Monte Carlo Localization Algorithm for TurtleBot3

António Morais-102643, Catarina Caramalho-102644, Miguel Gonçalves-102539, Neelam Visueshcumar-103376
Instituto Superior Técnico
Lisbon, Portugal
Email: antonio.v.morais.c@tecnico.ulisboa.pt, catarina.caramalho@tecnico.ulisboa.pt,
miguel.angelico@tecnico.ulisboa.pt, neelamjaiesh@tecnico.ulisboa.pt

***Abstract*— This study addresses the localization problem, a crucial component of autonomous navigation. The primary objective is to enable a robot to determine its pose (i.e. position and orientation) in real time within an indoor environment. This is accomplished using a 2D Light Detection and Ranging (LiDAR) sensor and a known map as input data for the Monte Carlo localization (MCL) method based on a particle filter.**

The development can be divided into four main steps: i) particle set creation and initialization; ii) predicting the position of the particles based on odometry; iii) updating the particle set by comparing the position of the particles with laser measurements and iv) resampling the particles based on the regions with the highest probability.

Experimental validation involves testing the algorithm with simulated data on a microsimulator and with real data from a TurtleBot3. Finally, the algorithm is compared with the groundtruth of AMCL (adaptive Monte Carlo localization) package from ROS (Robot Operating System).

***Index Terms*— Monte Carlo localization, particle filter, odometry**

I. Introduction

In mobile robots or autonomous systems, one of the greatest challenges lies in navigation, which consists in different tasks such as mapping, localization and path planning avoiding obstacles [1].

This study focuses on localization. The ability that a robot has to determine its current pose based on its previous pose and the distance traveled is known as odometry, and it accumulates errors over time, becoming unsuitable for accurate self-localization by the robot.

To ensure robustness and overcome the challenges already discussed, the selected method was the Monte Carlo localization (MCL) algorithm, which is based on a particle filter and it is renowned for its effectiveness [2]. This algorithm uses a set of particles representing hypothetical poses of the robot, each assigned weights corresponding to their likelihood of matching the actual robot pose. This approach is intended to facilitate localization in both static and dynamic environments, capable of detecting estimation failures and responding to potential challenges like the robot "kidnapping", where it must re-localize itself.

The difference between the particle filter and MCL resides in the fact that the last incorporates a motion and measurement models, each adapted to the specific requirements of the localization task in hand.

The experimental methodology involves testing the developed method on both Gazebo simulator and a real TurtleBot3.

II. Problem Description

The localization problem involves determining the pose of a robot, comprising its position and orientation, within a pre-existing map. The TurtleBot3 robot utilized in this study is equipped with four sensors: a camera, an inertial measurement unit (IMU), a laser distance sensor (LDS), and wheel encoders. Within the known map, the robot's pose can be regarded as a point within the 2D environment. It is assumed that velocities (linear and angular) remain constant between two consecutive timestamps, denoted as $t-1$ and t, with a significantly short time interval separating them.

A. Sensors

This project uses odometry data from velocities to estimate the pose of the robot at a certain timestamp.

1) IMU: Using a gyroscope and accelerometer it senses robot's orientation, angular velocity, and linear acceleration, aiding in pose estimation by integrating these measurements over time. *2) Wheel encoders:* Measure wheel rotation, enabling estimation of robot's distance traveled and changes in orientation crucial for odometry. *3) LDS:* With a scanning area of $360°$ and a resolution of $1°$ it can detect objects between 12 cm to 3.5 m that refine odometry data.

B. System Modulation

1) Motion Model: The TurtleBot3 executes 2D navigation, requiring the consideration of its positional coordinates in the x and y axes, as well as rotation about the z-axis (yaw), to model its motion.

Let x_t, y_t, and θ_t denote the x and y coordinates, and the orientation angle of the robot along the z-axis at timestamp t, respectively. On the other side let x_{t-1}, y_{t-1} and θ_{t-1} denote the mentioned but for a previous pose in time. Consider v_t as the linear velocity (positive indicating forward motion), w_t as the angular velocity (positive inducing counterclockwise rotation).

Having made the assumption of constant velocities in II, the accelerations are equal to zero. Thus, one can define the state variables and the robot's initial pose as $[x_t\, y_t\, \theta_t]^T$ and describe the system with the motion equations:

$$\begin{cases} \frac{dx_t}{dt} = v_t \cos(\theta_t) \\ \frac{dy_t}{dt} = v_t \sin(\theta_t) \\ \frac{d\theta_t}{dt} = \omega_t \end{cases} \tag{1}$$

Furthermore, one can relate the pose at a given time to the pose at a previous timestamp, where the new pose is described as the previous pose plus the motion information (1). Hence we obtain the kinetic motion model (2) after integrating (1) with respect to time.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + v_t \cos(\theta_t)\Delta t \\ y_{t-1} + v_t \sin(\theta_t)\Delta t \\ \theta_{t-1} + \omega_t \Delta t \end{bmatrix} \tag{2}$$

Any arbitrary motion $[\Delta x, \Delta y]^T$ can be performed as a rotation followed by a translation [3, p. 2]. Observing Fig. 1, the robot initially has a pose $[x_{t-1}\, y_{t-1}\, \theta_{t-1}]^T$, then rotates to orientation θ_t and subsequently translates by $\rho = \sqrt{\Delta x_t^2 + \Delta y_t^2}$ meters to reach a final pose $[x_t\, y_t\, \theta_t]^T$. Therefore, the noise model in III is applied separately to each type of motion, as they are assumed to be independent.

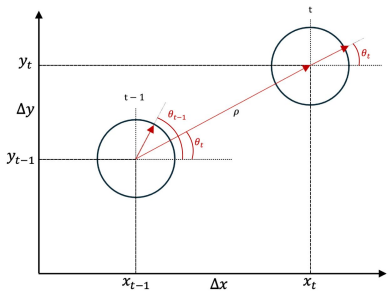

Fig. 1: Motion Model (Adapted from [3, p. 4]).

2) LDS: The sensor utilized is a Light Detection and Ranging (LiDAR), which emits infrared light beams that bounce back from objects to the sensor, determining the time from when the beam is emitted to its return. By knowing the velocity of light and the time of the beam's return, the distance from the sensor to an object can be approximated as:

$$distance = (1/2) \cdot (velocity \times time) \tag{3}$$

III. Methods and Algorithms

Particles represent the posterior distribution through a set of weighted samples. These particles can model arbitrary distributions, making them suitable for scenarios where measurement or prediction distributions are non-Gaussian. The localization process can be summarized as follows:

A. Particle Generation

A number of particles are generated to represent the robot, each with a random pose uniformly distributed within the map, reflecting the initial lack of knowledge about the robot's exact position. Consequently, each pose where the particle is within the map has an equal probability of representing the robot's location. The samples are then updated through two stages: Prediction and Update.

B. Prediction

Motion Model plays the role of the state transition model. This model is the conditional density $p(x_t \mid u_t, x_{t-1})$. Here, x_t and x_{t-1} are both robot poses, and u_t is a motion command provided by a robot's odometry [4, pp. 93-94]. This model describes the posterior distribution over kinematics states that a robot assumes when executing u_t when its pose is x_{t-1}. The motion model serves to estimate the posterior distribution over possible future robot poses x_t given its current x_{t-1} pose and the odometry u_t it receives. Additionally, a noise model is applied independently to simulate the inherent errors that occur during the robot's motion.

C. Update

Given the sensor readings from the robot and measurements from the known map, particles are weighted based on the likelihood of these observations. The laser values provided by the ranges vector z_t indicate the distance from the sensor to obstacles or objects in the environment at specific angles or directions, with 360 positions corresponding to each degree of observation. Values outside the sensor's limits, such as ∞ or zero, are adjusted to fit within the sensor's range maximum and minimum.

The predicted ranges vector z_t^* is generated by raycasting within the sensor model to simulate what the robot is observing. Each range value is evaluated at each angle. The sensor model incorporates the predictions from the motion model and corrects them using the sensor readings. In this context, local measurement noise is considered, represented by a Gaussian distribution. The goal is to assess the likelihood of the robot being at different poses based on the correspondence between simulated and actual measurements.

This comparison refines the probability distribution, increasing the accuracy of the robot's estimated pose. Consequently, particles are weighted based on the degree of match between their simulated measurements and the actual measurements. Poses that produce measurements closer to the real ones are assigned with higher weights, while poses with significant discrepancies are assigned with lower weights. The measurement model is given by the distribution of z_t (ranges measured) given x_t and m (map) with density:

$$p(z_t \mid x_t, m) = \frac{1}{\sqrt{(2\pi)\,|R_t|}} e^{-\frac{1}{2}(z_t - z_t^*)^T R_t^{-1} (z_t - z_t^*)} \tag{4}$$

where z_t^* denotes the vector of ranges predicted via raycasting. This distribution is characterized by its mean, z_t^*, and variance matrix, R_t. Utilizing this density, the particle weight is computed.

D. Resample

Resampling results in the updated belief of the system state. It ensures that particles stay in the meaningful areas of the state space. The choice of when to resample is intricate and requires practical experience: Resampling too often increases the risk of losing diversity. If one samples too infrequently, many samples might be wasted in regions of low probability.

A standard approach to determine whether or not resampling should be performed is reducing the sampling error with low variance sampling. The basic idea is that instead of selecting samples independently of each other, the selection involves a sequential stochastic process [4, pp. 86-87]. Resampling helps in the selection of particles according to their weights, thereby effectively redistributing the weight to maintain a diverse set of particles that more accurately represents the posterior distribution. By focusing computational resources on particles with higher weights, which are more likely to approximate the true state of the system, the algorithm can converge more swiftly and precisely towards the true location.

MCL in its present form solves the global localization problem but cannot recover from robot kidnapping. Fortunately, this problem can be solved by adding random particles to the particle sets, assuming that the robot might get kidnapped with a small probability. Even if the robot does not get kidnapped, the random particles add an additional level of robustness [4, pp. 204-205]. The Augmented MCL Algorithm 1 was used, it monitors the probability of sensor measurements $p\left(z_t \mid z_{t-1}, u_t, m\right)$ and relates it to the average measurement probability w_{avg}. As the amount of sensor noise might be unnaturally high, or the particles may still be spread out during a global localization phase, it is a good idea to maintain a short-term average of the measurement likelihood (w_{fast}), and relate it to the long-term average (w_{slow}) when determining the number of random samples. This algorithm requires $0 \leq \alpha_{slow} \leq \alpha_{fast}$. The parameters α_{slow} and α_{fast} are decay rates for the exponential filters that estimate the long-term and short-term averages, respectively. During the resampling process a random sample is added with probability $\max\left(0.0, 1.0 - w_{fast}/w_{slow}\right)$. If the short-term likelihood is better or equal to the long-term likelihood, no random sample is added. However, if the short-term likelihood is worse than the long-term, random samples are added in proportion to the quotient of these values [4, pp. 206-207].

As mentioned previously, resampling poses a risk as it may lead to the loss of critical samples. Therefore, resampling should be undertaken when a minority of particles carry the entirety of the weight. Considering

$$n_{eff} = \left(\sum_{m=1}^{M} \left(w_t^m\right)^2 \right)^{-1} \tag{5}$$

an empirical measure of how well the goal distribution is approximated by samples drawn from the proposal. Two distinct scenarios delineate the necessity for resampling: when all weights are equal (zero estimator variance), and when a single particle bears the entirety of the weight. In the first scenario, where $n_{eff} = M$ (no resampling), the target distribution closely aligns with the proposal distribution, while in the latter scenario $n_{eff} = 1$ (resampling) occurs. Consequently, particles undergo resampling solely when n_{eff} drops below a given threshold $\frac{M}{2}$. The addition of random particles only occurs when both conditions $w_{fast} < w_{slow}$ and $n_{eff} < \frac{M}{2}$ are satisfied.

Algorithm 1 Augmented MCL Algorithm

static w_{slow}, w_{fast}
$\overline{\chi_t} = \chi = \emptyset$
for $m = 1$ to M **do**
  $x_t^{[m]}$ = sample_motion_model(u_t,$x_{t-1}^{[m]}$)
  $w_t^{[m]}$ = measurement_model($z_t, x_t^{[m]}, m$)
  $\overline{\chi_t} = \overline{\chi_t} + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
end for
$w_{slow} = w_{slow} + \alpha_{slow}\left(w_{avg} - w_{slow}\right)$
$w_{fast} = w_{fast} + \alpha_{fast}\left(w_{avg} - w_{fast}\right)$
if $n_{eff} < M/2$ **then**
  for $m = 1$ to M **do**
    with probability $\max(0.0, 1.0 - w_{fast}/w_{slow})$ **do**
      add random pose to χ_t
    else
      draw $i \in \{1, ..., N\}$ with probability $\propto w_t^{[i]}$
      add $x_t^{[i]}$ to χ_t
    end with
  end for
end if
return χ_t

IV. Implementation

A. Error Analysis

The MCL algorithm will be evaluated using three types of tests: (1) synthetic data obtained from the micro-simulator, (2) simulated data obtained from Gazebo, and (3) real data obtained from TurtleBot3 on the fifth floor of the North Tower at Instituto Superior Técnico (IST). A suitable method for evaluating the algorithm is the root mean square error (RMSE), which serves as an excellent measure of error. In this evaluation, the Adaptive Monte Carlo Localization (AMCL[1]) package provided by ROS will be used as the ground truth. AMCL is a robust method for localization in mobile robotics, known for its accuracy and reliability in dynamic environments, making it an ideal benchmark to evaluate the performance of the developed algorithm.

It should be noted that AMCL will only be used for test types (2) and (3), as in test type (1) the robot's position is already known.

To calculate the RMSE, the following expression will be used:

$$\text{RMSE} = \sqrt{\left(\overline{x} - \overline{x}_{\text{AMCL}}\right)^2 + \left(\overline{y} - \overline{y}_{\text{AMCL}}\right)^2} \tag{6}$$

$\overline{x}$ and $\overline{y}$ represent the estimated pose, obtained through the weighted average of the top 1% heaviest particles. $\overline{x}_{\text{AMCL}}$ and $\overline{y}_{\text{AMCL}}$ represent the pose obtained by AMCL. It should be noted that in the test type (1) $\overline{x}_{\text{AMCL}}$ and $\overline{y}_{\text{AMCL}}$ will be obtained directly from the `/robotpose` topic published by the micro-simulator node.

[1] http://wiki.ros.org/amcl

The standard error of the mean (SEM) (9) is a way to know how close the average of the sample is to the average of the whole group. Then one can know how sure can be about the average from the sample. The primary purpose of SEM is to construct confidence intervals for the mean, which represent the range of values likely to contain the true RMSE value. SEM essentially indicates the expected fluctuation of the sample mean from the actual $\text{RMSE}_{\text{MEAN}}$ (7) if the process was repeated multiple times. To test the algorithm using SEM, 10 tests were conducted. The procedure begins with the computation of the $\text{RMSE}_{\text{MEAN}}$ of the 10 trials for each instant.

$$\text{RMSE}_{\text{MEAN}} = \frac{\sum_{i=1}^{10} \text{RMSE}_i}{10} \tag{7}$$

$$\text{SD} = \sqrt{\frac{\sum_{i=1}^{10} \left(\text{RMSE}_i - \text{RMSE}_{\text{MEAN}}\right)^2}{10-1}} \tag{8}$$

$$\text{SEM} = \frac{\text{SD}}{\sqrt{10}} \tag{9}$$

B. Ros Nodes and topics

In order to have a graphical representation of the MCL algorithm in operation, a node responsible for this function was created, called `Graphs Node`. The diagram in Fig.2 aims to summarize how data flows between the nodes. The main node `MCL Node` is responsible for the full implementation of the algorithm. It subscribes to the `/odometry` topic (for the motion model) and the `/scan` topic (for the measurement model), both published by the TurtleBot3.

The `Graphs Node` must subscribe to the `/particles` and `/pose` topics, both published by the `MCL Node`. The `/particles` topic contains the positions of all particles, while the `/pose` topic provides the position estimate calculated through the weighted average of the particles. The main objective is to compare the pose estimate with the one provided by AMCL. Consequently, the `Graphs Node` also subscribes to the `/amclpose` topic.

Fig. 2 represents the data flow in the ROS environment for tests of type (2) and (3). For type (1) tests, the `Graphs Node` subscribes to `/robotpose` instead of `/amclpose`, because the robot's position is known *a priori*.

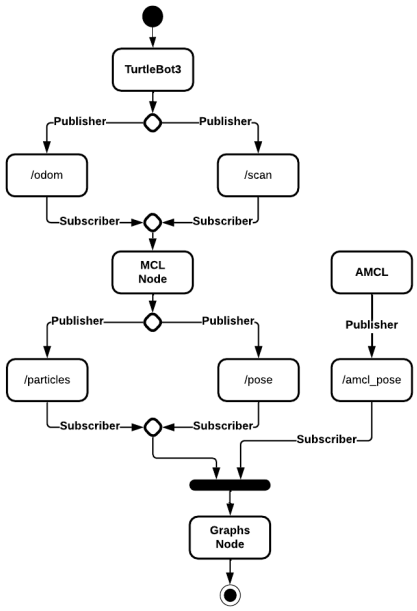

Fig. 2: Ros Nodes and topics

V. Simulation Results

As mentioned earlier, the developed algorithm will be tested for two different approaches: A) synthetic data on micro-simulator and B) simulated data from Gazebo.

A. Micro-simulator

For the micro-simulator, the assumed values were the ones that closely match reality: $range_{\text{min}} = 0.12$, $range_{\text{max}} = 3.5$, and $\Delta t = 0.2$ s. A fixed value of 0.4 m/s was used for the linear velocity, while the angular velocity was represented as a vector outlining the intended path. The ranges values that are obtained from the rosbag, had to be created artificially. Therefore, a ranges vector was generated, it matched the ranges predicted by raycasting but included added noise to simulate the real vector ranges.

It is essential to ensure that the algorithm performs effectively with different numbers of particles and reacts efficiently to instances of robot kidnapping.

1) For 100 particles

The first test (Fig. 3) was conducted on a map with irregular boundaries. This type of map highlights the differences in what a robot can see from various positions, helping to reduce ambiguity. Unfortunately, the particles did not converge to the robot's pose. However, these particles converged to a location on the map that is very similar to what the robot could see from its pose.

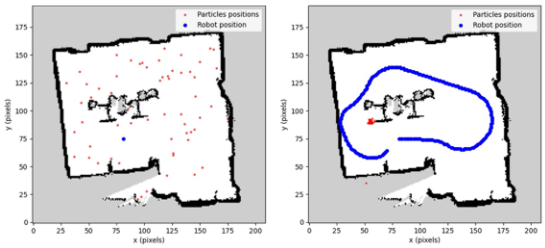

Fig. 3: Initialization and unsuccessful convergence

2) For 1000 particles

In this test (Fig. 4), the particles converged successfully to the robot's pose. The convergence of the particles results from their wide distribution across the map, efficiently covering the entire area. In contrast, when only 100 particles were used, they could not spread sufficiently to cover the wide area, leading to a failure to converge on the robot's pose.

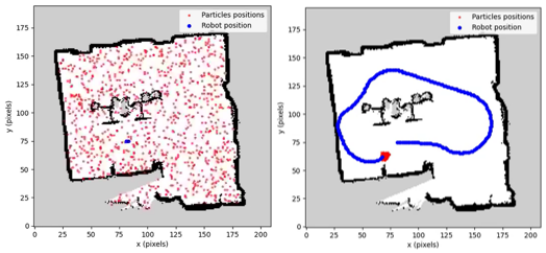

Fig. 4: Initialization and successfully converged

3) Kidnapping

Kidnapping occurs when the robot is suddenly moved to an unknown location without being aware of the relocation. The developed algorithm reacts efficiently to this phenomenon (Fig. 5) since w_{fast} and w_{slow} are being monitored as explained in chapter III-D. This test was ran for 1000 particles.

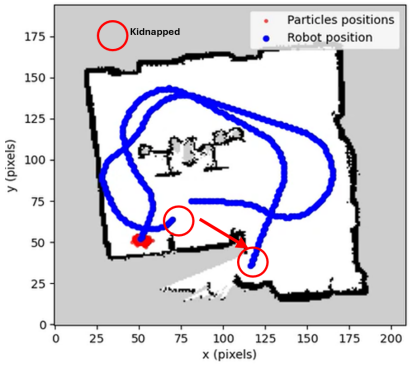

Fig. 5: Kidnapping tested for 1000 particles

4) RMSE results

The algorithm was tested using the RMSE metric for different variance values, each tested 10 times with the same α_{slow} and α_{fast} parameters, to analyze the convergence frequency. The experiments were conducted with 1000 particles using both micro-simulator and real data, as it was observed that convergence is more likely under these conditions (convergence is assumed when $\text{RMSE}_{\min} < 5$ cm for the micro-simulator and $\text{RMSE}_{\min} < 30$ cm for real data). The results are presented in Tables I and II obtained for the micro-simulator and real data, respectively.

TABLE I: Variance influence on convergence ($\alpha_{slow} = 0.01$ and $\alpha_{fast} = 0.1$ for micro-simulator data)

σ^2_{hit}	20	40	60	80	100	120
Number of convergences (out of 10)	6	8	10	5	4	4
RMSE (min error)	0.080	0.057	0.097	0.128	0.082	0.052

Upon analyzing the results, it was concluded that a variance value of 60 is the optimal choice for the micro-simulator, as it consistently achieves convergence with a non-significant $\text{RMSE}_{\min}$ error. These tests were conducted to validate the correct implementation of the algorithm. For the micro-simulator, the $\text{RMSE}_{\min}$ was found to be very low, indicating that the algorithm can accurately localize the robot's position.

TABLE II: Variance influence on convergence ($\alpha_{slow} = 0.01$ and $\alpha_{fast} = 0.1$ for real data)

σ^2_{hit}	20	40	60	80	100	120
Number of convergences (out of 10)	3	8	7	6	10	10
RMSE (min error)	0.355	0.062	0.136	0.273	0.018	0.015

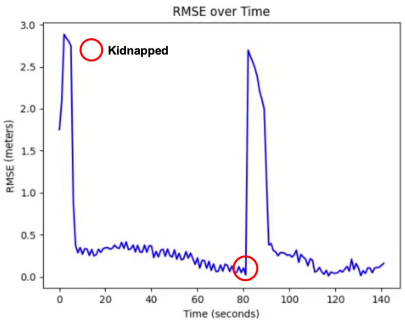

Fig. 6: RMSE with kidnapping over time

The graph (Fig. 6) shows the RMSE over time. Initially, the RMSE values are high due to the particles being widely dispersed across the map during the initial phase. As time progresses, the particles begin to converge, resulting in a significant decrease in RMSE values. At approximately 80 seconds, the robot is kidnapped, leading to a sudden increase in RMSE as the particles lose track of the robot's pose. Subsequently, the particles start to reconverge to the robot's new pose, causing the RMSE to decrease again.

A series of tests were conducted to investigate the influence of the parameters α_{slow} and α_{fast}. As expected, when α_{fast} isn't much greater than α_{slow} ($\alpha_{slow} = 0.2$ and $\alpha_{fast} = 1$), random samples are added more frequently, whereas when the difference between α_{fast} and α_{slow} is greater ($\alpha_{slow} = 0.001$ and $\alpha_{fast} = 0.8$), random samples aren't added so often. Thus, during the kidnapping in the 1st case, the algorithm reacts faster than in the 2nd case. This is explained by the changes that occur during the robot kidnapping, as the w_{avg} drops, w_{fast} and w_{slow} start decreasing, but random samples are only added when $w_{fast} < w_{slow}$.

B. Gazebo

The Gazebo environment of the existing world was used to run the same tests. The results obtain from gazebo were similar to the graphs obtained in micro-simulator.

VI. Experimental Results

To test the algorithm in a real-world scenario, real data obtained with the TurtleBot3 on the fifth floor of the North

Tower at IST. The corridor on the floor was utilized to record the acquired rosbags. Subsequently, a partial map of the corridor was generated via the gmapping[2] package provided by ROS, due to the Wi-Fi limitations on the floor.

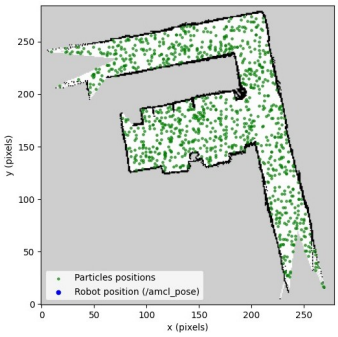

Fig. 7: Map used for real data with initialised particles

It can be seen from Table II that higher values of σ^2_{hit} result in convergence for most of the time (10 out of 10 convergences) and with lower $\text{RMSE}_{\min}$ values. Due to the fact that higher values of σ^2_{hit} rely more on odometry rather than laser values and resamples are less often, the particles converged in every test for higher values of σ^2_{hit}. On the other hand, a lower value of σ^2_{hit} relies more on laser values rather than odometry and resamples occur frequently. Despite of the fact that the result of a greater σ^2_{hit} is a guarantee of convergence, it was seen while running the tests that lower values of σ^2_{hit} took less time to converge, even though particles sometimes did not follow correctly the robot's pose while greater values of σ^2_{hit} took more time, but eventually did follow the robot's pose. In conclusion, for real data the best value of σ^2_{hit} is dependent on the data collected and the map used, as well as the laser specifications. Therefore this value should be picked by fine tuning (trial and error).

It can be concluded that for real data, excessive reliance on laser data is not advisable, as these measurements are more susceptible to inaccuracies and noise. Consequently, higher σ^2_{hit} values should be employed to decrease the dependence on laser data and enhance the reliability of the odometry. In the case of simulated data, the laser data is generated with Gaussian noise, then the data won't be as noisy as the real data, in fact, it will be more precise. Thus, a lower σ^2_{hit} value ensures the proper functioning of the algorithm. In other words, greater confidence can be placed in the data provided by the laser.

For the best variance ($\sigma^2_{hit} = 60$) obtained for the micro-simulator, the SEM was calculated and it was observed through the error bars that $\text{RMSE}_{\text{MEAN}}$ does not fluctuate significantly. Upon examining the $\text{RMSE}_{\text{MEAN}}$ for the 10 trials, it was evident that despite different initializations and resamples over time across the 10 trials, the RMSE consistently remains around the same value. The maximum SEM value is 0.4 meters, which is quite low. This indicates that the average

[2]http://wiki.ros.org/gmapping

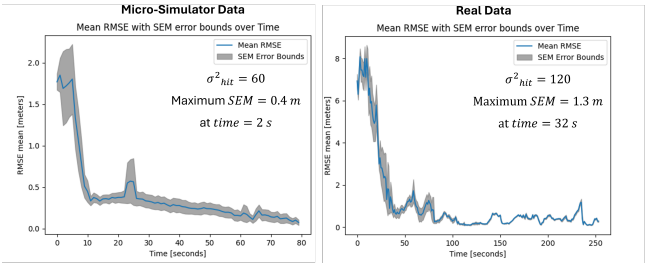

Fig. 8: Mean RMSE with SEM error bounds over Time

obtained is reliable, as the results show little variability and do not depend significantly on the initial conditions of each trial.

For the real data, the optimal variance value was also used ($\sigma^2_{hit} = 120$) and the same analysis was performed. For this data, the SEM value is higher (with a maximum SEM of 1.3 meters) compared to the data from the micro-simulator. Despite this, for later time instances the SEM values decrease substantially and become close to the values of the micro-simulator. Therefore, we can conclude that after some time a reliable average is obtained as the results show little variability and are not highly dependent on the initial conditions of each trial.

VII. Conclusions

This paper addressed the challenge of robot localization by employing the Monte Carlo algorithm within a preexisting map. The approach involved updating the probability of each particle using odometry and scan data. The implementation demonstrated success across both real-world data and simulated scenarios. The micro-simulator conducted our firsts experiences. After that, our algorithm was adapted to work for a real robot localization and not just with given odometry. Additionally, the algorithm was properly compared with the AMCL package provided by ROS, used as ground truth. The analysis of error metrics, including RMSE and SEM, showcased the correct implementation of the algorithm and its precision.

Furthermore, emphasizing the significance of MCL in real-world applications is crucial. MCL plays a vital role in various domains, including autonomous navigation, robotics and localization-based services. Its ability to accurately estimate a robot's pose relative to a known map, even in dynamic and uncertain environments, makes it indispensable for tasks such as autonomous exploration, mapping and path planning.

References

[1] Y. Sun, "A comparative study on the monte carlo localization and the odometry localization," 2022.
[2] N. Akai, "Reliable monte carlo localization for mobile robots," *Journal of Field Robotics 40, n.º 3*, May 2023.
[3] I. Rekleitis, "A particle filter tutorial for mobile robot localization," Jan. 2004.
[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.