EKF-SLAM using visual markers for ITER's Remote Handling Transport Casks

Diogo Oliveira, *81844,* Luís Simões, *81282,* Mariana Martins, *80856,* and Miguel Paulino, *79168*

Abstract—Nuclear energy requires operations to be performed by remote handling, due to radiation levels. Thus, accurate vehicle mapping and localization in this complex environments is essential to move forward on today's clean energy pursue.
SLAM(Simultaneous Localization And Mapping) with an EKF approach is one method of achieving this when the reliance on external sensors to the robot is not a plausible scenario. On this article, the aim is to implement and test this algorithm using artificial landmarks to help a International Thermonuclear Experimental Reactor(ITER) remote handling transport cask prototype navigate through space, and discuss the implementation on the real ITER cask.

Index Terms—EKF, SLAM, ITER, Visual Markers, arUco

1 Introduction

This article describes and discusses an implementation of the Extended Kalman Filter Simultaneous Localization And Mapping(EKF-SLAM) applied to the International Thermonuclear Experimental Reactor(ITER)[1] Remote Handling Transport Casks. The casks and Plug Remote Handling System(CPRHS) supported by a Cask Transfer System(CTS) are used to transport heavy loads in the rad-high environment of the Tokamak Building and Hot Cell Building [1].

In this context it is important that the vehicle can move autonomously and with high enough precision to ensure a $30cm$ safety distance to the nearest obstacle at all times. As such, the precise location of each vehicle needs to be known at each instant.

SLAM is the problem of creating and updating a map while simultaneously keeping track of the robots position in the said map. One may see the scenario presented as a SLAM problem, where the cask should be aware of its location and maintain a map of the premises. To create a map, the vehicle or robot uses environment features through visual markers or ultrawide band for example.

All the code used to implement the methods described below is available at https://github.com/Mrrvm/SA.

2 Methods and Algorithms

2.1 Notation

The notation utilised throughout this report follows the notation described in this section. Figure 1 illustrates what are the different variables. As can be seen in the figure, two main reference frames are used: the world frame and the vehicle frame.

The odometry, O, measured by the vehicle is expressed in the vehicle frame. $O = [V_F, \theta_F, V_R, \theta_R]$, where V is the wheel speed in m/s(converted from the original in $rpms$) and θ is the angle of the wheel. The subscripts F and R indicate the wheel the variable refers to, respectively front and rear wheel.

The measurement data, m, is also expressed in the vehicles reference frame. $m = [\phi, d]$ where ϕ is the the angle at which the measurement was made and d is the measured distance to the target.

The robot pose is expressed in the world frame and is given by $x = [x_r, y_r, \alpha]$ where x_r and y_r are the coordinates of the cask and α is its orientation.

Similarly, each landmark is defined as $L = [L^x, L^y]$ where L^x and L^y are the coordinates of the landmark.

The distance between wheels is defined as W. The time is identified as t and each landmark by $i(i = 1, ..., n)$. The map produced by the algorithm is defined as M. The covariance matrix is defined as Σ.

2.2 SLAM

SLAM problems can be divided into full SLAM and Online SLAM. Full SLAM estimates the entire path and map given all landmark measurements and all odometry measurements. Online Slam, on the other hand, seeks to recover only the most recent pose and map. The situation presented in this paper corresponds to an Online SLAM problem, as only the current position and map needs to be known, this is expressed by the probability given in equation 1.

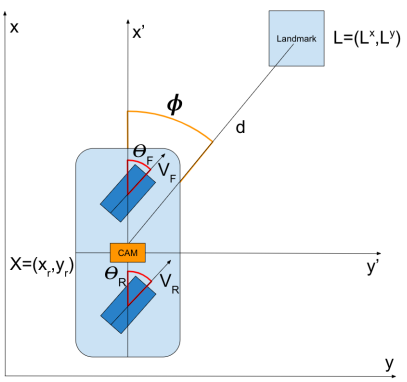

Fig. 1. ITER scheme with variables

Manuscript sent December 10, 2018.

1. https://www.iter.org/

$$p(x_t, M|z_{1:t}, u_{1:t}), \tag{1}$$

where z is the observations vector and u is the odometry vector.

SLAM algorithms can be separated into three different stages [2],

- Movement model - Estimate the robots position, based on the odometry measure and the movement model of the robot.
- Inverse Observation Model - If a landmark that has not yet been seen is detected then it's added to the map.
- Direct Observation Model - If a landmark that is already part of the map is detected, the new measurements are used to decrease both localisation and landmarks uncertainties.

The first step models the change in the vehicles pose in accordance to the odometry measurements collected. The measurements collected may be noisy and the movement model selected may not be able to reliably describe all the vehicle dynamics, e.g. slippage, drift and uneven terrain effects. As such, the estimated position of the robot has an associated error that increases every time the robot moves.

The second step happens if a new landmark is seen in the current iteration. When a new landmark is detected, it is added to the map and its position is estimated. This estimation has an associated uncertainty that results from the compound of the error present in the sensor data that detected the landmark and the estimated vehicle pose.

The last step occurs when a landmark that is already part of the map is detected. In this case, the measurement of the landmark is used to correct the uncertainties of both the robot position and the landmarks estimated location.

2.2.1 EKF-SLAM

There are several approaches to a SLAM problem, for example Graph-Based SLAM, FastSLAM or EKF-SLAM. This article will focus on EKF-SLAM, which employs an extended Kalman filter as the central estimator [2].

The extended Kalman filter is a nonlinear version of the Kalman Filter. It implements a Kalman filter for a linearized system dynamics of an originally non-linear system [3]. The Kalman Filter, based on the control input of the system and observations of its state, estimates the hidden state of a linear time-variant system [4]. It was invented as a technique for filtering and prediction in linear Gaussian systems [5].

2.3 Map

The map chosen for this work is a vector consisting on the concatenation of the robot pose (x) and landmark positions ($L_1 \cdots L_n$) as expressed in 2.

$$M = \begin{bmatrix} x & L_1 & \cdots & L_n \end{bmatrix}^T \tag{2}$$

A landmark is a feature of the environment that is identifiable, it can be either natural or artificial. An example of an artificial landmark could be a QR code and a natural landmark could be a tree or rock formation.

2.3.1 Co-variance Matrix Σ

The co-variance matrix 3 is what allows the algorithm to know how much it can trust the current map vector. To measure the uncertainty of all map features and their relation with each others, this matrix will be square with the same length as the map vector.

$$\Sigma = \begin{bmatrix} \Sigma_x & \Sigma_{xL} \\ \Sigma_{Lx} & \Sigma_L \end{bmatrix}, \tag{3}$$

where Σ_x is the co-variance matrix of the robot pose, Σ_L is the co-variance matrix of each landmark and Σ_{xL} and Σ_{Lx} represent the cross-variance between the robot pose and the landmarks.

The covariance matrix changes with every iteration of the algorithm. When something that increases the uncertainty in the map happens, the values on the matrix increase. As opposed to when seeing an already observed landmark or a similar event that decreases uncertainty the values on the matrix decrease.

3 Implementation

3.1 Map

The map is stored as a vector and its size is the robot pose plus twice the number of seen landmarks. The size is explained by the need to store the position for each landmark, corresponding to L^x and L^y. In order to keep track of the uncertainties, both the uncertainty of the robot pose and the landmark positions, a co-variance matrix (Σ), which is updated in every step, is used, just like the map vector.

3.2 Landmarks

The landmarks used throughout the work were visual markers from the arUco library, an OpenSource library for camera pose estimation [2]. Each marker has an identifying number associated that can be extracted when the marker is detected in an image, that distinguishes each landmark. The size of each square marker used was $15cm$.

Through the library API, the rotation and translation of the arUco's reference frame to the Camera's reference frame is obtained. This allows the distance, d, and angle, ϕ, from the robot to the marker to be extracted by equations 4 and 5.

$$t_{cam\ to\ arUco} = -R_{arUco\ to\ cam} \cdot t_{arUco\ to\ cam} \tag{4}$$

$$t_{cam\ to\ arUco} = [x\ y\ z],\ \phi = atan(\frac{x}{z}),\ d = ||(x, z)||_2 \tag{5}$$

Since the vehicle motion is only in 2D, the height, y, of the marker is ignored when doing the calculations, thus having the angle and distance exclusively as an horizontal difference. The camera was calibrated with arUco boards in order to have reliable measurements taken [3].

The camera used was an uEye LE USB 3.1 Gen 1 [4] which provides a wide view of space.

2. https://www.uco.es/investiga/grupos/ava/node/26
3. https://docs.opencv.org/3.1.0/da/d13/tutorial_arUco_calibration.html
4. https://en.ids-imaging.com/store/products/cameras/usb-3-1-cameras/ueye-le-usb-3-1-gen-1/show/all.html

3.3 EKF-SLAM

The EKF-SLAM algorithm implemented is described by the pseudocode in section 3.3.1. It assumes the casks initial pose as the origin of the world reference frame.

3.3.1 EKF-SLAM Pseudocode

$data \leftarrow$ INPUT odometry data
$[\overline{x}, \overline{\Sigma}] \leftarrow MovementModel(\overline{x}, lastcontrol, t)$
$lastcontrol \leftarrow [data.odom, data.time]$
if $data.Landmarksdetected.number \geq 1$ **then**
  for all $data.Landmarksdetected$ **do**
    $y_i \leftarrow iterator.measurements$
    if $y_i \notin Map$ **then**
      $[Map, \overline{\Sigma}] \leftarrow InverseObservationModel(\overline{x}, y_i)$
    end if
    $[\overline{Map}, \overline{\Sigma}] \leftarrow DirectObservationModel(Map)$
    $[Map, \Sigma] \leftarrow MatchingStep(\overline{Map}, \overline{\Sigma})$
  end for
end if

3.3.2 Movement Model

The vehicles movement model is described in equation 6 [4].

$$\overline{x}_t = \overline{x}_{t_r} + \frac{t - t_r}{2} \begin{bmatrix} V_F \cos(\theta_r^t + \theta_F) + V_R \cos(\theta_r^t + \theta_R) \\ V_F \sin(\theta_r^t + \theta_F) + V_R \sin(\theta_r^t + \theta_R) \\ \frac{V_F \sin\theta_F + V_R \sin\theta_R}{\frac{W}{2}} \end{bmatrix}, \tag{6}$$

where $\overline{x}_t$ is the estimated position of the vehicle at time t, $\overline{x}_{t_r}$ is the estimated robot position when the last odometry measures were taken and t_r is the time instant when the robot pose was last estimated.

In the EKF-SLAM algorithm, the movement model is used on the prediction step, which predicts the position of the robot and also updates the co-variance matrix, increasing the uncertainty of the robot pose.

3.3.3 Inverse Observation Model

The inverse observation model, described in equation 7, is used when a landmark that has not yet been seen is detected. From the sensor data, specifically from the distance d and angle ϕ, the inverse observation model estimates the landmarks location($\overline{L}_{it}$) and adds it to the map. As described in section 3.2, at each time instant t, several measurements can be made, one for each landmark i.

$$\overline{L}_{it} = \overline{x}_t + d_{it} \begin{bmatrix} \cos(\theta_r^t + \phi_{it}) \\ \sin(\theta_r^t + \phi_{it}) \end{bmatrix} \tag{7}$$

Also when a new landmark is seen, the map vector grows two positions in order to save the x and y coordinates of the new landmark. If the map grows, the co-variance matrix also grows, so whenever a new landmark is seen we add two columns and two rows to the co-variance matrix with values that represent the uncertainty of the newly measured landmark.

3.3.4 Direct Observation Model

Whenever a previously mapped landmark is detected the direct observation model calculates the expected measurement values, distance d and angle ϕ, based on its current estimation for the landmarks position $\overline{L}$. By comparing the expected and real measurements it is possible to improve the estimation of the vehicles position and of the location of all landmarks on the map (see equations 8 and 9).

$$\overline{d}_{it} = \left\| \overline{L} - \overline{x_t} \right\|_2 \tag{8}$$

$$\overline{\phi}_{it} = \arctan\left(\frac{\overline{L_{it}^x} - \overline{x_t^x}}{\overline{L_{it}^y} - \overline{x_t^y}} \right) - \theta_r^t \tag{9}$$

3.3.5 Matching Step

The matching step is meant to discard outliers that might appear in some observations. Some measurements made with the ArUco library do not identify the marker correctly, despite being a rare occurance, a miss-identification of a landmark might impede the algorithms convergence. The matching step prevents such miss-identifications from causing irreversible damage.

The matching step simply compares the range and bearing given by the measurement and the output of the Direct Observation Model for the same landmark and decides if that measurement is plausible or not. If the matching step decides that the measurement is an outlier, then it discards the measurement and doesn't update the map neither the co-variances matrix, if it is not an outlier than our algorithm can proceed, using the Kalman gain to update the map and the co-variance matrix.

4 Experimental Results

For the experimental results the algorithm was first tested in synthetic data and then with real data extracted using a scaled model of the ITER cask.

4.1 Synthetic data

The synthetic data is generated utilising the movement and observation models. The synthetic data generator, or simulator, takes odometry measurements and landmark locations as inputs. The simulator developed allows the landmark and odometry measurements to be generated randomly or with some predefined values. The premade paths are a straight line forward, a rotation, a circle, a square, a triangle and a path similar to two rectangles shown in figure 2.

From the given odometry values, the movement model and assuming that the starting pose is the origin of the world reference frame, the simulator generates the robot pose at all time instants.

After calculating the robot pose at all instants, it is now possible to use the direct observation model and the landmark locations to calculate the measurements made by the sensors at a user specified time interval. The simulator assumes the camera has an effective view range of $5m$ with a 180 degree field of view relatively to the front of the vehicle.

The odometry values and measurement data are then stored in a data structure and noise is added to both.

Figure 2 shows the simulation path and landmark location, on the left, and the EKF-SLAM estimated path and seen landmarks estimated location, on the right. It's important to note that throughout the entire trajectory made, the robots

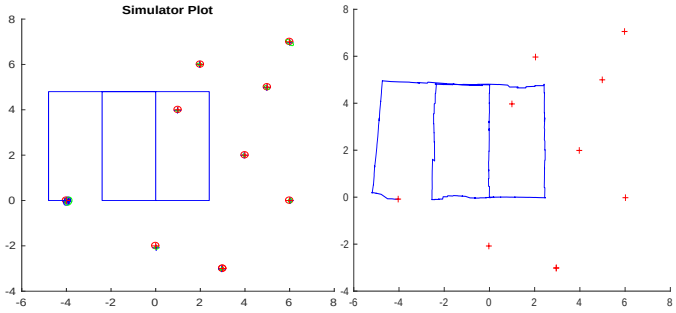

Fig. 2. The figure on the left represents the robot path and landmarks position without noise. The figure on the right represents the estimated path of the robot and landmarks position by the EKF-SLAM algorithm. Blue line is the robot's trajectory, red crosses are the landmarks' positions.

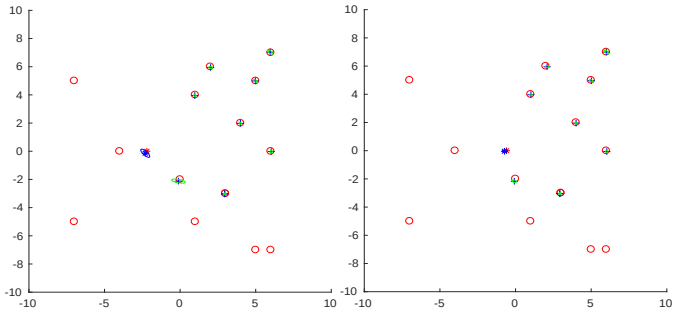

Fig. 3. Estimation of the position of the cask (blue star) the landmarks (blue cross) with their respective uncertainties (blue and green ellipsis) overlayed to the groundthruth(red star and red circles) at times 49 (left) and 56 (right).

orientation has not changed, it is always concordant with the x axis.

From the plot of the predicted trajectory one can see that it is similar to the original plot. The trajectory error, however, is larger on the first squares descending edge. This can be explained by the distance to the landmarks. When moving backwards on the top edge the landmarks stop being detected, as they are more than $5m$ away from the sensor, and from that point onwards the robot pose is estimated based solely on odometry. The uncertainty of the pose and with it the error in the trajectory increases until the vehicle detects an already seen landmark, where it corrects its estimated pose, which in figure 2 happens around $[0, -0.25]$. Note that, since online-SLAM is used, the trajectory on previous instants will not be corrected, only the current estimate is corrected. The landmark's positions shown in the figure, however, are updated, as only the final map was printed.

Figure 3 shows two iterations of the algorithm during the simulation at times 49s and 56s. The blue and green ellipses shown in both sides represent the area where the algorithm believes the true location may be. As so, smaller ellipses represent a greater confidence in the algorithms estimation. At time 49s the algorithms uncertainty for both the robot pose and the closest landmark is larger than in time 56s. This happens because, as described above, the cask at time 49s had not seen a landmark for some iterations relying solely

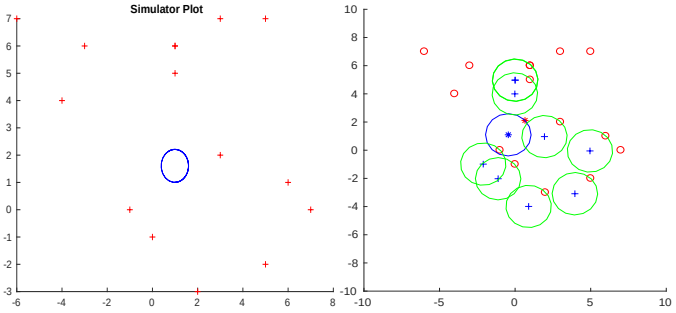

Fig. 4. Ground truth and trajectory on the left (red crosses are landmarks, blue line is trajectory). Final iteration of the algorithm that started with an offset of $[1, 1]$ on the right (blue crosses are estimated landmarks, red circles are simulated landmarks, green circles are landmarks' uncertainties, blue line is robot's trajectory).

on the odometry measurements which increase the uncertainty. When the new landmark was detected it had a high uncertainty associated. However, at instant 56s, landmarks that have already been seen and have a small uncertainty come into range and the algorithm corrects its estimations, which decreases the uncertainty of both the landmark and robot pose estimation.

Figure 4 utilises the predefined circle trajectory, where the robot always faces the direction of movement, and starts with an offset of $[1, 1]$, meaning the algorithm assumes its starting position at world coordinates $[0, 0]$ while in reality its position is $[1, 1]$. The initial uncertainty was also changed to reflect the error in the assignment.

The right side of figure 4 shows the map at the last time instant. It shows that the map is reconstructed accurately but is deviated from the real location by the initial offset, this can be observed by comparing the estimates, blue star for robot pose and blue crosses for landmark locations, with the ground truth locations, red star and red circles respectively. The image also shows a large uncertainty on the last iteration. This happens because without further information about the world reference frame and about its possible error the algorithm has no means of self-correcting past its initial certainty. This could be counteracted by feeding external information into the algorithm, for instance by assigning a fixed location to a certain landmark.

This experiment with an initial uncertainty is interesting to prove the robustness of the implemented algorithm, since the first seen landmarks will have higher uncertainty so as the robot pose, making it harder for the algorithm to converge to the right map.

4.2 Landmarks

Using the scenario of figure 5, the obtained results to the right side landmark were $1.08m$ on the z direction and $48cm$ on the x direction. Having tried this testing approach several times, the error of the landmark's estimation is around $2 - 5cm$.

4.3 Real data

As written before, it was used a prototype of the ITER cask built upon a LEGO NXT hardware in order to get real

estimated results of the EKF-SLAM algorithm for the ITER cask movements.

Moving from a synthetic simulation to a real data simulation is not straight forward since we can't simply define how much will the uncertainty of the measurements be, for both odometry and observation measurements.

Figure 6 shows the real-life scenario and position of the landmarks while the right side shows estimated map of the landmarks and the estimated robot trajectory. As expected, while in front of the landmarks the trajectory is a straight line. However, when the landmarks can no longer be detected the error increases and as can be seen in the image the trajectory is no longer straight.

The right turn seen in the figure is explained by the odometry measurements and how the data was created. The ITER cask scale model has a tendency to turn either side. On testing the deviation measured was approximately $0.7m$ for a distance of $3m$. Note that it was not due to a certain angle at departure but the angle of the robot changed as it moved. During later experiments the direction of the deviation was not consistent, changing the side to which it happened. This extreme deviation is suspected to be due to uneven support wheels on the sides of the cask, which allowed it to wobble. This behaviour is aggravated by defects on the drive wheels, for instance holes in the rubber of the tires. The effects described are not considered in the model and are, therefore, treated as noise. The noise caused by this dynamic is not gaussian.

Due to this error, the user had to manually compensate this effect by turning every time the vehicle steered off course. These corrections had an impact on the odometry measurements as the wobble is not detected by the odometry but the corrections are. Which causes the movement model to assume a turn where none happened.

The effect of the landmarks on the robot pose when they are detected again can be seen in figure 7. When the landmarks are not considered (right image) one can see that the backwards motion of the robot is a straight line, as there was no need to correct any deviations. On the other hand, the landmarks correct the vehicles estimated position and

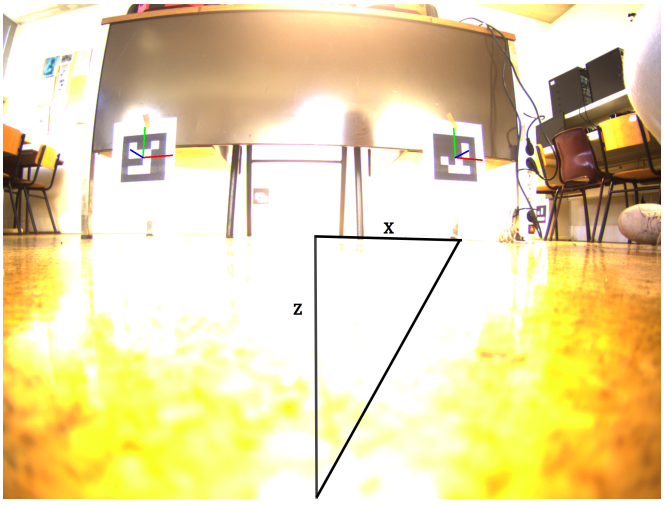

Fig. 5. Landmark pose accuracy test. The landmark on the right side is at 1.10m on the z direction and at 46cm on the x direction.

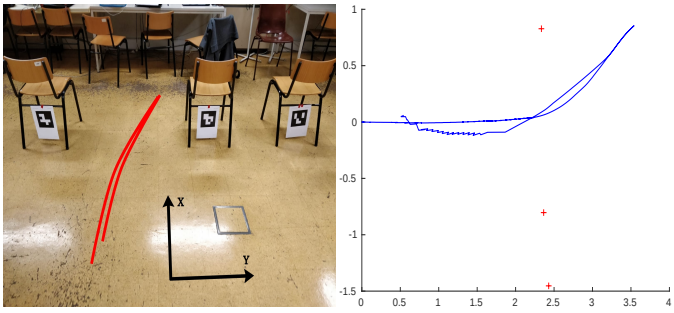

Fig. 6. Experiment where the cask moves forward between the first and second chairs from the left to the right and then backwards returning to the original position. The real life scenario(left) and the map generated(right)

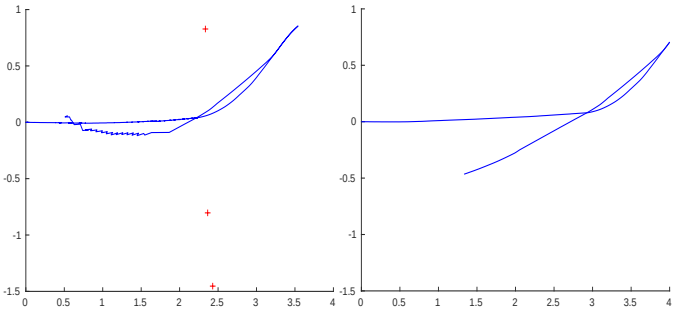

Fig. 7. Same experiment as 6, with(left) and without(right) landmarks.

from this correction results the stepped curve seen in the trajectory estimated by SLAM.

Figure 8 shows the real-life scenario and position of the landmarks while the right side shows estimated map of the landmarks and the estimated robot trajectory. As expected, the estimated robot trajectory has an error and resembles a slightly slanted square due to ITER's deviation explained above. In this experiment the ITER never changed its orientation, as the wheels were rotated by 90 degrees in every corner. The curves observed in the estimated trajectory after the first and the second turns correspond to the ITER's deviation issue and reaching the third curve, when the first landmarks are seen again, the ITER's estimated position is corrected and, from the correction results, the "noisy" straight line is almost perpendicular to the ITER trajectory before the first curve.

In order to explore a bit more the kinematics of the ITER prototype and the robustness of the algorithm, a more complex path with lots of turns and changes of direction was defined.

This new path presented a large odometry error, one example of such an error can be seen on the 90^o turn estimated by the algorithm that in reality was a 180^o turn. From figure 9 one can see that for this large path with large stretches where no landmarks are visible the algorithm did not converge.

4.4 Computation time

The algorithm implemented has a complexity of $O(n^2)$ where n is the number of seen landmarks.

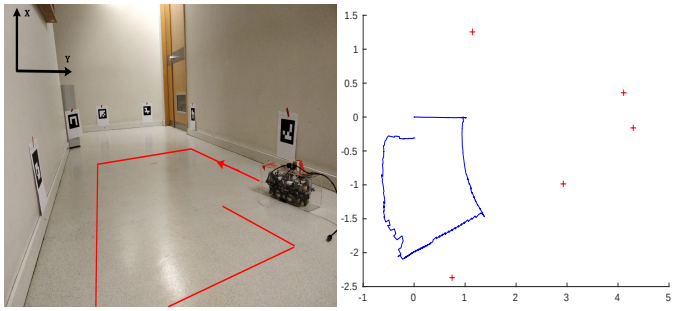

Fig. 8. Experiment where the cask draws a square. The real life scenario(left) and the map generated(right)

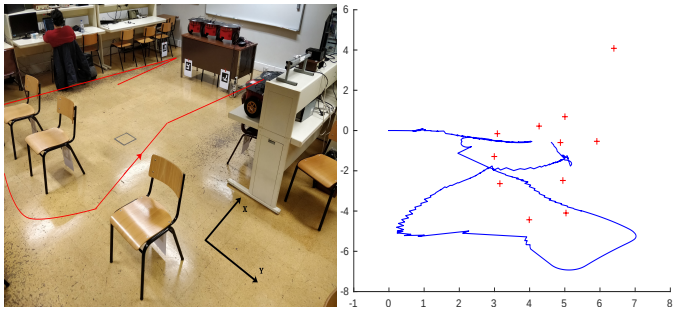

Fig. 9. Another experiment with the cask. The real life scenario(left) and the map generated(right)

This complexity is explained by the amount of calculations involving the co-variance matrix which, as told before, is a square matrix with a length of three positions for the robot pose plus two times the number of landmarks. The most complex computation is by far the Kalman gain (implemented on the update step), that requires a matrix inversion (equation 10).

$$K = \overline{\Sigma} H^T (H \overline{\Sigma} H^T + Q)^{-1} \tag{10}$$

Where K is the Kalman gain, H is the jacobian of the observation model and Q is a diagonal matrix with the uncertainty variances when observing landmarks.

The largest test performed with real data was the one described in figure 9 where eleven markers were detected by the vehicle. The maximum amount of time needed to run one iteration of the algorithm was $0.0331s$ which is faster than the sampling period for the odometry measurements which were $0.1s$ which in turn is faster than the camera frequency of $1fps$.

5 Conclusion

The EKF-SLAM algorithm is still considered a good approach to the SLAM problem, although the amount of requirements might lead the algorithm to diverge. As told in the beginning, this algorithm was built upon the assumption of Gaussian probabilities and relies on many linearizations. In the Real Data section (4.3) the odometry errors of the robot were discussed. Those uncertainties are far from Gaussian, so the assumption of Gaussian noise is an "over-assumption".

Bearing all that in mind our algorithm was specifically built upon the movement uncertainty of the cask so that it was possible to guarantee a high ratio of convergence and also a minimum error for long paths where the robot moves a long time without seeing any landmark.

EKF-SLAM does not seem to be an appropriate algorithm for the proposed problem. The vehicles will work in a highly dangerous environment moving heavy loads and as the algorithm has no guarantee of convergence it is not reliable enough.

5.1 Future work

5.1.1 *Co-variance Matrix*

The co-variance matrix is where the implementation part has a bigger impact, since the complexity of EKF lies on updating this matrix. Therefore, it could be taken advantage of the sparsity of most of the updates in order to increase the performance of the algorithm. This means, for the prediction step, where only some parts of the co-variance matrix are updated, it's possible to reduce the computation complexity by using the equations 11, 12 and 13.

$$\Sigma_x \leftarrow \frac{\partial f}{\partial x} \Sigma_x \frac{\partial f}{\partial x}^T + \frac{\partial f}{\partial n} N \frac{\partial f}{\partial n}^T \tag{11}$$

$$\Sigma_{xL} \leftarrow \frac{\partial f}{\partial x} \Sigma_{xL} \tag{12}$$

$$\Sigma_{Lx} \leftarrow \Sigma_{xL}^T \tag{13}$$

Since the movement model (f) is non-linear, we have to compute its jacobians with respect to the robot pose (x) and with respect to the noise (n) in order to update the co-variances matrix.

5.1.2 *Cooperative SLAM*

It was not the purpose of this work but it may be interesting to apply a way of cooperation between a number of real ITER vehicles.

The simplest first objective would be to produce the same map for every vehicle, and this is possible by deploying an anchor on the map, this anchor could simply be a landmark which we know exactly its position and orientation. The anchor would provide reliable information to every vehicle which would adjust the map in order to prevent offsets and different map orientations for different robots.

A more complex objective would be the idea of cooperation between vehicles, sharing information about their whereabouts and maps in order to build a large unique map for every vehicle without the need of each robot go through every portion of the map.

References

[1] Ferreira et al, Vehicle localization system using offboard range sensor network, 2013
[2] J. Sola, Simulataneous localization and mapping with the extended Kalman filter: 'A very quick guide... with Matlab code!', 2014
[3] M. Ribeiro, Kalman and Extended Kalman Filters Concept, Derivation and Properties, 2004
[4] R. Ventura, Derivation of the discrete-time Kalman filter, 2017
[5] M. Klingensmith, Statistical Techniques in Robotics (16-831, F10) Lectures 20, 21