Monte Carlo Localization of a Pioneer 3-DX Robot

João Mendes
Nº 95804
MEAer
Group 4.2

João Peixoto
Nº 95807
MEAer
Group 4.2

João Félix
Nº 97238
MEAer
Group 4.2

Tomás Luís
Nº 100101
LEEC
Group 4.2

***Abstract*—This report presents an approach to solve the localization problem using data obtained from a Pioneer 3D-X robot equipped with an Hokuyo laser range finder. It implements the Monte Carlo Localization algorithm while also taking into consideration the kidnapping problem, which occurs when the robot is taken to an arbitrary location.**

***Index Terms*—Introduction, Monte Carlo Algorithm, Models, Algorithm Implementation, Data Acquisition, Results, Conclusion**

I. Introduction

Robot localization is defined as the problem of estimating the robot's position and orientation, also known as its *pose*. Knowing where the robot is with precision is required in order for the robot to perform any meaningful task or action. As such, localization is a key problem in autonomous systems. This report implements and tests a technique called Monte Carlo Localization (MCL). MCL is a localization approach based on a particle filter, a non-parametric Bayesian filter that approximates the posterior distribution $\text{Bel}(x_t)$ using a finite number of samples (particles). One of the major benefits of this approach is that it can be used with nonlinear, non-Gaussian processes. For the localization problem, it is assumed that a known map of the surroundings already exists, thus avoiding the problem of simultaneous localization and mapping (SLAM).

II. Monte Carlo Algorithm

As mentioned, MCL uses a particle filter to estimate posteriors over robot poses. Particle filters are an implementation of the Bayes filter that represent the distribution using a set of samples, called particles, instead of a parametric form. As such, particle filters are non-parametric Bayes filters. Usually, particle filters construct the belief $\text{Bel}(x_t)$ recursively from $\text{Bel}(x_{t-1})$, a more efficient approach than storing the entire input history. In other words, the set of particles χ_t is constructed from the set of particles of the previous time step χ_{t-1}, along with the control input u_t, and measurement Z_t.

In general, the particle filter is composed of three different steps: prediction, update, and resampling. [1]

1) ***Prediction:*** A hypothetical state x_t is generated from the previous state x_{t-1} and the control input u_t, i.e., the posterior distribution at the previous time step is combined with the motion model to obtain the prior distribution of the current time step, the predicted state. [2]

2) ***Update:*** The importance factor, or weight, w_t is computed for each particle obtained in the prediction step. The importance factor is the probability of measurement Z_t given the particle x_t, i.e., $w_t = p(Z_t|x_t)$, obtained through the measurement model.

3) ***Resampling:*** The particles generated in the prediction step are randomly drawn, where the probability of drawing each particle is given by its importance factor computed in the update step. Thus, the distribution of the particles is changed, approximately according to the posterior $\text{Bel}(x_t)$. Since particles are drawn with replacement, there are likely to be present many duplicates where the weights are higher, while particles with lower weights tend to not be present in the new set χ_t. [1]

III. Models

A. Motion Model

An odometry motion model was considered. Odometry is the use of motion sensors to determine the change in the robot's position relative to a known position. In general, odometry is not suitable for use in long distances due to error accumulation; however, as the distances traveled by the robot will be small, in the order of a few meters, this drawback will not have a significant effect.

In the time interval $[t-1, t]$, the robot advances from pose x_{t-1} to pose x_t, while the odometry returns a relative advance from pose $\bar{x}_{t-1} = [\bar{x}\ \bar{y}\ \bar{\theta}]^T$ to $\bar{x}_t = [\bar{x}'\ \bar{y}'\ \bar{\theta}']^T$ (the bar denotes odometry measurements, in a coordinate system whose relation to the global coordinate system is unknown). Thus, the motion information is given by $u_t = [\bar{x}_{t-1}\ \bar{x}_t]^T$. u_t is then divided into three consecutive steps: a first rotation δ_{rot_1}, a translation δ_{trans}, and a second rotation δ_{rot_2}. Equations 1, 2 and 3 establish the relation between these transformations and the motion information u_t:

$$\delta_{rot_1} = \text{atan2}\left(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}\right) - \bar{\theta} \tag{1}$$

$$\delta_{trans} = \sqrt{\left(\bar{x} - \bar{x}'\right)^2 + \left(\bar{y} - \bar{y}'\right)^2} \tag{2}$$

$$\delta_{rot_2} = \bar{\theta}' - \bar{\theta} - \delta_{rot_1} \tag{3}$$

It is assumed that the motion error is given by independent noise ε_b with zero mean and variance b, usually a Gaussian distribution. With this in mind, the "true" values of these transformations, $\hat{\delta}_{rot_1}$, $\hat{\delta}_{trans}$ and $\hat{\delta}_{rot_2}$, can be obtained by subtracting the error from the measured transformations δ_{rot_1}, δ_{trans} and δ_{rot_2}:

$$\hat{\delta}_{rot_1} = \delta_{rot_1} - \varepsilon_{\alpha_1|\delta_{rot_1}|+\alpha_2|\delta_{trans}|} \tag{4}$$

$$\hat{\delta}_{trans} = \delta_{trans} - \varepsilon_{\alpha_3|\delta_{trans}|+\alpha_4|\delta_{rot_1}+\delta_{rot_2}|} \tag{5}$$

$$\hat{\delta}_{rot_2} = \delta_{rot_2} - \varepsilon_{\alpha_1|\delta_{rot_2}|+\alpha_2|\delta_{trans}|} \tag{6}$$

The constants a_i $(i = 1, ..., 4)$ are robot-specific parameters to be determined that model the error accumulation with motion.

With equations 4, 5 and 6, it is possible to obtain the true state x_t from x_{t-1}:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot_1}) \\ \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot_1}) \\ \theta + \hat{\delta}_{rot_1} + \hat{\delta}_{rot_2} \end{bmatrix} \quad . \tag{7}$$

[1]

B. Measurement Model

Measurement models describe the formation process by which sensor measurements are generated in the physical world. The probability model specifies the probability of the observations given a pose x_t and a map *m*. Observations are given by a vector of different individual observations $Z_t = (z_1, ..., z_t)$ where $z_i = (\rho_i, \theta_i)$, ρ_i the range measured and θ_i the yaw in relation to the orientation of the robot. Since each measurement Z_t is made of several different measurements z_t^i, it was assumed that these measurements z_t^i are conditionally independent and, as such, $p(Z_t|x_t, m) = \prod_{i=1}^{n} p\left(z_t^i|x_t, m\right)$. This assumption holds true in an ideal scenario, whereas in reality there might be a variety of different factors that lead to dependencies.

To determine the range to a given object, ray casting can be used. In an ideal world, the sensor would read the true distance, z_t^{i*}, to the object. In practice, however, this situation is impossible and the sensor will always return a value with noise. This noise will be modelled by a Gaussian distribution p_{hit} with mean z_t^{i*} and variance σ_{hit} limited by the maximum range of the sensor z_{max}:

$$p_{hit}\left(z_t^i|x_t, m\right) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{(z_t^i - z_t^{i*})^2}{2\sigma_{hit}^2}}, & 0 \le z_t^i \le z_{max} \\ 0\ , & \text{otherwise} \end{cases} \tag{8}$$

where η is a normalizing factor given by

$$\eta = \left(\int_0^{z_{\max}} \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{(z_t^i - z_t^{i*})^2}{2\sigma_{hit}^2}} dz_t^i \right)^{-1} \quad . \tag{9}$$

A measurement error related to random unexplained noise is modeled using a uniform distribution spread over the entire sensor measurement range $[0, z_{max}]$, because without the additional dependency on the maximum laser range, a particle located near the robot could have a near-zero likelihood due to high error, an undesirable situation for the purposes of this project:

$$p_{rand}\left(z_t^i|x_t, m\right) = \begin{cases} \frac{1}{z_{max}}, & 0 \le z_t^i \le z_{max} \\ 0\ , & \text{otherwise} \end{cases} \quad . \tag{10}$$

These two distributions are then mixed by a weighted average, defined by the parameters z_{hit} and z_{rand} subject to $z_{hit} + z_{rand} = 1$, as represented by equation 11:

$$p\left(z_t^i|x_t, m\right) = \begin{bmatrix} z_{hit} \\ z_{rand} \end{bmatrix}^T \begin{bmatrix} p_{hit}\left(z_t^i|x_t, m\right) \\ p_{rand}\left(z_t^i|x_t, m\right) \end{bmatrix} \quad . \tag{11}$$

[1]

IV. Algorithm Implementation

*1) **Initialization**:* The particles are initialized uniformly along the map and every weight is set to $\frac{1}{N}$, where N is the total number of particles. An alternative initial particle distribution was also considered following a Gaussian distribution along the initial position of the robot.

*2) **Prediction**:* The prediction step follows the methodology explained in section II and the motion model presented in section III-A.

*3) **Update**:* The update step is based on the methodology explained in section II and on the measurement model presented in section III-B. To compute the true distance z_t^{i*}, each laser beam is represented by a line segment between the points (x_1, y_1) and (x_2, y_2), where (x_1, y_1) is the current pose of the particle and (x_2, y_2) is given by:

$$\begin{cases} x_2 = x_1 + z_{max}\cos\theta + \delta_k \\ x_2 = 2_1 + z_{max}\sin\theta + \delta_k \end{cases} \quad . \tag{12}$$

θ is the current orientation of the particle and δ_k the increment of each laser beam. The true distance is then given by the mathematical intersection of this line with the closest wall in the map or z_{max} if there is no intersection.

*4) **Resampling**:* For the resampling step, selective resampling was performed. This approach is a variation of the resampling step aimed at improving the efficiency of the algorithm by only resampling a subset of the particles. More concretely, if the effective number of particles n_{eff} is more than half the total number of particles, no resampling is performed. n_{eff} is given by equation 13.

$$n_{eff} = \frac{1}{\sum_{i=1}^{N} w_i^2} \tag{13}$$

*5) **Kidnapping**:* Kidnapping happens when the real-world position of the robot changes and the odometry does not reflect that, for example, if the robot was picked up and placed in another position not following the motion model and causing the algorithm to lose track of the position. In order to recover from this, a simple condition was implemented: if the mean of the non-normalized weights is below a given threshold twice in a row, the new number of particles increases by a factor of 1.2, and 80% of this new set is uniformly distributed throughout the map. This way, some of the new particles will be a better estimate of the new position and the algorithm will start to converge to a new position. This technique not only allows to recover from kidnapping, but also from a wrong convergence, which happens more regularly when the number of particles is low.

V. Data Acquisition

Experimental data was collected with a Pioneer 3-DX robot, a medium-sized two-wheel robot commonly used for research purposes equipped with a Raspberry Pi containing Ubuntu 18.04LTS and ROS 1 Melodic, and with the P2OS ROS package. A Hokuyo URG-04-LX-UG01 2D laser rangefinder is also used, which delivers distance readings in a 240-degree field of view. It features a high scan rate of 10 Hz and a range of up to 5.6 meters. The experiments were performed on the 5th floor of the North Tower at Instituto Superior Técnico.

A. Micro Simulator (Synthetic data)

In order to have an easier and more effective way to test and debug the algorithm, a micro simulator was created. With it, synthetic data and a map were created.

The map created was a simplified version of the area where the experiment happened, in which the black lines represent the walls, the grey zones are areas not accessible to the robot, and the white zones are areas accessible to the robot. The map can be seen in Fig. 1. It should be noted that the map is not up to scale.

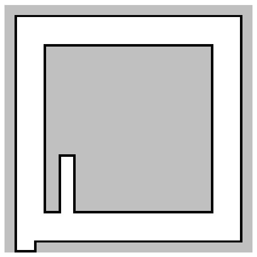
Fig. 1. Map created for the micro simulator.

As for the synthetic data, it was created using the motion and measurement models explained in III-A and III-B. The motion model is used to simulate the pose of the robot, while the measurement model is used to simulate the measurements of the robot.

B. Real data

The Robot Operating System (ROS) is a set of software libraries and tools that help build robot applications. In this implementation of the MCL algorithm, ROS was used to access and record different types of data through the use of topics and compute the reference map with the data recorded.

*1) **Odometry**:* In order to implement the algorithm, the pose is required and obtained by accessing the data from the odometry sensor. ROS does this by subscribing to the "/pose" topic through a script. This topic provides the pose of the robot in each instant relative to the position where the robot was turned on. The position is represented in meters, while the orientation (yaw) is represented in quaternions. However, in the implementation, the yaw is converted to radians to allow for easier mathematical calculations.

For the algorithm, only the current and previous poses are required and saved.

*2) **Measurements**:* Similar to obtaining the odometry data, the algorithm implementation requires the robot's observations, which can be obtained by accessing data from the Hokuyo laser range finder. ROS does this by subscribing to the "/scan" topic through a script. This topic provides an array of distance measurements in meters. Additionally, it also provides the minimum and maximum ranges that can be detected and the yaw increment between each laser scan.

For the algorithm, only 26 out of 726 measurements were kept, approximately one each $10°$. This selection was made because of the heavy computational burden that would result from processing 726 measurements since the measurement model implementation is the heaviest in the algorithm. However, despite the reduction, the results were not significantly affected by this.

*3) **Reference map and localization**:* A reference map of the robot is needed to implement the algorithm. The map can be obtained by using the gmapping package on the recorded data from the robot through the use of a launch file. The map obtained is represented in Fig. 2.

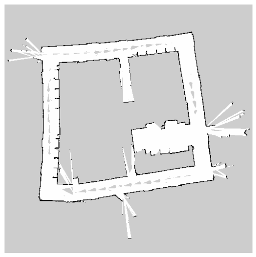
Fig. 2. Map obtained using gmapping.

As for the comparison of results, the reference localization of the robot is needed. This can be obtained using the AMCL ROS package which stands for Adaptive Monte Carlo Localization through the use of a launch file.

VI. Results

In order to test the accuracy, robustness, and consistency of the algorithm, several different tests were performed using the synthetic data and the real data. The results were evaluated using the root mean square error (RMSE) metric that, considering a predicted position (x, y) and a reference position (x_{ref}, y_{ref}), is defined in equation 14.

$$\text{RMSE} = \sqrt{(x - x_{ref})^2 + (y - y_{ref})^2} \quad . \qquad (14)$$

A. *Synthetic data*

For the synthetic data, two different tests were considered: one where there is no kidnapping and another where kidnapping occurs. In these tests, the correct position of the trajectories was used as a reference for the RMSE calculations. It should be noted that, again, the map is not up to scale. However, the positions were scaled in a ratio of 50:15, where 50 is the length of the map and 15 meters is the size of the hall.

*1) **Trajectory without kidnapping**:* For the trajectory illustrated in Fig. 3, six tests were performed in each of the following conditions: using a Gaussian distribution around the starting position to initialize the particles for 400, 800, and 1600 particles and using a uniform distribution to initialize the particles considering the number of particles as 400, 800, or 1600. It should be noted that the total number of tests for this trajectory was 36 and all data sets differ from each other given that in each test the particles initialized by the distributions were in different positions. The results are shown in Fig. 4 and Fig. 5.

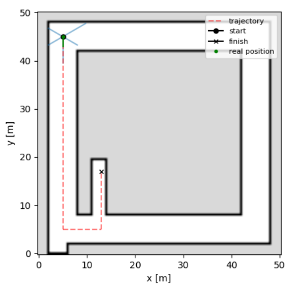

Fig. 3. Trajectory of the robot without kidnapping.

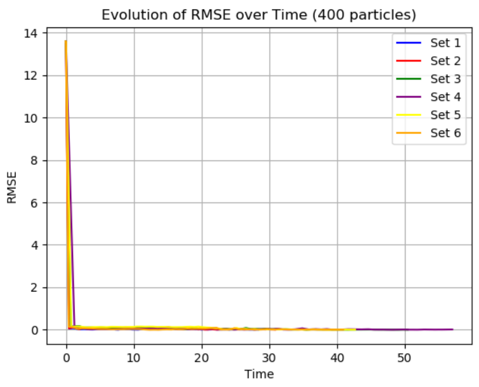

Fig. 4. RMSE of the algorithm prediction compared to the robot position using a Gaussian distribution for 400 particles.

Analyzing Fig. 4, it can be seen that the particles converge very quickly to the correct position, having a very small error in the order of 10^{-2}. This is expected given that they are initialized around the initial position of the robot. In this situation, only the RMSE of the 400 particles test is shown

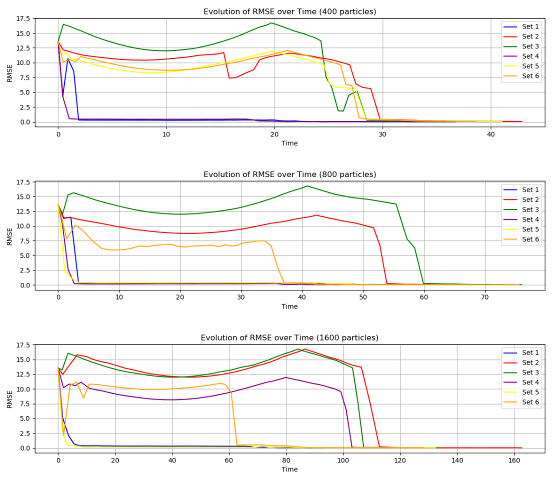

Fig. 5. RMSE of the algorithm prediction compared to the robot position using a Uniform distribution for 400, 800, and 1600 particles, respectively.

because, given that there is no kidnapping, the results for bigger numbers of particles are similar.

However, the initial position might not be known. In this case, a uniform distribution has to be used. Analyzing Fig. 5, it can be seen that, although, in two out of the six tests, the particles converge to the correct position very quickly, in the other tests they converge to the wrong position. This error is due to the symmetry of the halls on the map. Despite this, the system is robust enough to still correct its prediction and converge to the correct position. This happens when the average of non-normalized likelihoods reaches a value below a given threshold, causing the number of particles to increase and distributing 80% of them uniformly across the map, as explained in section IV-5. Besides this, it can also be seen that the more particles there are, the more time it takes to converge and reach the end of the path, which is to be expected given that more computational power is required to compute more particles.

*2) **Trajectory with kidnapping**:* For the trajectory illustrated in Fig. 6, six tests were performed in the same conditions mentioned in VI-A1. The total number of tests for this trajectory is 36 and the results are shown in Fig. 7 and Fig. 8.

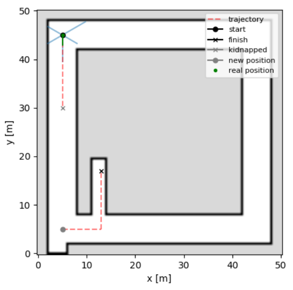

Fig. 6. Trajectory of the robot with kidnapping.

Analyzing Fig. 7, it can be seen that the particles initially converge very quickly to the correct position. However, at

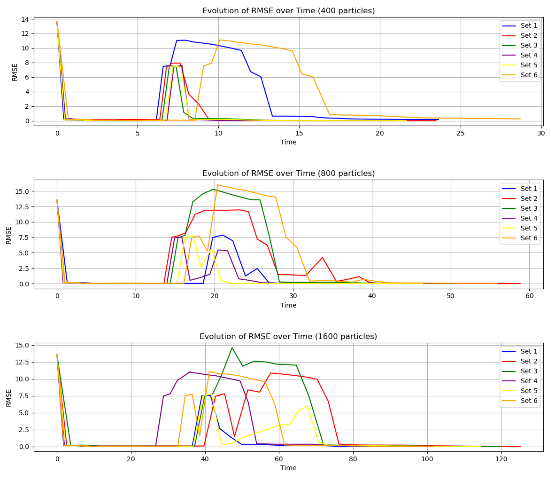

Fig. 7. RMSE of the algorithm prediction compared to the robot position using a Gaussian distribution for 400, 800, and 1600 particles, respectively.

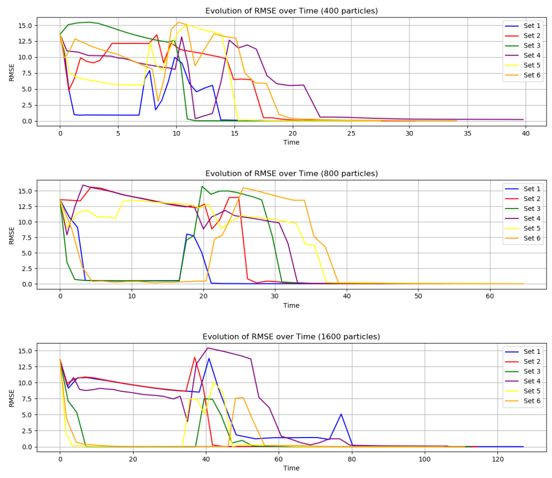

Fig. 8. RMSE of the algorithm prediction compared to the robot position using a Uniform distribution for 400, 800, and 1600 particles, respectively.

some point, there is a peak in the RMSE attributed to the kidnapping. After the robot has been kidnapped, the average of non-normalized likelihoods reaches a value below a given threshold, causing the number of particles to increase and distributing 80% of them uniformly across the map, as explained in section IV-5. This allows the algorithm to recover and converge to the correct position again. Furthermore, given that the distribution is uniform, a bigger number of particles will take more time to converge, as was explained in VI-A1.

Considering a uniform distribution, it can be concluded by observing Fig. 8 that the algorithm does not initially converge to the correct position for 400 particles in any test. However, such does not happen when using 800 and 1600 particles. This can be explained due to the fact that there are too few particles compared to the size of the map. This leads to having very few particles around the correct position for it to be considered. Therefore, more than 400 particles should be computed in this situation.

B. *Real data*

For the real data, the following tests were considered: one where there is no kidnapping and another where kidnapping occurs. A similar trajectory to the one done in the micro-simulator was followed, labeled as *Reference* in Fig. 9 and Fig. 11. It should be noted that the reference trajectory was only used for visualization purposes and actually not computed in the algorithm. Therefore, in order to compute the RMSE in real data, the AMCL estimate was used as a reference.

The use of real data showed that the algorithm was slow relative to the frequency of the messages it received, leading to bigger jumps in information between iterations, and making it harder to correctly converge with a low number of particles. These tests also took longer than the ones from the section VI-A, prompting a reduction in the number of tests per condition.

*1) **Trajectory without kidnapping**:* For the trajectory illustrated in Fig. 9, three tests were performed in each of the following conditions: using a uniform distribution to initialize the particles considering the number of particles as 400 or 1600. The total number of tests for this trajectory was only 6 because of the time consumed by each test. The results are shown in Fig. 10.

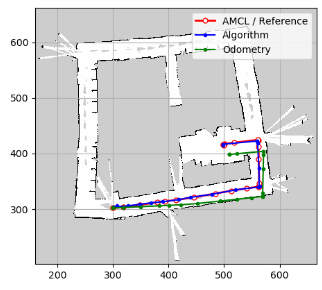

Fig. 9. Trajectory of the robot without kidnapping.

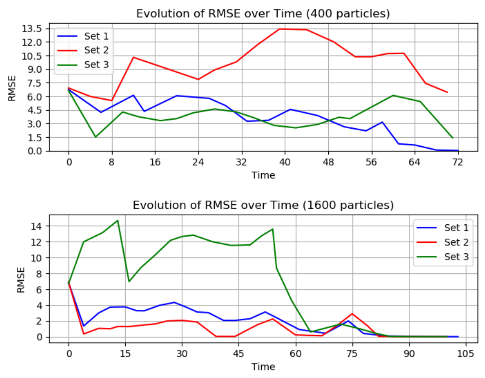

Fig. 10. RMSE of the algorithm prediction compared to the robot position using a Uniform distribution for 400 and 1600 particles.

Comparing the results with the ones in Fig. 5 for 400 particles, one can see that, with the use of real data, the algorithm takes longer to converge to the correct position, and, in some cases, it does not predict it, while with synthetic data it always converged to the right position. This can be explained by the low number of particles compared to the size of the real map. Also, because the particles are randomly initialized, there is no guarantee that any of them is a good estimate for the position of the robot. Even with the robustness added by the kidnapping condition, the same can happen.

As for the comparison of results for 1600 particles, the algorithm converged to the right position in all three tests of the real data. However, while sets 1 and 2 had a good initialization, converging to the correct position and leading to accurate results, set 3 initially converged to the wrong position but, due to the robustness added by the kidnapping condition, corrected itself and converged to the correct position. Furthermore, it can also be seen that, even when the algorithm estimate is initially correct in sets 1 and 2, there is still an error associated with the noise in the real data, leading to a varying RMSE with values between 0 and 4.

2) ***Trajectory with kidnapping***: For the trajectory illustrated in Fig. 11, three tests were performed in the same conditions mentioned in VI-B1. The total number of tests was 6, once again because of the time each test took. The results are shown in Fig. 12.

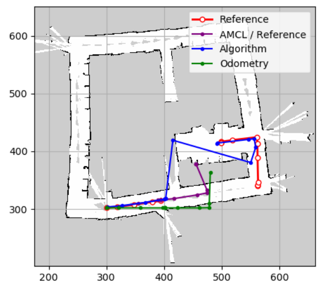

Fig. 11. Trajectory of the robot with kidnapping.

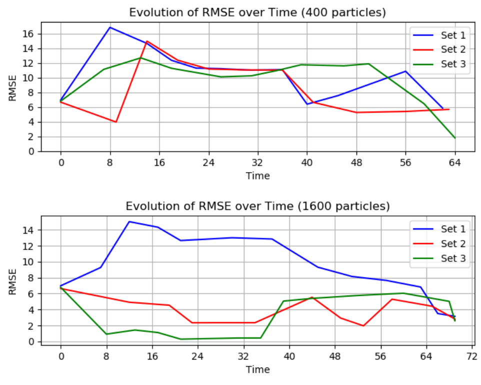

Fig. 12. RMSE of the algorithm prediction compared to the robot position using a Uniform distribution for 400 and 1600 particles.

While in the kidnapping test of the micro-simulator, all tests converge to the reference position considered in the RMSE, the same did not happen with real data. As seen in Fig. 12, multiple tests converge to an RMSE value between 2 and 4, meaning there is a constant deviation between the algorithm estimate and the AMCL estimate. Analyzing Fig. 11, it is concluded that this deviation occurs because the AMCL does not recognize the kidnapping and its final estimate is in a position outside the map, while the algorithm's estimate is in the correct final position relative to the reference trajectory. Therefore, the AMCL failed to determine the correct position of the robot after the kidnapping and hence the convergence to a deviated RMSE value.

This error can be explained by the combination of changing the orientation of the robot with the kidnapping and the small time interval given between the kidnapping and the end of the trajectory, not giving enough time for AMCL to converge to the correct estimate.

With this information, it is possible to conclude that the algorithm converged to the correct position for all the tests with 1600 particles and in set 3 with 400 particles, i.e. it is robust to the kidnapping problem even with real data.

VII. Conclusion

With regards to the micro-simulation, the algorithm correctly converged despite the conditions of the test, demonstrating that the algorithm was accurate and robust to cases where the robot was kidnapped or the initial distribution started converging incorrectly, the main objective of this implementation. On the other hand, the real data tests showed that the algorithm could have a better performance time-wise, although the results remained as expected even in the kidnapping tests where the AMCL failed for the specific data. If the algorithm also failed, it could be concluded that it was a problem with the data, however, since the algorithm correctly predicted the robot's position, this wrong result from the AMCL was due to the combination of changing the orientation of the robot in the kidnapping and the small time given for AMCL to converge after the kidnap.

The results obtained indicate that the implementation of the MCL algorithm was successful, since it is capable of solving the kidnapping problem, but could still use some performance and robustness improvements, such as a more robust measurement model with distributions referring to unexpected objects and failures and considering all the beams in each measurement. To implement such changes, the performance would need to improve drastically in order to keep the algorithm in a usable state.

References

[1] S. Thrun, W. Burgard, and D. Fox, ”Probabilistic Robots”, 1st ed. Cambridge, MA: MIT Press, 2005.

[2] J. Elfring, E. Torta, and R. van de Molengraft, ”Particle Filters: A Hands-On Tutorial”, *Sensors* (Basel), vol. 21, no. 2, p. 438, Jan. 2021. doi: 10.3390/s21020438.