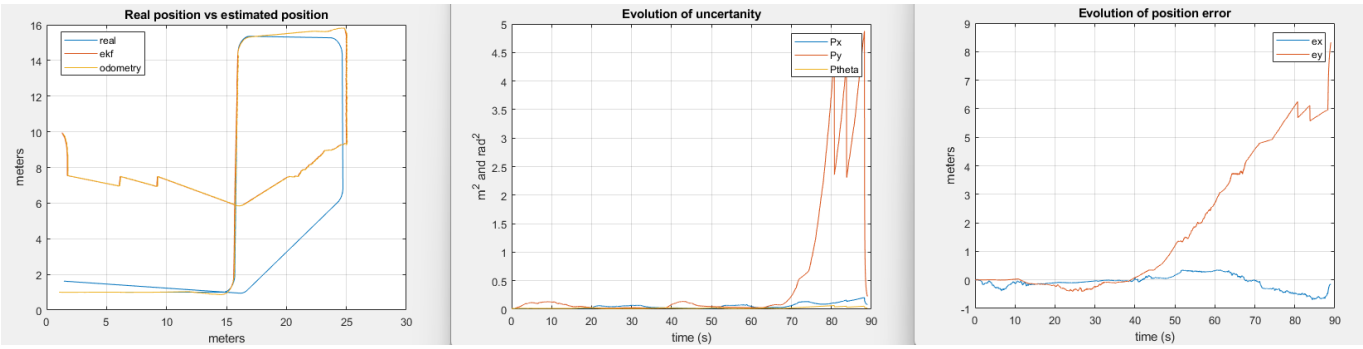

Figure 19: *Results of test with Mahalanobis distance* $= 2$, $R = 5$.

38 secs approximately, the third one from time 47 to 52 secs, the fourth one from time 58 to 63 secs and the last one from time 72 to 75 secs.

On one hand, it seems that one of the problems is that the Mahalanobis filter must be poorly implemented, as it should allow to introduce measurements as the uncertainty grows. In addition, the covariance matrix R must be really big to have a acceptable uncertainty, otherwise the algorithm does not take into account any exteroceptive measurement. When it comes to performance, it was proved that it is much better to have little acceptance of exterioceptive measurements than to accept many.

Regarding the computing of the Jacobian, it does not seem to be right. When the EKF allows more measurements to pass through, the algorithm tends to slightly converge to the real position, except for situations like Figure 19. In this case it looks like there was a mistaken association which provokes the divergence or the filter is unable to correct the odometry error. After the third curve, the algorithm usually begins to diverge despite the inclusion of exteroceptive measurements. It can be seen easily in the Figure 19, where the algorithm starts to diverge at time 40 secs, but it stops including measurements at time 68 secs. Examining the construction of the Jacobian with respect to the "y" derivative, it was observed that the parameter B is almost always zero because the vertical lines have B=0, causing that this value is usually zero. This detail may be causing the malfunction.

When the distances are plotted, everything seems to be consistent. The associations are correct, the robot detects correctly the straight lines on the map and the calculated distances are correct. Nevertheless, there are some situations in which it must struggle with something, otherwise we cannot explain what happens in situations like in Figure 17 and Figure 19.

In conclusion, the implementation must have some bugs when it comes to the matching process, Jacobians and their acceptance with the filter. Should be take into account changing the infinite lines representation. One option proposed in other works and by our teacher is the vector form.

3.3 EKF based on landmarks

Several landmarks where placed along the map, 35 exactly, as can be seen in Figure 20. The EKF process uses the landmarks ID detected to compute the estimated distance and angle of the landmarks respect with the estimated robot pose ($\hat{z}$).

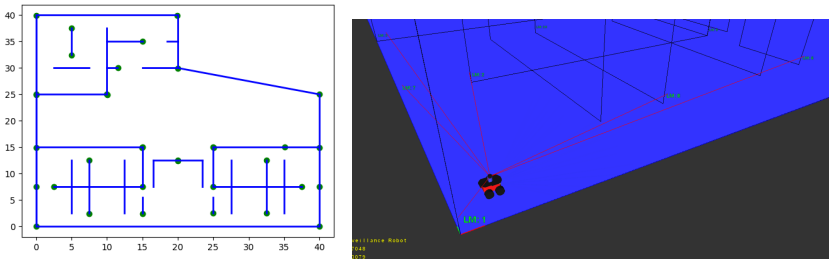

Figure 20: *(a): Map (blue lines) with landmarks (green points) ; (b): Example of landmarks detection*

As it uses the distance and angle, $\hat{z} = h(\hat{x}_{k+1|k}, L_i)$ matrix dimensional must be $2N$x2, where N represents the number of landmarks detected. The same happens with the Jacobian matrix H with dimensional $2N$x3. It is assumed the robot never confuse landmark IDs, therefore this parameter is not included in the matrices.

The distance is computed thorough the euclidean distance between two points. The angle of the landmark shown from the robot is calculated as the subtraction of the function "atan2" with the estimated angle of the robot. The quantity "atan2(y,x)" is the measure of the absolute angle between the axis x and a ray from the origin (robot pose estimation) to a point (x, y) (landmark). To obtain the correct angle of the reference point (relative angle), it is necessary to subtract the angle of the robot. The Jacobian matrix H is calculated as the Jacobian matrix of the function h with respect to the robot pose $[x, y, \theta]$.

$\mathbf{L}_i$ is a vector with landmark coordinates and its ID that it is already known by the algorithm (not reported by sensor). $\hat{x}_{k+1|k}$ represents the state vector from prediction step. z is a vector with measurements reported by sensor.

$$L_i = \begin{bmatrix} x_i \\ y_i \\ id \end{bmatrix}; \hat{x}_{k+1|k} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}; z = \begin{bmatrix} distance \\ \theta \\ id \end{bmatrix}$$

$$h(\hat{x}_{k+1|k}, L_i) = \begin{bmatrix} \sqrt{(x_i - \hat{x})^2 + (y_i - \hat{y})^2} \\ \text{atan2}(x_i - \hat{x}, y_i - \hat{y}) - \hat{\theta} \end{bmatrix}; H_{k+1} = \begin{bmatrix} -\frac{x_i - \hat{x}}{\sqrt{(x_i - \hat{x})^2 + (y_i - \hat{y})^2}} & -\frac{y_i - \hat{y}}{(x_i - \hat{x})^2 + (y_i - \hat{y})^2} & 0 \\ \frac{y_i - \hat{y}}{\sqrt{(x_i - \hat{x})^2 + (y_i - \hat{y})^2}} & -\frac{x_i - \hat{x}}{(x_i - \hat{x})^2 + (y_i - \hat{y})^2} & -1 \end{bmatrix}$$

Once H_k and $\hat{z}_{k+1}$ are computed, the update steps can be done:

$$y_{k+1} = z_{k+1} - \hat{z}_{k+1} \tag{10}$$
$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1} \tag{11}$$
$$K_{k+1} = P_{k+1|k} H_{k+1}^T (H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1})^{-1} \tag{12}$$
$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1} y_{k+1} \tag{13}$$
$$P_{k+1|k+1} = (I - K_{k+1} H_{k+1}) P_{k+1|k} \tag{14}$$

3.4 Experiments to Analyze the Filter's Parameter Influence

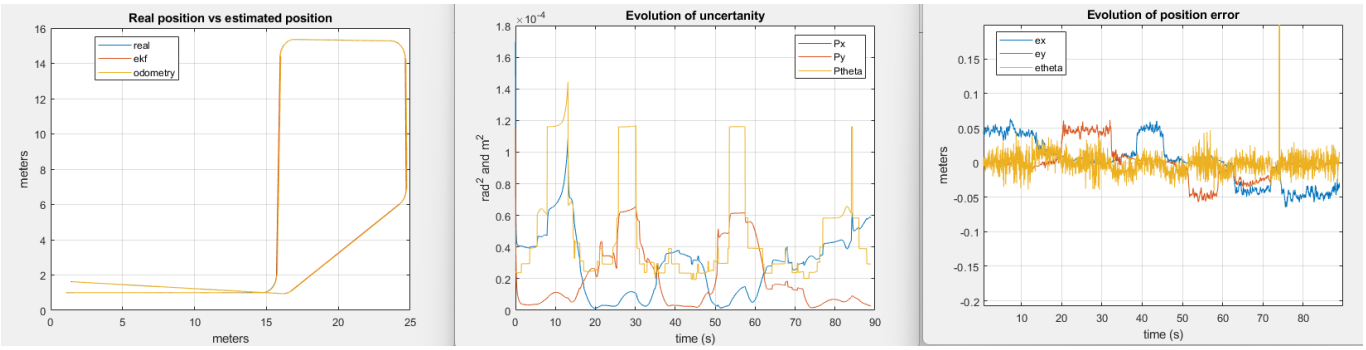

Figure 21: *Result of testing a programmed route with EKF based on landmarks; QLineal=1.3e-05; QGiro=0.013 — RDistancia=3.4665e-04; RAngulo=1.177e-04*

In order to compare the results with previous tests, the same trajectory will be followed and the variances of the measurements will be varied to see how they affect to the filter. In test of the Figure 21, it can be seen a really good performance. The error does not increase over 5 *cm* along

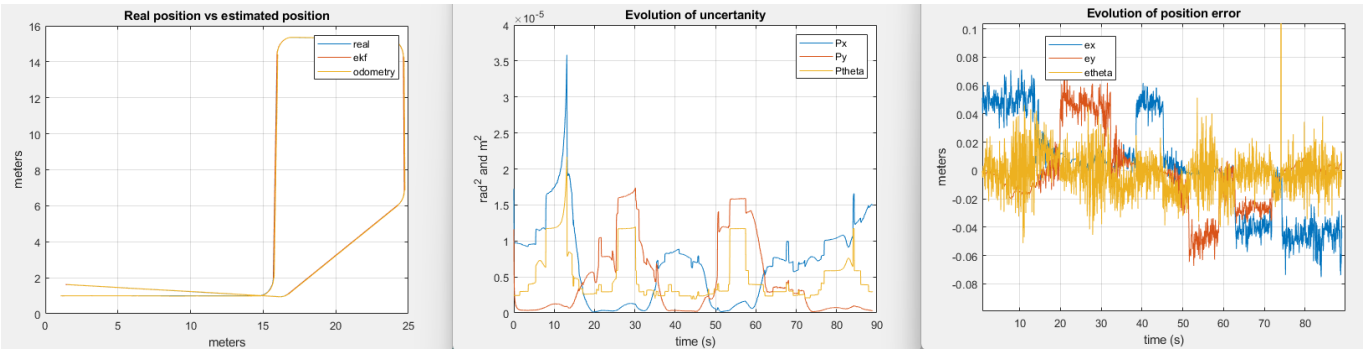

Figure 22: *Result of testing a programmed route with EKF based on landmarks;* $QLineal = 1.3e-05$*;* $QGiro = 0.013$*,* $RDistancia = 3.4665e-05$*;* $RAngulo = 1.177e-05$*.*

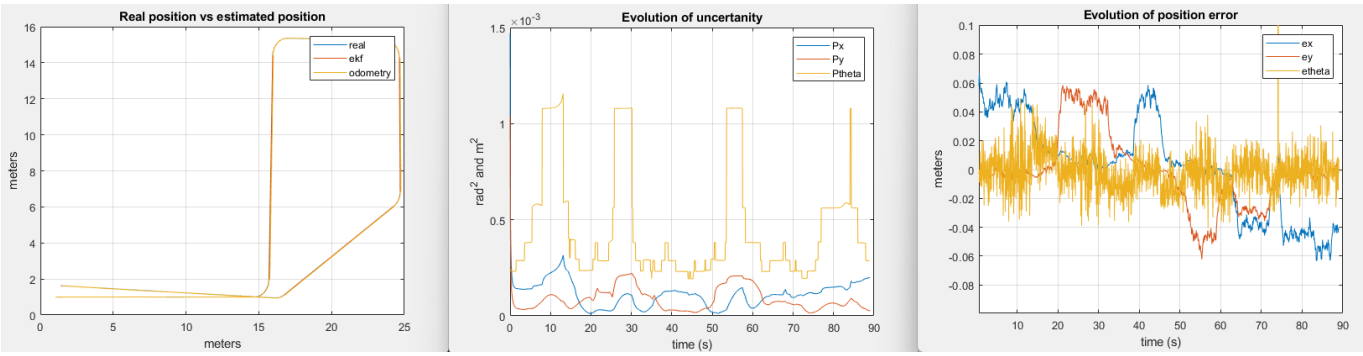

Figure 23: *Result of testing a programmed route with EKF based on landmarks;* $QLineal = 1.3e-05$*;* $QGiro = 0.013$*,* $RDistancia = 3.4665e-03$*;* $RAngulo = 1.177e-03$*.*

all the test, except for a sporadic error in the angle. Observing the evolution of uncertainty, it is highlighted how it changes when a turn occurs, which are the moments when its value changes rapidly.

Observing the tests performed, no major differences are found when varying the variances. It stands out when the variance of the exteroceptive measurements is larger, the adjustments made are smoother. On the contrary, it is observed that the smaller the variance is, the stronger the adjustments are. This increases the inconsistency of the algorithm due to sudden adjustments, which may affect the overall performance. It can be noticed as a noisy adjustment in Figure 22.

If, in the opposite case, the variance of the odometry is modified, the result is the same but with the modifications reversed. In other words, the larger the variance is, the stronger the corrections are. This effect makes a lot of sense. The filter gain balances the relevance of the corrections as a function (12) of the variance of odometry and exteroceptives. The more reliable the odometry, the less influence the exteroceptive measurements will have on the algorithm.

After testing several variances it was concluded that the fit with the variances obtained in the calibration tests gave the best performance. This indicates that the tests appear to be well done.

CHAPTER 4

Path Planning, Reactive Control and Obstacles Avoidance

This section explores the trajectories and velocity commands generation. The initial part focuses on the global and local path planners, detailing its integration from an existing library and the simulations performed to calibrate the algorithm. The next section delves into the controller and reactive control mechanisms.

4.1 Path Planning

4.1.1 Global Path Planning

Global path planning is in charge of obtaining a discrete trajectory between the destination point and the starting point through the generation of waypoints (reference points) which the avoidance of environment elements, already identified in a on-board map. It consist in the optimized Rapidly-exploring Random Trees (RRT*) that will achieve the optimal path with the required number of iterations.

The global path planner has been implemented based on "Lessons on Mobile Robot Localization and Kalman Filters: MathWorks File Exchange" [2] and "Biorobotics Lab UNAM 2024" [1], simultaneously. The first step consists in loading the map image where occupancy areas are distinguished from the free areas considering a binary discretization. A set of parameters required for the the path generation are defined: minimum Turning radius of the robot locomotion, minimum allowable distance to the walls, maximum number of iterations and maximum allowable connection distance. The algorithm uses the function *plannerRRTstart(Dubins State Space)* to explore the free areas through the propagation of ramified samples with connected nodes in random directions. The output of this module is a matrix containing the coordinates of the generated waypoints.

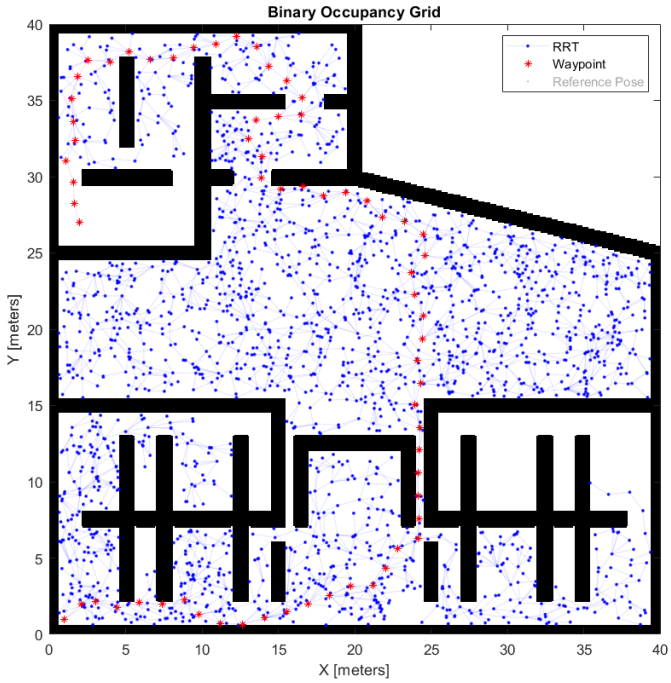

Figure 24: *Global planification with the waypoint computation using a RTT* planner.*

Figure 24 presents the waypoints for the final global path planner setting with initial pose at $(1, 1)$ and goal at $(2, 27)$. Note how the walls have been enlarged for safety. In our case, once the final goal is reached the random search stops and the path it is not optimized. The reason lies in the "bottleneck" that the map presents in the corridors leading to the central hall. This lengthens the planning time and it has been preferred to leave the smoothing of the trajectory to the local planner.

Experiments: Once the planner is implemented and adapted for the project requirements, some tests and simulations have been done in order to select the best values for the three parameters that controls the behavior of the planner.

They are the *MinTurningRadius*(mTR), the *ValidationDistance* (VD) and the *MaxConnectionDistance* (mCD).

For the experiments, three reference values have been taken for each of the parameters to be studied, 0.5, 0.5 and 1.5 for mTR, VS and mCD respectively. Thus, keeping two fixed at the reference value, a sweep was made at the third (single degree of freedom). The metrics chosen were run length, run time and smoothness (0 means straight lines) of the trajectory. All simulations were performed 10 times and the mean and variance of these simulations were obtained. This is necessary given the randomness of the RTT* algorithm. The following tables show the results obtained, where the values that minimize each parameter are highlighted in bold. The complete tables can be found in the annexes (at the end of Section 6).

MinTurningRadius	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.1	0.13473	0.0013915	**28.2172**	1.4024e-29	**3.0797**	0
0.2	0.12506	0.00095943	29.2282	1.0759	2.5573	0.28727
0.3	**0.1201**	0.00070726	29.6281	1.0358	2.3953	0.24254
0.4	0.12701	0.0008894	30.1627	1.6495	2.4223	0.18259
0.5	0.49192	0.55754	30.004	1.4157	2.7012	0.46279

ValidationDistance	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.1	**0.16497**	0.0010859	33.3004	0	**4.3693**	0
0.3	1.3794	0.78093	30.6796	3.5526	4.0009	0.07019
0.5	1.3095	0.72423	29.9642	2.9794	3.643	0.44362
0.6	1.425	0.67498	**29.865**	2.5244	3.672	0.37269
1.0	0.99173	0.69458	30.3281	3.951	3.3521	0.68184

MaxConnectionDistance	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.5	0.65991	0.019637	33.9094	5.6097e-29	**11.9207**	0
1.0	**0.43202**	0.033313	31.5038	2.9949	8.9891	7.4243
1.75	0.91441	0.58564	**30.3924**	2.7324	6.1236	13.1022
2.25	0.89318	0.49659	31.1046	3.8901	5.2323	12.2323
3.0	0.68747	0.47398	31.8831	6.1129	4.1226	12.1917

The path selected for these simulations is short in order to speed up the process. The initial pose is $[1, 1, 0]$ while the final pose is $[20, 12, 0]$.

4.1.2 Local Path Planning and Active Control

Local planning is responsible for obtaining a smooth and achievable trajectory accomplishing the robot kinematics restrictions. There exist a high dependency between the selection of the minimum Turning radius and the success of the reference trajectory chase. Depend on the wheel dimensions, a specific value is required. A minimum Turning radios of 0.5 m has been selected for this purpose. Smooth path points generation is done using the function *interpolate(initial and final points, number of intermediate points)*, belonging to the Matlab NavPath toolbox. This function generates intermediate points following a spline shape with rotations higher than the minimum Turning radius. The number of points is related to the simulation time step: implementation is based on keeping a constant forward velocity of 0.5 m/s; the robot shall go from two consecutive intermediate points at this forward velocity in the selected time step. This module extracts the two velocity commands to the control module: a constant forward velocity of 0.5 m/s and a variable angular velocity based on the required instantaneously rotation determined by the local intermediate points. This command has been called as active control.

Besides, a local planning module has been developed against a unique global planning thinking of the capability of re-planning in real-time in case of obstacle detection in an iterative process, as it is explaining in Section *Wall and Obstacles Avoidance*.

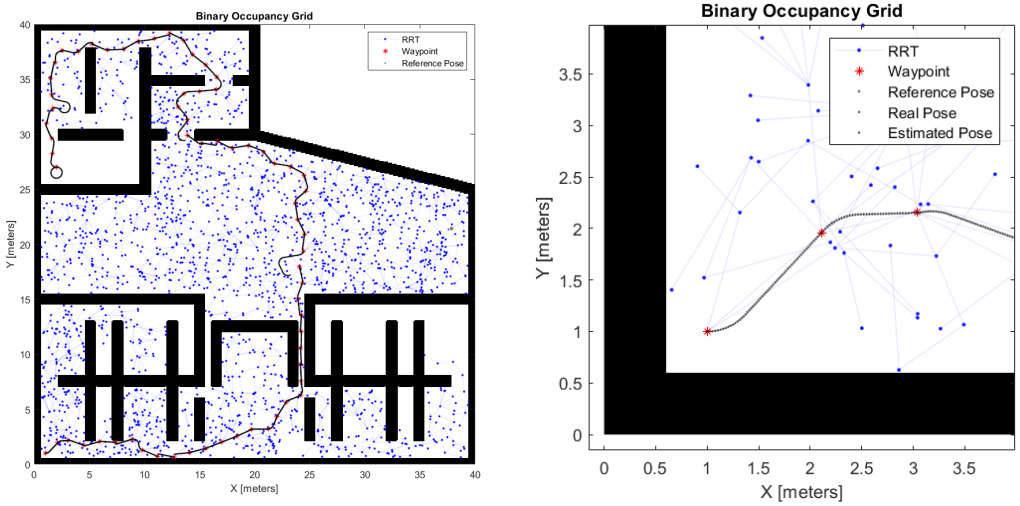

Figure 25: *(a) Reference trajectory points based on splines ; (b) Zoom-in.*

4.2 Reactive Control

The reactive control pursuits two objectives: to reduce the discrepancy between the reference position and the estimated position due to no-holonomic wheel inaccuracies and to introduce a collision avoidance maneuver in case of obstacles or closest elements are detected.

For the first purpose, a linear PI controller has been introduced where the current reference $P_{ref} = [x_0, y_0]$ and estimation positions are the controller inputs. To calculate their error, the formula of the distance between a point and a line has been used. For that, the equation of the line is obtained from the current estimation position $P_{est\,k} = [x_2, y_2]$ and the estimation from the previous time step $P_{est\,k-1} = [x_1, y_1]$:

$$(A \cdot x + B \cdot y + C = 0)\ ;\ A = (y_2 - y_1)\ ;\ B = (x_1 - x_2)\ ;\ C = (x_2 \cdot y_1 - x_1 \cdot y_2) \tag{15}$$

$$error = \frac{|A \cdot x_0 + B \cdot y_0 + C|}{\sqrt{A^2 + B^2}} \tag{16}$$

The distance error between the reference point and the equation of the estimated path line is computed in absolute values. To know the direction of this error and being able to compute the direction of the rotation (positive or negative), it is necessary to know the coordinates of the intersection. Using the dot product, it is possible to know the angle between the estimation path line and the line that joints the reference position with the intersection:

$$P_{closest} = \left[\frac{B \cdot (B \cdot x_0 - A \cdot y_0) - A \cdot C}{A^2 + B^2}, \frac{A \cdot (A \cdot y_0 - B \cdot x_0) - B \cdot C}{A^2 + B^2}\right] \tag{17}$$

When both lines are in the same direction, the dot product computes an angle between them closest to 0 deg and the error contribution would be very low and the controller does not command a rotation. Therefore, the controller is sensitive to the required direction of the reference, applying an accurate angular velocity command just when it is needed. Finally, the magnitude of the angular velocity is obtained from a PI controller that gives a proportional value as function of the distance to the reference and an integral part that allows to reduce the offset produced when the angle has stabilized around a value. Thus, when the lateral distance to the reference is high, the angular

velocity command will also be high. The gains are $K_P = 0.6$ and $K_I = 0.0025$. Figure 26 represents the position error along the path explained in Section 5 and the angular velocity proportional and integral parts contributions.

$$\omega_{robot} = K_P \cdot error(t) + K_I \cdot \int_0^t error(t)\, dt \tag{18}$$

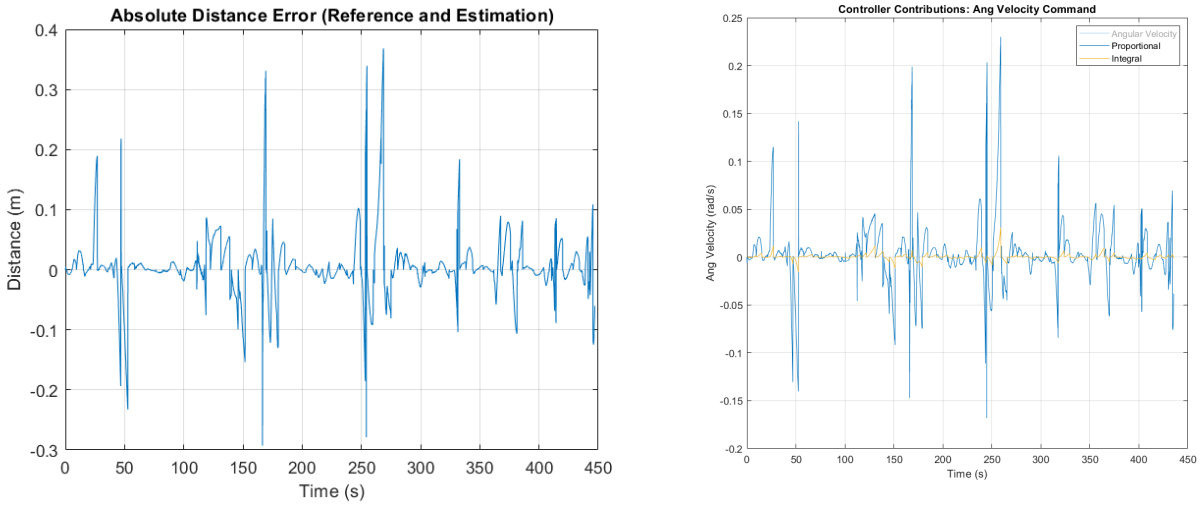

Figure 26: *(a) Distance error (reference VS estimated) ; (b) PI controller contributions.*

4.3 Walls and Obstacles Avoidance

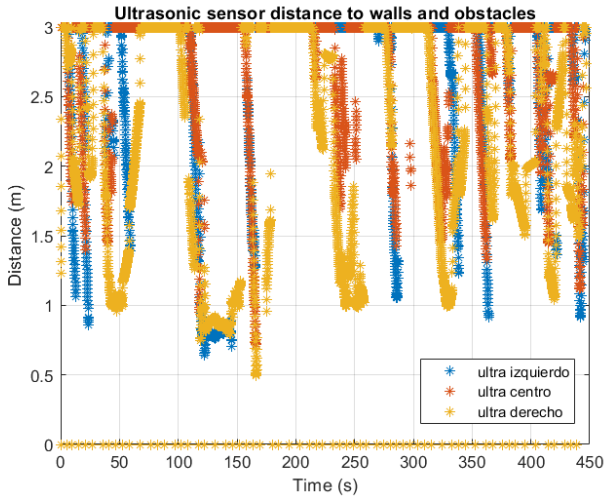

Figure 27: *Distance to walls and obstacles detected by ultrasonic sensors*

To detect the presence of dangerous or unexpected elements on the map, three ultrasonic sensors have been incorporated following the default sensor accommodation using in the subject lessons. There are a frontal sensor and two lateral sensors accommodated at 30 deg and -30 deg with respect to the longitudinal direction. Each sensor has a 10 deg field of view and a distance range of [0.5; 3] m. The function *apoloGetUltrasonicSensor()* directly provides the measurements. Measurements of 3 m are discarded as they are considered as saturated. A obstacle detection flag is raised when ultrasonic measurements are in such range, independently of the element: it might be a very close wall identified in the map or a new obstacle.

The robot behaves different as a function of which sensor has detected the proximal objects:

- **Lateral objects:** when lateral sensors detect close objects, an angular velocity command is applied to keep an exclusion distance, proportional to the distance to the object and in opposite lateral direction. This control is mainly thought for walls and narrow halls (Figure 28).

- **Frontal objects:** when the frontal sensor detects close objects, it is considered as very critical case and would suppose an imminent collision in case of dismiss the situation. For that, an avoidance maneuver is applied with a high value of constant angular velocity. The direction of this rotation is determined by the lateral sensors. They detect which lateral presents the farthest objects and apply this direction. Once, the robot has gone away from the obstacle, the obstacle detection flag is written to 0 and the local path planning sends a new reference path away from the object (Figures 29 and 30).

The values of these angular velocity commands are added to the values of the active control command and the previous reactive control command, already explained. In this way, a unique velocity command is executed in the wheels. It is important to highlight that the forward velocity is always 0.5 m/s.

4.3.1 Walls Avoidance Validation Experiments

This case occurs when the robot detects close walls, in one or both laterals. We want to overwhelm the situation with very narrow halls. The reference trajectory would generate points closer to one of the walls. However, the robot has the capability to generate a small angular velocity until centering in the middle of the hall and present the same error to both laterals, thanks to the proportional controller.

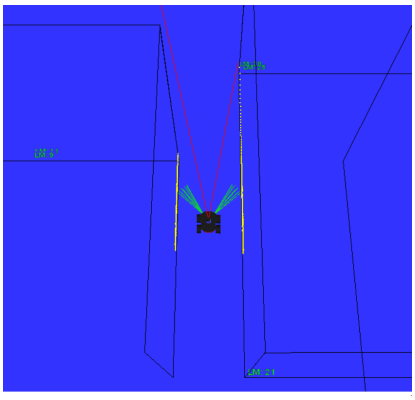

Figure 28: *Narrow hall recognition and walls avoidance.*

4.3.2 Obstacles Avoidance Validation Experiments

When an object not identified on the map is critically detected in front of the robot, the controller commands a 0.5 rad/s angular velocity in the favorable direction that eases to avoid the obstacle. Once the frontal sensor does not detect anymore the object, the local path planning generates a new trajectory from the current estimated position.

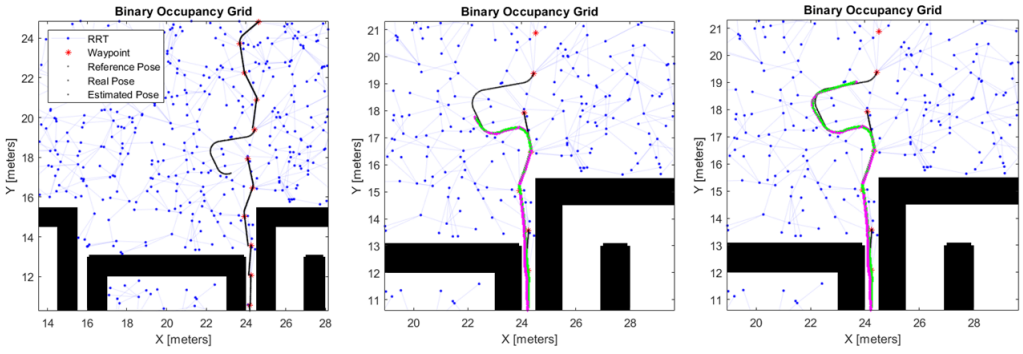

Figure 29: *Local re-planning and execution of obstacle avoidance trajectory.*

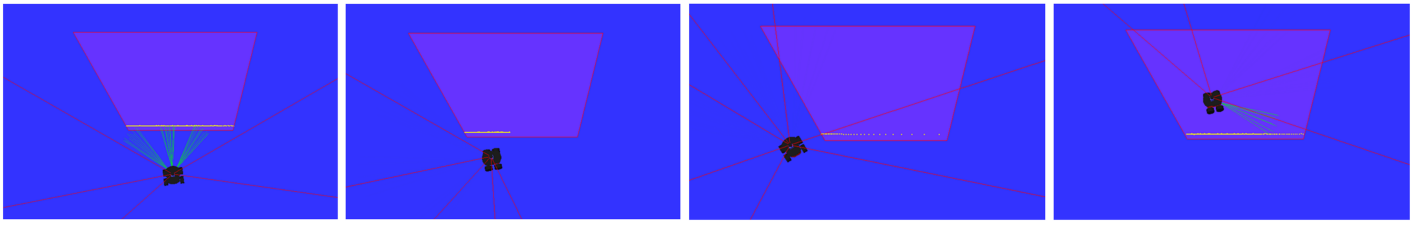

Figure 30: *Robot obstacle avoidance on the map*

CHAPTER 5

Conclusion and Future Work

5.1 Demonstrator Results and Accomplishments

To validate the Robotic Navigation and Control algorithm after the integration of the modules, a complete simulation study has been developed. Two worst-case simulations (as seen in Figure 31) contains the most-demanding constraints that an autonomous surveillance robot can find, with initial and final poses of Table 1:

	Starting Point (m)	Goal Point (m)
Simulation 1	[1; 1]	[2; 27]
Simulation 2	[2; 37]	[37; 10]

Table 1: *Initial and final poses for the two worst-case simulations.*

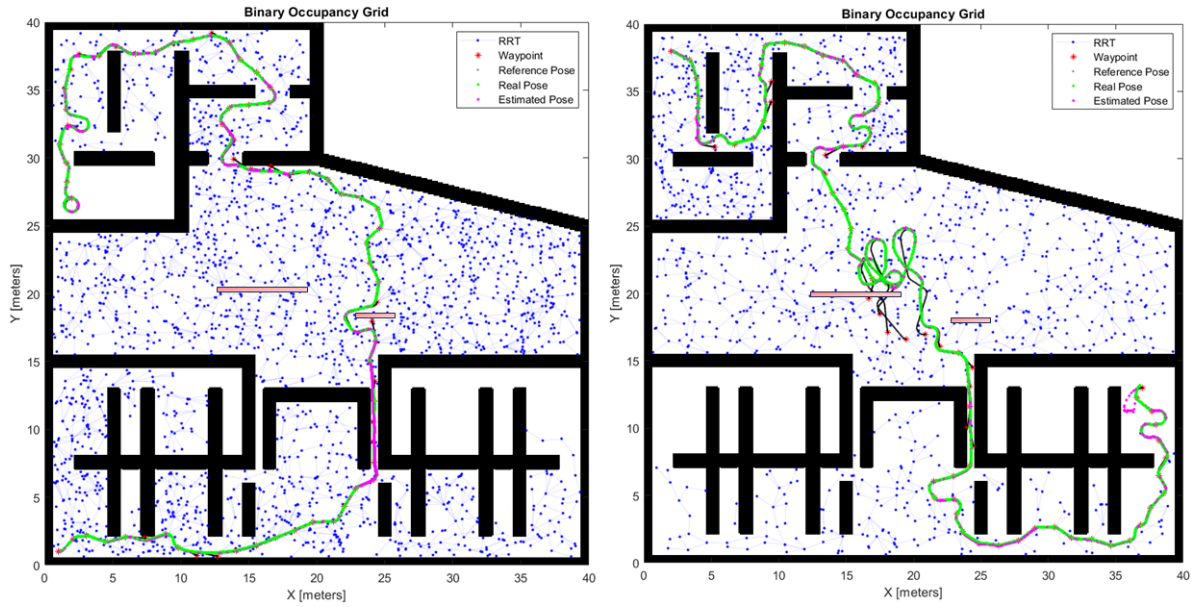

Figure 31: *Reference, estimated and real robot trajectories: (a) Simulation 1 ; (b) Simulation 2.*

- **Path planning capacity of always finding a trajectory:** thanks to the good performance of the global path planning based on RRT*, the algorithm is always able of providing a trajectory that connects the current position with the goal position with a pre-load map. It also incorporates the possibility of optimizing the trajectory but increasing the computational time.
- **Accurate location by the EKF application:** the robot is able to accurately locate thanks to the landmarks distributes along the map with a precision minor to 0.2 m between the estimated and actual position in the worst cases. When reactive control is used, the error in the estimated position and and its correction strength increase significantly. It can be seen clearly in the Figure 33 in the range of 100 to 170 seconds and 250 in simulation 1, when the error increases and gets more noisy. Nevertheless, the robot never loses an accurate estimation of its position and always recovers showing a robust performance in unexpected situations.
- **Presence of obstacles:** the robot successfully reacts to obstacles that are not identified by the global path planning. In Figure 31, it is able to detect and avoid the unexpected red wall in just one attend (Simulation 1) and in three attends (Simulation 2).

- **Presence of very narrow halls:** the robot detects the presence of close walls, it executes a command to separate from the walls. It the case of narrow halls, it successfully centers in the middle of the hall.
- **Presence of tiny rooms requiring 90 deg rotations:** the robot is able to success in trajectories that demand constrained rotations, as non-holonomic limitations have been considered during the algorithm design.
- **Presence of open spaces:** open space would be convenient for wall avoidance but would degrade the performances of the EKF. However, thanks to the strategically presence of landmarks and good implementation of the filter, the navigation performances are not significantly degraded.

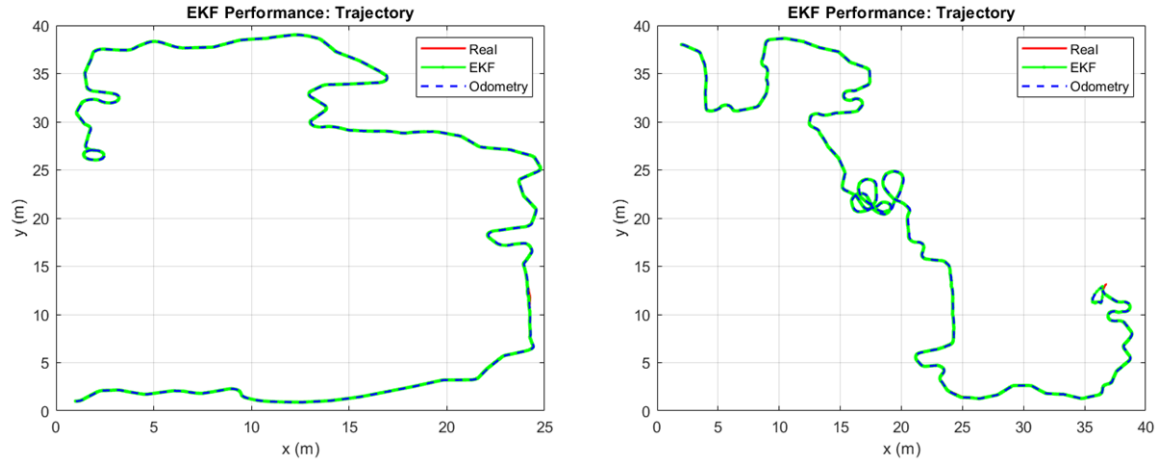

Figure 32: *EKF position error (real VS estimation): (a) Simulation 1 ; (b) Simulation 2.*

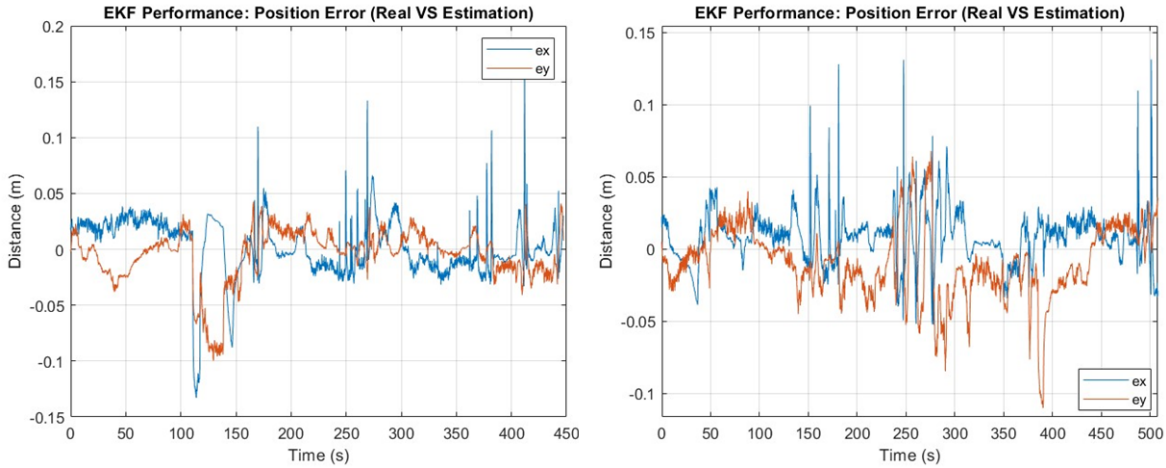

Figure 33: *EKF position error (real VS estimation): (a) Simulation 1 ; (b) Simulation 2.*

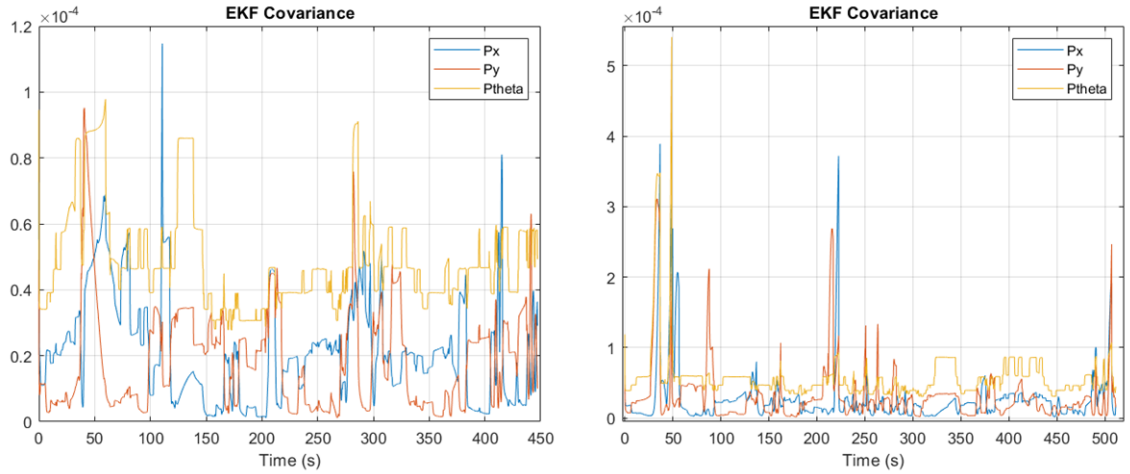

Figure 34: *EKF position error (real VS estimation): (a) Simulation 1 ; (b) Simulation 2.*

5.2 Discussion of Limitations and Suggestions for Future Improvements

During this project, an attempt has been made to build a robot pose estimation algorithm based on the distance to the straight lines. However, the performance of this algorithm was not good enough to implement it. Nevertheless, it remains for future lines to improve the algorithm.

The global path planning implements a RTT* algorithm with very good performance. But due to the challenging environment it has been proven that some times the obtained path it is not optimal, and sometimes even presented loops. It is believed that an implementation of trajectory smoothing would allow the elimination of such loops while optimizing the trajectory. On the other hand, in the implementation carried out, local localization allows to partially solve this shortcoming (by discretizing and interpolating). However, smoothing would ensure the optimality of the trajectory, which cannot be guaranteed now.

As far as reactive control is concerned, it usually performs very well. However, when it encounters obstacles that are more complex than a wall, such as something concave in shape, the control fails. An interesting implementation would be to give it the ability to reverse out of the concave-shaped obstacle.

CHAPTER 6

Role Distribution

For the distribution of each of the tasks contemplated in the guide for the preparation of the project report, the following table 2 has been created, where each member of the group is a letter from A to C, according to the table of team members (see Table 3).

TASKS	A	B	C
Spokesperson		X	
Sensor Calibration	X		X
Localization and EKF		X	
Global Planning			X
Local Planning and Reactive Control	X		
General Control	X	X	
Demonstrative Video	X		
Writing and documentation	X	X	X

Table 2: *Task distribution. Letters A to C represent each member whose details can be seen in table 3.*

PARTICIPANTS	NAME	REGISTRATION NUMBER
A	Alberto Ibernón Jiménez	23079
B	David Redondo Quintero	23147
C	Josep Mª Barberá Civera	17048

Table 3: *Participants and registration numbers*

The distribution in the project report writing is detailed below (Table 4).

PROJECT REPORT WRITING	A	B	C
Introduction	X		X
Sensor Calibration			X
Localization and EKF		X	
Global Planning			X
Local Planning and Reactive Control	X		
Conclusion and Future Work	X	X	X
Role Distribution			X

Table 4: *Project report documentation distribution.*

References

[1] Biorobotics Laboratory UNAM. *Lessons on Mobile Robot Localization and Kalman Filters*. 2024. URL: https://github.com/RobotJustina/MRS_EKF_MatLab/releases/tag/v1.0.1.

[2] I. The MathWorks. *Create an optimal RRT Path Planner (RRT*)*. 2023. URL: https://es.mathworks.com/help/nav/ref/plannerrrtstar.html.

Annexes

Software Repository

All the code developed or used for this project can be consulted in the next GitHub repository: https://github.com/albertoibernon/Autonomous_Surveillance_Robot

Path Planning Experiments

The following tables give all the results of the experiments carried out for the selection of the operating parameters of the planner used.

MinTurningRadius	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.1	0.13473	0.0013915	**28.2172**	1.4024e-29	**3.0797**	0
0.2	0.12506	0.00095943	29.2282	1.0759	2.5573	0.28727
0.3	**0.1201**	0.00070726	29.6281	1.0358	2.3953	0.24254
0.4	0.12701	0.0008894	30.1627	1.6495	2.4223	0.18259
0.5	0.49192	0.55754	30.004	1.4157	2.7012	0.46279
0.6	0.47707	0.46432	30.2402	1.4594	2.9022	0.5898
0.7	0.43108	0.40991	30.596	2.0183	2.7493	0.64658
0.8	0.39199	0.36889	30.445	1.9245	2.6522	0.63157
0.9	0.37825	0.32899	30.22	2.1177	2.5152	0.71238
1.0	0.38348	0.29658	30.457	2.4142	2.4299	0.70652

ValidationDistance	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.1	**0.16497**	0.0010859	33.3004	0	**4.3693**	0
0.2	1.0601	0.85933	31.3348	4.0668	4.093	0.080349
0.3	1.3794	0.78093	30.6796	3.5526	4.0009	0.07019
0.4	1.1396	0.75771	30.1129	3.6299	3.5996	0.5477
0.5	1.3095	0.72423	29.9642	2.9794	3.643	0.44362
0.6	1.425	0.67498	**29.865**	2.5244	3.672	0.37269
0.7	1.2442	0.77601	30.3558	3.6246	3.7716	0.37908
0.8	1.1413	0.75296	30.1129	3.584	3.5996	0.54077
0.9	1.0714	0.70825	30.4675	4.1984	3.4753	0.60492
1.0	0.99173	0.69458	30.3281	3.951	3.3521	0.68184

MaxConnectionDistance	$\bar{x}_{\text{time}}$	s^2_{time}	$\bar{x}_{\text{length}}$	s^2_{length}	$\bar{x}_{\text{smoothness}}$	$s^2_{\text{smoothness}}$
0.5	0.65991	0.019637	33.9094	5.6097e-29	**11.9207**	0
0.75	0.48207	0.042687	32.1297	3.3343	10.7612	1.4151
1.0	**0.43202**	0.033313	31.5038	2.9949	8.9891	7.4243
1.25	0.80538	0.45725	30.9567	3.148	7.9209	9.0322
1.5	1.0449	0.60089	30.6392	2.917	7.1	9.9389
1.75	0.91441	0.58564	**30.3924**	2.7324	6.1236	13.1022
2.0	0.97006	0.52047	30.6417	2.7147	5.6665	12.4748
2.25	0.89318	0.49659	31.1046	3.8901	5.2323	12.2323
2.5	0.81134	0.49499	31.6651	5.9943	4.8107	12.2962
2.75	0.74414	0.48612	32.0188	6.5259	4.4227	12.4222
3.0	0.68747	0.47398	31.8831	6.1129	4.1226	12.1917