EKF-BASED LOCALIZATION WITH LRF

Group 9 - Isabel Silva (73169), Renato Silva (73279), João Rosa (74149), Vítor Alveirinho (77003), José Mendes (82258), Maciej Przydatek (83022)

Abstract—The goal of this project is to explore both the theory behind the Extended Kalman Filter and the way it was used to localize a four-wheeled mobile-robot. This can be achieved by estimating in real-time the pose of the robot, while using a pre-acquired map through Laser Range Finder (LRF). The LRF is used to scan the environment, which is represented through line segments. Through a prediction step, the robot simulates its kinematic model to predict his current position. In order to minimize the difference between the matched lines from the global and local maps, a update step is implemented. It should be noted that every measurement has associated uncertainty that needs to be taken into account when performing each step of the Extended Kalman Filter. These uncertainties, or noise, are described by covariance matrices that play a very important role in the algorithm. Since we are dealing with an indoor structured environment, mainly composed by walls and straight-edged objects, the line segment representation of the maps was the chosen method to approach the problem.

Index Terms—Extended Kalman Filter, Laser Rangefinder, Localization, Mobile-Robot

1 Introduction

Localization is a fundamental problem in mobile robotics. In order for the robot to be autonomous, it needs to knows its own pose in the environment. Localizing the robot only with data provided by odometry is inaccurate since the measurement noise is constantly accumulating. Taking this into consideration, the robot will need to improve the information about its pose, which can be achieved by comparing the current environment scan with an already built global map, subsequently resulting in a better pose estimation. The environment is scanned with the aid of a Laser Rangefinder (LRF).

The global static map can be obtained through SLAM (Simultaneous Localization and Mapping), a process which plays a crucial role in the execution of the Extended Kalman Filter. This map is then subjected to a post-process that transforms all its boundaries (mainly walls) into line segments, a method which was chosen to the detriment of occupancy grids that divide the environment in cells. Since the environment is mainly composed by straight lines, the latter method would be computationally less efficient and therefore not the best option. To extract the line segments from the map, a method of Least Squares was chosen.

The Extended Kalman Filter provides optimal estimates for non-linear systems, relying on the input and output-noise covariance matrices of the process. As a matter of fact, to obtain estimations of the robot pose is to solve a non-linear problem. This localization problem can be split in two phases: the Prediction Step and the Update Step. The Prediction Step of the EKF is performed by simulating the kinematic model of the robot. The standard deviation of each of the wheels' angular speed of the robot is estimated as being proportional to the wheels' angular speed. In the Update Step of the EKF, the main objective is to correct the pose of the robot by subtracting the parameters of matching lines from local and global maps. For this we need to find the line segments from both the local and global maps, applying a matching procedure, where the most similar local and global line segments are paired if the difference is below a threshold.

The remainder of the paper is organized as follows. In Section 2, we give a brief introduction on what is ROS and how it was used in the context of our project. Section 3 describes how the implemented localization algorithm works. Experimental results of the algorithm are shown and discussed in Section 4. Finally, we state our conclusions in Section 5.

2 Using ROS

The Robot Operating System (ROS) is a set of software libraries and tools that help in building robot applications. Its modularity, extensibility and standardization allow system designers to cooperate flawlessly, while working separately. It can be used in many applications starting at industrial-type robotic manipulators, going through mobile platforms, such as wheeled autonomous vehicles or aerial multicopters, and ending at social and humanoid robots.

ROS consists of 4 main elements:

- **Roscore** - main program, which creates basic framework common for all modules;
- **Nodes** - modules, which perform specific tasks;
- **Topics** - allow nodes to communicate between each other;
- **Parameter Server** - set of values, which can be read, set or deleted by every node.

The work was conducted using Adept MobileRobots PIONEER 3-AT. In order to accomplish the task, several packages containing desired nodes were used, listed as follows:

- `rosaria` - written and maintained by Adept MobileRobots. Allows to establish connection between

ROS and PIONEER 3-AT, send commands to wheels and read robot's state;

- `sicktoolbox_wrapper` - performs communication with SICK LMS200 Laser Rangefinder, providing current environment scans;
- `gmapping` - used to implement SLAM (Simultaneous Localization and Mapping), and therefore to obtain the global static map needed for the implementation of the EKF algorithm;
- `map_server` - tool used to manipulate the map obtained by the `gmapping` package;
- `rosaria_client` and `teleop_twist_keyboard` - allow to easily control the robot's linear and angular velocities using the keyboard;
- `rviz` - used to visualize the map, the robot estimated position and orientation, path and laser scans;
- `tf` - manages multiple coordinate frames transitions during program execution.

2.1 Obtaining the Maps

In order to obtain the map, four packages were used: `rosaria`, `sicktoolbox_wrapper`, `gmapping` and `rviz`, to visualize the process. After setting up the nodes, the robot was driven over the available area. The `gmapping` package performed the SLAM process and, while generating the map, was sending it to the `map` topic. After closing the loop in the area, the finished map was saved using the `map_server map_saver` command-line tool. Then, it was converted to an ASCII text file with the values 0, representing free space , 100, representing occupied space and -1, for unknown area, and finally loaded to the main program for feature extraction.

2.2 Transform Configuration

In order to visualize the final process of the robot localizing itself on a map, a `rviz` node was used. To simultaneously show the map, robot localization based on odometry, localization based on EKF and laser scans on one referential frame, a coordinate transition was implemented. This should be done using `tf` node and its `tf.TransformBroadcaster()` method. Several few transform broadcasting nodes were created, to manage such dependencies as laser-to-robot, robot-to-odometry and odometry-to-map frame transitions. An enormous advantage of the `tf` tool is the fact that the transitions have to be specified only once and then the node manages all calculations on itself.

3 Localization Algorithm

The system of coordinates used is illustrated in Fig. 1. $x_G y_G z_G$ forms the global referential that is equal to the referential of the robot $x_R y_R$ in his initial position. $\mathbf{x}_p(k) = [x_r(k), y_r(k), \varphi_r(k)]^T$ denotes the robot's pose with respect to the global coordinates.

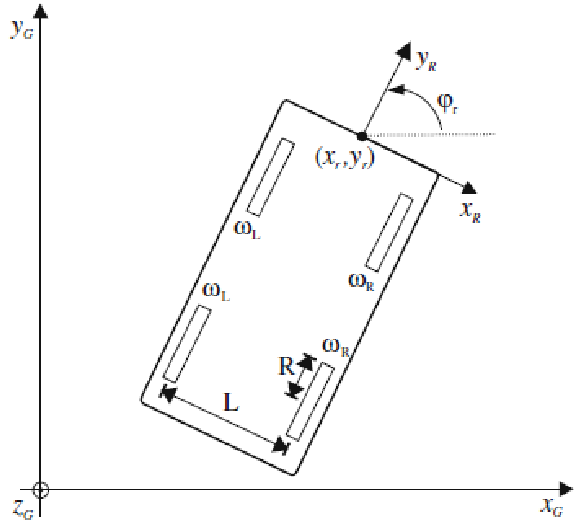

Fig. 1. Robot's pose according to the global coordinates.

3.1 Prediction Step

The robot's pose is predicted by simulating the discrete kinematic model of the robot

$$\begin{aligned} \mathbf{x}_p(k+1) &= \mathbf{f}(\mathbf{x}_p(k), \mathbf{u}(k)) : \\ x_r(k+1) &= x_r(k) + T\frac{R}{2}(\omega_R(k) + \omega_L(k))\cos(\varphi_r(k)), \\ y_r(k+1) &= y_r(k) + T\frac{R}{2}(\omega_R(k) + \omega_L(k))\sin(\varphi_r(k)), \\ \varphi_r(k+1) &= \varphi_r(k) + T\frac{R}{L}(\omega_R(k) - \omega_L(k)), \end{aligned} \tag{1}$$

where T is the sampling time, R denotes the radius of each wheel and L denotes the distance between the two front wheels. $\mathbf{u}(k) = [\omega_R(k), \omega_L(k)]^T$ is the input vector where $\omega_R(k)$ and $\omega_L(k)$ are measurements of the rotational speed of the left and right wheels, respectively, at the time kT.

Let $\omega_{R_c}(k)$ and $\omega_{L_c}(k)$ be the rotational speeds of the left and right wheels, which yields a correct estimation of the robot's pose $\mathbf{x}_p(k)$ (Eq.1). The error for the rotational speeds of the corresponding wheels $\omega_{R_n}(k)$ and $\omega_{L_n}(k)$ can then be defined as

$$\omega_R(k) = \omega_{R_c}(k) + \omega_{R_n}(k),\ \omega_L(k) = \omega_{L_c}(k) + \omega_{L_n}(k), \tag{2}$$

$$\mathbf{n}(k) = [\omega_{R_n}(k), \omega_{L_n}(k)]^T. \tag{3}$$

The error vector $\mathbf{n}(k)$ above captures the uncertainties of the odometry model and is assumed to be zero mean and Gaussian noise.

The covariance matrix of this vector is the input-noise covariance matrix $\mathbf{Q}(k)$ for the EKF, defined as

$$\mathbf{Q}(k) = \begin{bmatrix} \delta\omega_R^2(k) & 0 \\ 0 & \delta\omega_L^2(k) \end{bmatrix}. \tag{4}$$

The parameter δ in the covariance matrix above was estimated experimentally, as shown in Section 4.

For each time we compute the prediction step of the EKF, we first calculate the rotational speed of the wheels $\omega_R(k-$

1) and $\omega_L(k-1)$ using (Eq. 1), where $\mathbf{x}_p(k-1)$ and $\mathbf{x}_p(k)$ are given by odometry at times $(k-1)T$ and kT, respectively. After this, we can use the results to compute the prediction step.

The prediction step gives us the state prediction $\tilde{\mathbf{x}}_p(k)$ and the covariance matrix of the state prediction error $\tilde{\mathbf{P}}(k)$.

$$\tilde{\mathbf{x}}_p(k) = \mathbf{f}(\hat{\mathbf{x}}_p(k-1), \mathbf{u}(k-1)), \tag{5}$$
$$\tilde{\mathbf{P}}(k) = \mathbf{A}(k)\hat{\mathbf{P}}(k-1)\mathbf{A}^T(k) + \mathbf{W}(k)\mathbf{Q}(k-1)\mathbf{W}^T(k),$$

$$\mathbf{A}_{ij}(k) = \left.\frac{\partial \mathbf{f}_i}{\partial \hat{\mathbf{x}}_{pj}(k-1)}\right|_{(\hat{\mathbf{x}}_p(k-1),\mathbf{u}(k-1))},$$
$$\mathbf{W}_{ij}(k) = \left.\frac{\partial \mathbf{f}_i}{\partial \mathbf{n}_j(k-1)}\right|_{(\hat{\mathbf{x}}_p(k-1),\mathbf{u}(k-1))}, \tag{6}$$

where $\hat{\mathbf{x}}_p(k-1)$ denotes the state estimate at time instant $k-1$ based on all the measurements collected up at the time. $\hat{\mathbf{P}}(k-1)$ is the covariance matrix of the corresponding estimation error.

3.2 Obtaining Line Parameters

3.2.1 Clustering Points of the Global Map

In order to obtain the parameters for the lines of the global map, it is necessary to analyse the matrix obtained by SLAM and create a cluster of points for each line that is on the map. Using the Corner Harris Detection algorithm [2] we can detect all the corners of the map. We know that each corner belongs, at least, to one line. So, if we check the neighbourhood of the corner in all the directions, we can create a cluster of all the points that belong to one line. Each point is analysed by the distance from the previous point in the cluster to check if it is a good candidate to the line. The difference between the slope of the line composed by all the points that are already in the cluster and the slope of the line composed by the corner and the candidate point should be below a small threshold. If these conditions are true, the point is clustered.

3.2.2 Clustering Points of the Local Map

We need to know the lines that the robot is "seeing" in each cycle of the EKF in order to update the pose of the robot. For that, we use the data received from the LRF, which consists of a set of distances d between the laser and an obstacle, for a given set of angles $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. We transform each pair of distance and angle to a point (x, y) in the robot frame (Fig. 2) as in

$$x_i = d_i \sin(-\theta_i), \quad y_i = d_i \cos\theta_i. \tag{7}$$

It is important to notice that in y axis the angle θ is zero, on the positive x axis $\theta = -\frac{\pi}{2}$ and in the negative x axis $\theta = \frac{\pi}{2}$.

To store the points in clusters we need to analyse if the distance between two consecutive points is bellow a threshold $T_{d_{points}}$ and also if the difference between the angle of those two points and the angle of the line composed by the points that are already stored in the cluster is bellow a threshold $T_{a_{points}}$. If these two conditions are verified the point is clustered. If these conditions are not fulfilled

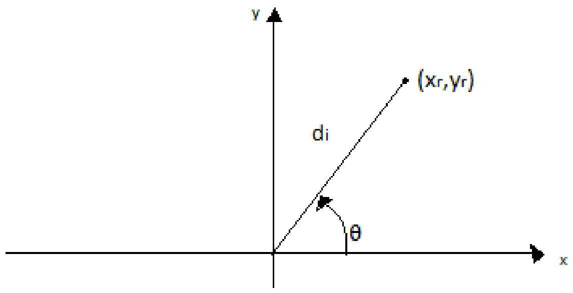

Fig. 2. Relation between the distances d_i, angle θ_i and the point (x_i, y_i), in the referential of the robot

it means that the point does not belong to the cluster and the cluster is finalized. Then a new cluster is created and we store this last point in it. When a cluster is finalized, if a number of points that belongs to it is above a threshold n_{min}, we create a line. If not, the cluster is ignored.

3.2.3 Line Parameters

At this point we already know clusters of points $(\mathbf{x}, \mathbf{y})$ that form lines in both global and local maps. Now we need to find the parameters of the line equation in the normal form (Fig. 3), for each cluster of points.

For the local map we want to obtain the parameters ψ_i and r_i, for each line i, in the robot referential

$$x_R \cos\psi_i + y_R \sin\psi_i = r_i, \tag{8}$$

and for the global map we want to obtain the parameters α_j and p_j, for each line j, in the global referential.

$$x_G \cos\alpha_j + y_G \sin\alpha_j = p_j. \tag{9}$$

To find this parameters we used the LSQ method.

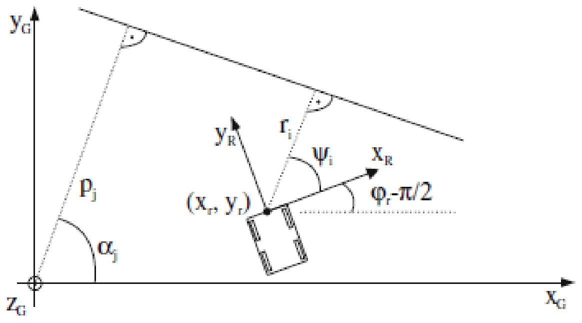

Fig. 3. The parameters (p_j, α_j) of the lines of the global map, according to the global coordinates. And the parameters (r_i, ψ_i) of the lines of the local map according to the robot coordinates.

Least Squares Method

We will be explaining this method for the parameters ψ and r, but all the parameters stated in (Eq.8 and 9) can be obtained in the same way.

If a cluster of points belongs to a vertical line we will not be able to apply this method since the value of the slope of the line will be infinite. To prevent this, we first calculate the approximated value of the slope for each line using the first and last points of the cluster. If the absolute value of the slope is bigger than 1, then we apply a rotation of $-\frac{\pi}{2}$ to all the points of the cluster. This is done by exchanging the vector **x** with the vector **y** and **y** with the vector **-x**. In the end of the method we need to apply a rotation of $\frac{\pi}{2}$ to ψ to obtain the correct line parameters.

We then apply the following equations,

$$\hat{\theta} = [\hat{k}_l, \hat{c}_l]^T = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{y}, \quad \mathbf{U} = \begin{bmatrix} \mathbf{x}(1) & \cdots & \mathbf{x}(n) \\ 1 & \cdots & 1 \end{bmatrix}^T,$$

$$\mathbf{y} = [\mathbf{y}(1), \cdots, \mathbf{y}(n)]^T, \tag{10}$$

where n is the number of points of the cluster; $\hat{h}_l$ and $\hat{c}_l$ correspond to the line parameters of the line equation in the form $y = k_l x + c_l$. Now we obtain ψ and r with

$$r(\hat{k}_l, \hat{c}_l) = \frac{\hat{c}_l}{\sqrt{\hat{k}_l^2 + 1}} sign(\hat{c}_l), \tag{11}$$

$$\psi(\hat{k}_l) = \arctan 2\left(\frac{sign(\hat{c}_l)}{\sqrt{\hat{k}_l^2 + 1}}, \frac{-\hat{k}_l}{\sqrt{\hat{k}_l^2 + 1}} sign(\hat{c}_l)\right), \tag{12}$$

Estimation of Line Parameters Covariances

For the update step of the EKF we will need to compute the covariance matrix R_i that contains the variance of the line parameters r_i and ψ_i and the covariances between them.

$$\mathbf{Ce} = var(\mathbf{y}(j))(\mathbf{U}^T\mathbf{U})^{-1} = \begin{bmatrix} var(\hat{k}_l) & cov(\hat{k}_l, \hat{c}_l) \\ cov(\hat{k}_l, \hat{c}_l) & var(\hat{c}_l) \end{bmatrix}, \tag{13}$$

$$var(\mathbf{y}(j)) = \frac{\sum_{j=1}^{n}(\mathbf{y}(j) - \hat{\mathbf{y}}(j))^2}{n-1}, \hat{\mathbf{y}}(j) = \hat{k}_l \cdot \mathbf{x}(j) + \hat{c}_l \tag{14}$$

where $var(\mathbf{y}(i))$ is the vertical error variance of the line-segment points (**x**(j),**y**(j)) (j=1,...,n) according to the estimated line with the parameters $\hat{k}_l$ and $\hat{c}_l$. Knowing the variances and covariances between the parameters $\hat{k}_l$ and $\hat{c}_l$ (Eq. 13), we can now calculate the variances and covariances between the parameters r and ψ as follows.

$$\begin{aligned} var(\psi) &= K_{\psi k}^2 var(\hat{k}_l), \\ var(r) &= K_{rk}^2 var(\hat{k}_l) + K_{rc}^2 var(\hat{c}_l) + 2K_{rk}K_{rc} \cdot cov(\hat{k}_l, \hat{c}_l), \\ cov(r, \psi) &= K_{rk}K_{\psi k} var(\hat{k}_l) + K_{rc}K_{\psi k} \cdot cov(\hat{k}_l, \hat{c}_l), \\ cov(\psi, r) &= cov(r, \psi), \end{aligned} \tag{15}$$

where

$$K_{rk} = \frac{-\hat{c}_l\hat{k}_l}{\sqrt{\hat{k}_l^2+1}(\hat{k}_l^2+1)} sign(\hat{c}_l), \quad K_{rc} = \frac{sign(\hat{c}_l)}{\sqrt{\hat{k}_l^2+1}}, \quad K_{\psi k} = \frac{1}{\hat{c}_l^2+1}. \tag{16}$$

3.3 Matching Line Segments

The parameters of the lines of the global map, α and p (Eq. 9), are all collected in a vector **G**

$$\mathbf{G} = [p_1, \alpha_1, ...,, p_{n_G}, \alpha_{n_G}]^T, \tag{17}$$

where n_G is the number of lines of the global map. The vector **G** is computed once and is always the same since the global map is static.

The parameters of the lines of the local map, ψ and r (Eq. 8), are all collected in the vector **z**, (k) (Eq. 18), which are based on the current LRF scan,

$$\mathbf{z}(k) = [r_1, \psi_1, ...,, r_N, \psi_N]^T. \tag{18}$$

The robot's pose correction is based on the difference between the line parameters of the local environment map and the line parameters of the global map, transformed into the robot's coordinates. This transformation is given by

$$C_j = p_j - \tilde{x}_r(k)\cos\alpha_j - \tilde{y}_r(k)\sin\alpha_j, \tag{19}$$

$$\begin{aligned} \begin{bmatrix} \hat{r}_i \\ \hat{\psi}_i \end{bmatrix} &= \mu_i(\tilde{\mathbf{x}}_p(k), p_j, \alpha_j) \\ &= \begin{bmatrix} |C_j| \\ \alpha - (\hat{\varphi}_r - \frac{\pi}{2}) + (-0.5sign(C_j) + 0.5)\pi \end{bmatrix}, \end{aligned} \tag{20}$$

where $\tilde{\mathbf{x}}_p$ (Eq.5) denotes the prediction of the robot's pose. The observation model can then be defined by the vector

$$\mu(\tilde{\mathbf{x}}_p(k)) = [\mu_1(\tilde{\mathbf{x}}_p(k), p_1, \alpha_1)^T, ..., \mu_{n_G}(\tilde{\mathbf{x}}_p(k), p_{n_G}, \alpha_{n_G})^T]^T \tag{21}$$

For each cycle we then want to find the lines of the global map that correspond to each line of the local map. For this we need to find the line of the global map (transformed into the robot coordinates) that is closest to each line of the local map. We first obtain a point in Cartesian coordinates for each line. For the lines of the global map we use Eq.22 and for the local lines we use Eq.23, for $j = 1, ..., n_G$ and $i = 1, ..., N$.

$$x_{\mu_j} = \cos\hat{\psi}_j \hat{r}_j, y_{\mu_j} = \sin\hat{\psi}_j \hat{r}_j, \tag{22}$$

$$x_{z_i} = \cos\psi_i r_i, y_{z_i} = \sin\psi_i r_i \tag{23}$$

With this we compute a matrix $v(k) \in \mathbb{R}^{2n_G \times N}$

$$v_{ij}(k) = \sqrt{(x_{z_i} - x_{\mu_j})^2 + (y_{z_i} - y_{\mu_j})^2}. \tag{24}$$

For each column of $v(k)$ we find the line that contains the minimum value, making it possible to obtain pairs of lines: each local line has an associated global line. With this pairs we compute a matrix $V(k)$

$$V(k) = \begin{bmatrix} r_1 - \hat{r}_{min_1} \\ \psi_1 - \hat{\psi}_{min_1} \\ \vdots \\ r_N - \hat{r}_{min_N} \\ \psi_N - \hat{\psi}_{min_N} \end{bmatrix}. \tag{25}$$

We also compute a matrix $\mu_{minim}(\tilde{\mathbf{x}}_p(k))$

$$\mu_{minim}(\tilde{\mathbf{x}}_p(k)) = [\hat{r}_{min_1}, \hat{\psi}_{min_1}, ..., \hat{r}_{min_N}, \hat{\psi}_{min_N}]^T. \tag{26}$$

Matching Step

We now have N pairs of lines but we have to analyse which pairs are valid. This is called matching and is necessary because some lines may correspond to moving objects of the environment. Since our map is composed only by static objects, we are going to find a pair but not a match, because the lines from the local map do not really correspond to the paired lines from the global map. The valid pairs can be found by applying

$$\begin{gathered} r_i - \hat{r}_{min_i} < T_r \\ and \\ \psi_i - \hat{\psi}_{min_i} < T_\psi \end{gathered} \tag{27}$$

All the differences between pairs of lines that were above the threshold are discarded. We then compute a new matrix $V_{match}(k)$ that is very similar to the matrix in Eq. 25 but contains only the differences between the line segments that passed the matching step.

We also compute a new matrix $\mu_{match}(\tilde{\mathbf{x}}_p(k))$ that is very similar to the matrix in Eq. 26 but contains only the parameters of lines that were matched.

3.4 Update Step

At this point we only need to compute the update step

$$\begin{gathered} \hat{\mathbf{x}}_p(k) = \tilde{\mathbf{x}}_p(k) + \mathbf{K}(k)\mathbf{V}_{match}(k), \\ \hat{\mathbf{P}}(k) = \tilde{\mathbf{P}}(k) - \mathbf{K}(k)\mathbf{H}(k)\tilde{\mathbf{P}}(k), \\ \mathbf{K}(k) = \tilde{\mathbf{P}}(k)\mathbf{H}^T(k)(\mathbf{H}(k)\tilde{\mathbf{P}}(k)\mathbf{H}^T(k) + \mathbf{R}(k))^{-1}, \\ \mathbf{H}_{ij}(k) = \left.\frac{\partial \mu_{match_i}}{\partial \tilde{\mathbf{x}}_{pj}(k)}\right|_{\tilde{\mathbf{x}}_p(k)}, \end{gathered} \tag{28}$$

where $\hat{\mathbf{x}}_p(k)$ denotes the state update; $\hat{\mathbf{P}}(k)$ denotes the covariance matrix of the state update error; $\mathbf{K}(k)$ is the Kalman gain; and $\mathbf{R}(k)$ is the output-noise covariance matrix associated to the vector $\mathbf{z}(k)$. $\mathbf{R}(k)$ has a block-diagonal structure, where the *i-th* block is given by

$$\mathbf{R}_i(k) = \begin{bmatrix} var(r_i) & cov(r_i,\psi_i) \\ cov(\psi_i, r_i) & var(\psi_i) \end{bmatrix} \tag{29}$$

with $var(r_i)$, $cov(r_i,\psi_i)$, $cov(\psi_i, r_i)$ and $var(\psi_i)$ defined in Eq. 15.

4 Experimental Results

After the algorithm was implemented, we defined the sampling time as $T = 100ms$, the distance between two opposite wheels as $L = 0.400m$, the radius of the wheels as $R = 0.109m$ and δ = 0.01, in order to start testing the performance of the EKF. The threshold values used to perform clustering were $T_{d_{points}} = 0.15m$ and $T_{a_{points}} = 0.30rad$. These two parameters were estimated based on visualizations of the data received from the Laser Rangefinder SICKLMS200. To obtain line-segments from the clusters, the minimum points required were $n_{min} = 4$. For the initial position of the robot, the pose is the origin of the referential and the orientation is $\psi = \frac{\pi}{2}$.

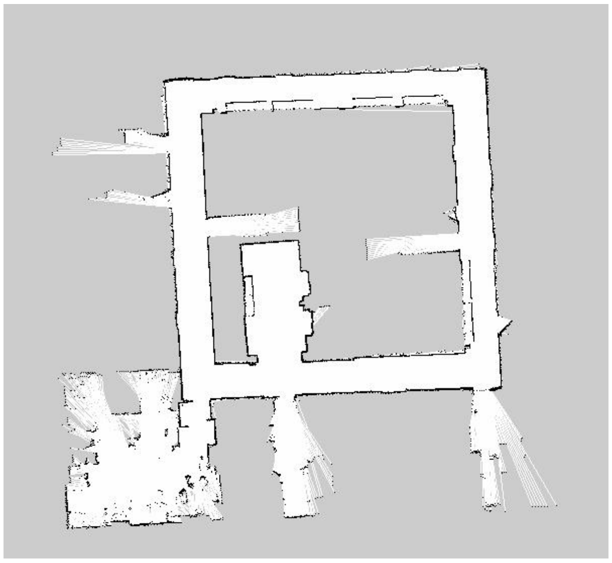
Fig. 4. Global map of the fifth floor of the North Tower, obtained through the SLAM package

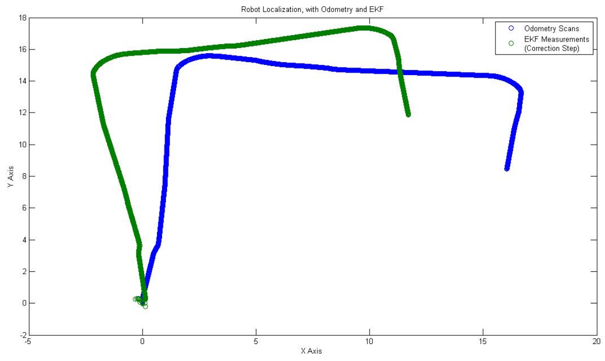

Fig. 5. Representation of the trajectory made by the robot. The points in blue are the values obtained with odometry. The points in green are the values obtained with EKF, with small number of matched lines.

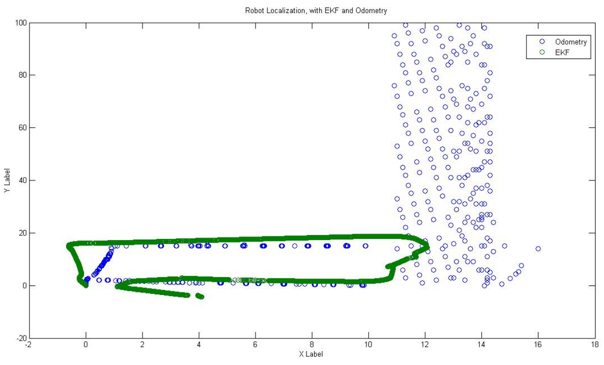

Fig. 6. Representation of the trajectory made by the robot. The points in blue are the values obtained with odometry. The points in green are the values obtained with EKF, with a representative number of matched lines.

The first step in a localization problem is to create a visual representation of the environment where the robot is in. For that, it was created a map using the SLAM package, represented in Fig. 4.

To analyse the trajectory of the robot using EKF, we did two experiments. The first one with a small number of matches and the second one with a considerable number of matches.

For the first experiment, as we can see in Fig. 5, the matching process did some small corrections, but the number of matched lines was rather small. In the cases where there are no matched lines, the value of the state update is the same as the state prediction. So, in this case, the update pose was close to the odometry values. It should be noted that we are in presence of a case where the odometry values have a low error. In the case of Fig. 5, the covariance matrix stays almost static, which is a result of the small number of matched lines.

If the number of matches increases significantly, it means that the covariance matrices are being updated and the corrections made to the trajectory are being done, meaning that the knowledge of the robot's current position is close to reality.

We then changed the values of the thresholds for the matching algorithm to $T_r = 0.20m$ and $T_\psi = 0.5rad$ and obtained the values presented in Fig. 6, with a good number of matches. The path of the robot was very closed to the real trajectory in the fifth floor. We note that we are in a case that the odometry values have a high error. Of course, the values of the update state are not perfect in some situations. One of the reasons for this to happen is of environmental nature where, for example, an open door might be closed in the global map, contrary to the local scan, or some people might be walking in front the laser. In both of this cases, the robot can change his location but after a while the EKF value can converge to a better one.

Another problem that the robot faces is the odometry. At the beginning of the robot's trajectory, the position given by the EKF and the odometry are very close, but after some time the odometry is completely wrong, which could be a problem in regards to the robot's localization. With the laser visualizations and the previous location knowledge, represented not only in the position but also on the covariance matrices, the robot is able to correct those deviations and almost ignore the unreliable data.

The thresholds for the matching algorithm are $T_r = 0.20m$ and $T_\psi = 0.5rad$.

5 Conclusion

In this paper, an EKF algorithm is implemented in order to achieve the most accurate localization of a mobile robot with four wheels. First, it is necessary to linearise the kinematic model of the robot to perform the prediction step of the EKF. To perform the matching step, it is necessary to extract the line parameters from the global map of the fifth floor in the North Tower of Instituto Superior Técnico and extract the data from the Laser Rangefinder and compare both lines created using the LSQ method. If the lines from the global map are similar enough to the lines that the robot observes, then the matching can be done and the robot will know that the line he observes is in a determined position in the global map. Using that information, the update step is performed, updating the position of the robot and also updating the covariance matrices.

During the whole project, we faced several challenges. All of the algorithm was implemented using Python and the Robot Operating System, which were tools hard to grasp for us at the start, since none of the elements of the group knew them before the beginning of the academic term. Another problem arised during the creation of clusters of points from the global map. Using the Corner Harris detection method, the lines created were not perfect. So, in order to achieve the maximum similarity, we did the extraction of the first and last points of each line by observing the map. In this report, we did not present the values of the covariance matrices, but we will present them in the poster.

References

[1] L. Teslic, I. Škrjanc, G. Klančar, *EKF-Based Localization of a Wheeled Mobile Robot in Structured Environments* J Intell Robot Syst, 2010

[2] C. Harris, M. Stephensčar, *A combined corner and edge detection* Alvey Vision Conference, 1988

[3] P. Lima, *Notas das Aulas Teóricas* , Lisboa - Instituto Superior Técnico, 2015