Estimate position of Turtlebot3 using Monte Carlo Localization Algorithm

Rafaela Oliveira-100066*, Rita Mendes-100072*, Rodrigo Coimbra-100078*, Tomás Martins-100102*
*Instituto Superior Técnico, Lisboa
Email: rafaela.oliveira@tecncico.ulisboa.pt, rita.cordeiro@tecnico.ulisboa.pt
rodrigo.coimbra@tecnico.ulisboa.pt, tomasmartins77@tecnico.ulisboa.pt

***Abstract*—This article addresses the problem of localization in autonomous systems using the Monte Carlo algorithm. This localization estimation utilizes the sensors embedded in the robot to approach the real position and orientation. This method is implemented using a particle filter, thereby gathering data from the two sensors used. The contributions of this article can be divided into two key aspects: i) the motion model chosen and how it was used to solve the problem and ii) the way the algorithm was developed so that the localization was as accurate as possible. The simulations conducted and the real-world experiments provide substantial evidence regarding the progress made in the localization of the robot, as indicated by the conclusive outcomes.**

***Index Terms*—Monte Carlo localization, particle filter, likelihood, raytracing, RMSE**

I. Introduction

Localization in robotics plays a fundamental role in enabling robots to understand and interact with the space around them. It is essential for autonomous navigation, obstacle avoidance and effective human interaction. Over the years, localization in robotics has significantly driven the evolution of the human species, enabling notable advancements in various fields.

An example is the evolution of service robots used in hospitals and restaurants. These robots, with the ability to accurately locate themselves, perform tasks such as delivering supplies and providing services to patients or customers. By knowing their exact location, these robots can move efficiently and safely through the environment, meeting the needs of people and improving the quality of services provided [1].

Moreover, localization in robotics has been essential for space exploration. Robots used in space missions, such as rovers on Mars [3], rely on precise localization to navigate unknown terrains, avoid obstacles and safely return to their initial position. This localization capability has propelled our understanding of space and opened new possibilities for exploration.

This work addresses two fundamental problems: localization and orientation of the TurtleBot3. The Monte Carlo localization method is employed to solve these problems. The proposed solution is evaluated using the Gazebo simulator[1], and after testing in the simulator, the algorithm was tested on real data.

[1]https://gazebosim.org/home

II. Problem Description

The localization problem involves determining the position of the robot and orientation on a known map. In this project, the Turtlebot3 is used and equipped with three sensors: odometry, camera, and LDS. Two assumptions are made: 1) The center of mass of the robot is a point in a 2D environment with a static known map, and 2) Velocities (linear and angular) remain constant between timestamps due to discrete measurements and the low time interval between them. Although velocity cannot be precisely determined, this assumption is a reasonable approximation.

A. Sensors

1) Odometry: Odometry provides information about the position, orientation, linear and angular velocities. In this project, the data used to develop the algorithm were the linear and the angular velocities, given that both position and orientation are obtained by the odometry from the velocities.

2) Laser: The embedded LDS in the Turtlebot3 has a scanning area of 360 degrees, with a resolution of 1 degree, and can detect objects between 12 cm to 3.5 m.

B. Problem Framework

*1) **Motion Model**:* Turtlebot3 provides us with the translation and rotation velocities of its wheels using odometry, making this motion model rather simple, utilizing the equations of motion to predict where the robot moved. When working in two dimensions, it is indeed important to consider not only the x and y positions but also the angle between the two dimensions.

Let v_t denote the linear velocity with time t and w_t the angular velocity and considering that a positive w_t induces a counterclockwise rotation, known as left turns, while positive v_t correspond to forward motion. Let x_{t-1} represent the previous position in time and x_t represent the current one. Furthermore, we assume a_t as the linear acceleration and α_t as the angular acceleration.

$$\frac{d\theta_t}{dt} = w_t + \alpha_t \cdot t \tag{1}$$

$$\frac{dx_t}{dt} = v_t cos(\theta_t) + a_t \cdot t \tag{2}$$

$$\frac{dy_t}{dt} = v_t sen(\theta_t) + a_t \cdot t \tag{3}$$

In Equations 2 and 3, θ_t must be accounted in order to perform a translation on the new orientation. Furthermore, using the assumption that velocities are constant then accelerations are equal to zero. After this consideration, Equations 1, 2 and 3 were integrated with respect to time. Therefore, the kinetic motion model used can be written as Equation 4:

$$\begin{bmatrix} \theta_t \\ x_t \\ y_t \end{bmatrix} = \begin{bmatrix} \theta_{t-1} + \omega_t \Delta t \\ x_{t-1} + v_t cos(\theta_t)\Delta t \\ y_{t-1} + v_t sen(\theta_t)\Delta t \end{bmatrix} \tag{4}$$

To determine the new position, two steps need to be performed: rotation, followed by a translation. In Figure 1, we can observe the initial orientation of the robot, θ_{t-1}, then it rotated and its orientation became θ_t. In the end, it performed the translation in the direction of θ_t for ρ [4, pp. 2–3] meters (Equation 5).

$$\begin{bmatrix} \Delta x_t \\ \Delta y_t \\ \rho \end{bmatrix} = \begin{bmatrix} x_t - x_{t-1} \\ y_t - y_{t-1} \\ \sqrt{\Delta x_t^2 + \Delta y_t^2} \end{bmatrix} \implies \begin{bmatrix} \Delta x_t \\ \Delta y_t \\ \rho \end{bmatrix} = \begin{bmatrix} v_t cos(\theta_t)\Delta t \\ v_t sen(\theta_t)\Delta t \\ \sqrt{\Delta x_t^2 + \Delta y_t^2} \end{bmatrix} \tag{5}$$

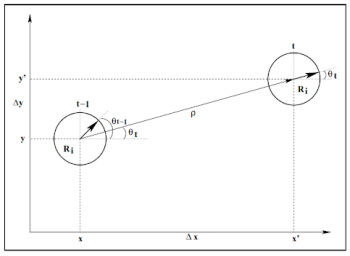

Fig. 1: Motion Model

*2) **Range finders**:* This model measures the range to nearby objects, in this case, measured along a beam. In this project, the robot emits an infrared light that measures the time it takes for the signal to go and bounce back after hitting an object. Based on the time-of-flight and on the velocity of light, the sensors can estimate the distance, Equation 6, to the object:

$$distance = \frac{1}{2}(velocity \times time) \tag{6}$$

III. Methods and Algorithms

A. Algorithm Outline

*1) **Initial Particle Distribution**:* To achieve absolute localization when the initial position of the robot is unknown, we randomly assign an orientation and initial position from a **uniform distribution** using an appropriate number of particles. Each particle represents a possible robot location and the number of particles is proportional to the map size. If the initial location belief is known, we initialize the particles around this belief using a**uniform distribution** and a range of starting angles.

*2) **Prediction**:* As we obtain data from the odometry meter, it is imperative for us to accurately predict the position of all particles, taking into account this new information. This allows us to effectively align with the movement of the robot. To achieve that, every particle must update its state **t-1** to the next state **t** with the help of the motion model.

Additionally, a layer of noise is introduced to the aforementioned prediction in order to rectify any potential inaccuracies or updates. As this filter operates on a probabilistic basis, it becomes necessary to compensate for the inherent errors that typically occur during the motion of the robot.

*3) **Update**:* This particular step of the particle filter involves receiving data from laser measurements to calculate the likelihood of a given particle occupying a particular location. The laser provides a set of **N** range values for each reading. Considering that the sensor has defined maximum and minimum ranges, any values falling outside those ranges are appropriately converted to the corresponding maximum or minimum range detected by the laser.

Then, for every particle n = 1 ... N, the *raytracing 2D algorithm*[2] is used as seen in Figure 4. This algorithm simulates where the ranges of the laser sensor would hit if that given particle were the correct robot position, providing a difference between the real measurements and the simulated ones, this difference can be used to compute the likelihood of that given particle being in the correct position.

This *likelihood* [5, p. 126] is calculated through a **Gaussian distribution**, Equation 7, where the predicted ranges r^p and the measured ranges r^m are compared, giving us:

$$p(r_t^m | x_t^n) = \frac{1}{\sqrt{2\pi |Q|}} e^{-0.5(r_n^m - r_n^p)^T Q^{-1} (r_n^m - r_n^p)}, \tag{7}$$

Where **Q** is the variance matrix of the sensor measurement readings and both **r** are the matrices containing all the laser measurements, as multiplying matrices is more efficient. After that, it is calculated an average weight among all particles that will be used for kidnapping purposes. To enter the next phase, we must first normalize these weights.

Short-term and long-term likelihood: This step is exclusively to deal with the kidnapping problem. For that, two new weights are calculated, using Equation 8:

$$w_i = w_i + \alpha_i \cdot (w_{avg} - w_i) \tag{8}$$

These weights, referred to as w_{slow} and w_{fast}, are used to determine the quantity of new particles that will be created, which will replace the existing ones while maintaining the same number of particles. For that, each one has a different α, where $\alpha_{slow} < \alpha_{fast}$. What this does is that whenever the particles lose the robot, their average weight goes down, and due to the fact that w_{fast} has a greater α than the w_{slow}, it will decrease faster [5, pp. 206–207].

*4) **Resample**:* The resampling process entails generating new particles at the locations where the ones with higher weights reside, while proportionally eliminating particles with lower weights. This ensures that the number of particles remains consistent, and with each sampling iteration, the particles gradually converge towards the pose with the highest probability. Although, resampling is one of the key components of this algorithm, excessive resampling takes the risk of

[2]https://github.com/yz9/Monte-Carlo-Localization blob/master/python /monte_carlo_localization_v2.py

losing the diversity of particles. Due to that, every time that the robot remains in the same state, i.e, the linear velocity and angular velocity received are below the threshold of 0.001 $m \cdot s^{-1}$ and 0.01 $rad \cdot s^{-1}$ respectively, it does not resample. Besides that, resampling only occurs when a given number of effective particles, N_{eff} (9), falls below a certain threshold.

$$N_{eff} = 1/\sum_{i=1}^{N}(w_t^i)^2 \tag{9}$$

To improve the update frequency and maintain the particle population, the ***low variance sampler algorithm*** [5, p. 86] was selected. This algorithm reduces errors caused by the randomness of resampling by using a sequential stochastic process. It has a computational time complexity of *O(N)* and ensures that if the mathematical expression 10 is positive, a proportional number of new particles will be added to the new set of particles.

$$1.0 - \frac{w_{fast}}{w_{slow}} \tag{10}$$

This steps are detailed in Algorithm 1, where X_t is the entire set of N particles in that instance of time:

Algorithm 1 Monte Carlo Algorithm

static w_{slow}, w_{fast}
Initialization: Create **N** particles:
$x_0^n \sim$ U(0, TotalEmptySpaces)
$w_0^n = 1/N$
Prediction: Receive odometry readings:
for n = 1 in N **do**
  x_t^n = *sample_motion_model*(x_{t-1}^n)
end for
Update: Receive laser readings:
for n = 1 in N **do**
  Update particle weight:
  $w_t^n = w_{t-1}^n \cdot p(r_t^m | x_t^n)$
  $w_{avg} = w_{avg} + \frac{1}{N} \cdot w_t^n$
end for
Normalize the weights for each particle:
$w_t^n = w_t^n / \sum_{i=1}^{N} w_t^i$
Short-term and long-term likelihood:
$w_{slow} = w_{slow} + \alpha_{slow} \cdot (w_{avg} - w_{slow})$
$w_{fast} = w_{fast} + \alpha_{fast} \cdot (w_{avg} - w_{fast})$
Compute the effective sample size:
$N_{eff} = 1/\sum_{i=1}^{N}(w_t^i)^2$
Resample:
if $N_{eff} <$ N_{thresh} **then**
  for n = 1 in N **do**
    P = $1.0 - \frac{w_{fast}}{w_{slow}}$
    if $P > 0$ **then**
      add with probability P a new random particle to X_t
    else
      draw and add to X_t from x_t with probability $\propto w_t$
    end if
  end for
end if
Go to prediction

IV. Implementation

A. Error Metrics

The particle filter [2] performance will be evaluated in two environments: Gazebo, a digital simulation tool, and the real-world data collected from the fifth floor of the North Tower at Instituto Superior Técnico. This approach allows for comprehensive assessment and validation of the particle filter functionality and robustness across both simulated and real-world scenarios. To better analyze this data, it must be compared with the groundtruth, for that will be used Adaptative Monte Carlo Localization, **AMCL**[3], a package provided by **ROS**[4]. The performance will be evaluated in multiple runs, gathering the difference between the particle filter and **AMCL** in terms of the root-mean-square-error (**RMSE**) 11. To better understand the performance, the dead reckoning will be available for comparison. In Equation 11, $\bar{x}_{AMCL}$ denotes the average x position from **AMCL** and $\bar{x}$ denotes the average x position from either dead reckoning or estimate, depending on the calculation.

$$RMSE = \sqrt{(\bar{x}_{AMCL} - \bar{x})^2 + (\bar{y}_{AMCL} - \bar{y})^2} \tag{11}$$

B. Monte Carlo Node

To generate the graphs shown in Figures 3, 5, 7, 10, data was extracted from the odometry and laser scan of the robot using the */odom* and */scan* topics, respectively. This data was then used as input for the *montecarlo_node*, which executed the Monte Carlo localization algorithm. The *montecarlo_node* published two topics: */MaxWeight* and */particles*. The */MaxWeight* topic provided the mean position of the best particles, representing the estimated position of the robot. The */particles* topic contained data for all the positions of a set of particles, showing the possible points where the robot could be. The *graphs_node* subscribed to both topics and plotted the required graphs as shown in Figure 2.

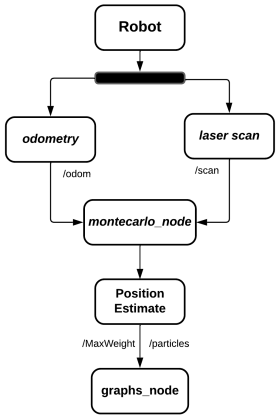

Fig. 2: Ros nodes and topics

[3]Adaptative Monte Carlo Localization http://wiki.ros.org/amcl
[4]Robot Operating System http://wiki.ros.org/

V. Simulation Results

*1) **Simulation environment**:* Testing procedures began in Gazebo, a robot simulator, using a digital environment. The simulation involved emulating a turtlebot within a 3D square map measuring 4.75 meters in length and width. The map included obstacles for the robot to navigate around. This approach offers a valuable testing platform as the odometry meter and laser scanner readings closely resemble real-world scenarios. By conducting experiments in this simulated environment, we can evaluate the performance of the system before transitioning to real-world data collection.

*2) **2D occupancy grid map**:* Utilizing **gmapping**[5], a **ROS** package tool to help create a 2D grid occupancy map, the constructed map was very accurate to the gazebo world, due to its simplicity and the precise readings of the laser scan, in this particular environment, this map was used throughout all simulation tests.

*3) **Monte Carlo localization results**:* The parameters utilized by the particle filter in the simulations are given in Table I. For these tests, the particles were initialized (in Figure 3) in a uniform distribution through the whole map in free spots (white pixels), with sweeping angles of [0, 2π].

Parameters	Values
Number of particles, N	800
Resample threshold, N_{thresh}	240
Number of laser rays, L	90
α_{slow}, α_{fast}	0.1, 0.8
Laser variance matrix, Q[L x L]	$\begin{bmatrix} 60 & \dots & 0.00 \\ \dots & 60 & \dots \\ 0.00 & \dots & 60 \end{bmatrix}$

TABLE I: Particle filter parameters

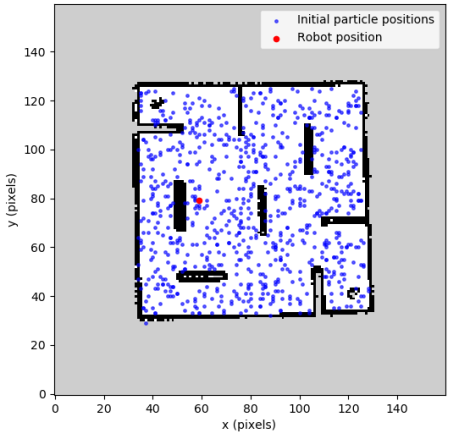

Fig. 3: Initialization of particles

To better exemplify how the **raytracing** algorithm works, Figure 4 shows the 90 rays of the LSD laser from the center of the position of the particle until it reaches a wall.

For the first experiment, the algorithm was run alongside the **AMCL** algorithm (the groundtruth) to compare the behavior of the mean of the 16 particles with the highest weights when

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

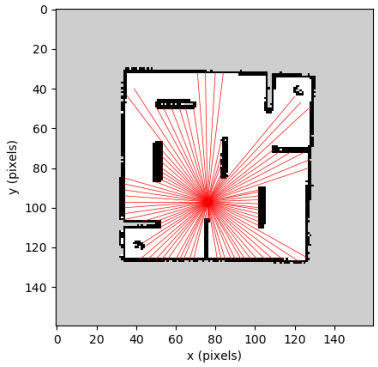

Fig. 4: Raytracing for an example particle

comparing their estimated position with the ground truth and dead reckoning positions, shown in Figure 5. To ensure reliable results, we calculated **RMSE** metric and conducted 10 separate runs for a more robust analysis. The average **RMSE** across these runs is shown in Figure 6. The dead reckoning method initially has a lower error but it increases over time, reaching an **RMSE** of around 1 meter. On the other hand, Monte Carlo localization starts with a higher error due to the uniform distribution of particles, but it gradually reduces the error to below 0.3 meters. With each iteration, the error continues to decrease, resulting in a final RMSE below 0.2 meters.

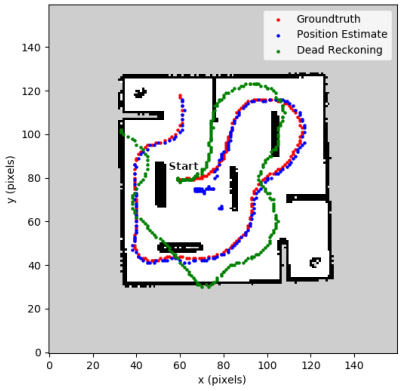

Fig. 5: Groundtruth, dead reckoning and best position estimate trajectory over a map

In Table II, it is presented a summary of the various tests that were performed. For the same type of simulation, we varied the variance values and observed the number of times the algorithm converged, and found the robot. We assumed that the robot converges when the **RMSE** value is stable and less than 0.2 meters. Fifteen tests were conducted for each variance value.

Variance	1	5	15	60	90	120	160
Count	8	11	13	15	15	15	15
time (s)	0.96	3.92	6.87	8.64	12.49	22.17	26.39

TABLE II: Number of times it converged in 15 runs

Based on these results, we can infer that lower variance leads to faster convergence but a higher frequency of misses

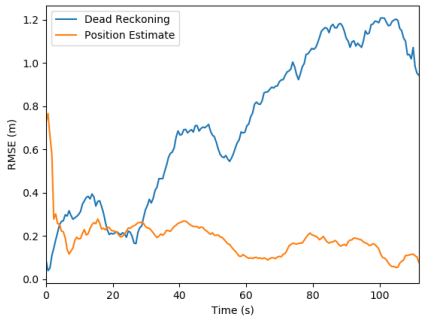

Fig. 6: RMSE of simulated data

in the algorithm. Conversely, increasing the variance improves accuracy but lengthens the convergence time significantly. This occurs because lower variances rely more on laser readings and frequent resampling, while higher variances rely more on odometry, reducing the resampling frequency and resulting in a larger particle cloud representing the position of the robot.

Finally, kidnapping tests were performed in Figure 7 to assess the robustness of the algorithm. In these tests, with variance of 20, data was intentionally withheld from a specific timestamp of approximately 20 seconds. When the missing data was reintroduced, the algorithm detected that it had lost track of its position and initiated a process of scattering particles across the map in an attempt to relocate itself. Remarkably, this method proved successful, as the robot managed to reestablish its position in all 10 test runs conducted.

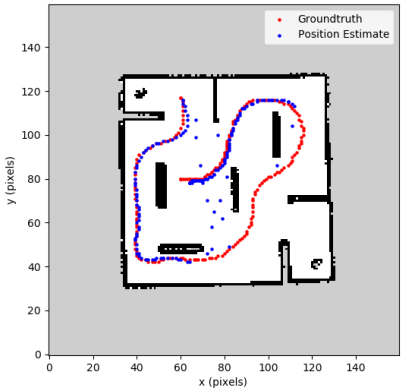

Fig. 7: Robustness to kidnapping

The **RMSE** values obtained from these 10 runs were also promising. In Figure 8, it can be observed that around the 50-second mark, the algorithm lost track of its position. However, it quickly and accurately reestablished its location within 10 seconds resulting in a significant decrease in the **RMSE** to normal levels. This demonstrates the ability of the algorithm to recover from unexpected disruptions and maintain accurate localization performance.

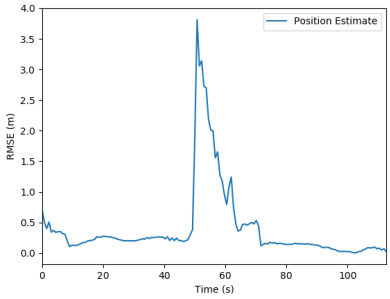

Fig. 8: RMSE of position estimate with kidnapping

VI. Experimental Results

To test our algorithm in a real scenario, we created a 2D map of the fifth floor of the north tower at Instituto Superior Técnico (IST), as shown in Figure 9. Due to limitations of the WiFi of the robot, it was not possible to fully map the entire floor. However, the figure displays the complete path taken by the robot. The trajectory shown is extracted from a recorded *rosbag* file using **AMCL**.

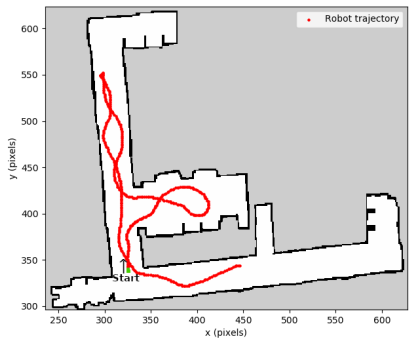

Fig. 9: Map of floor 5 with the trajectory of the robot

After conducting several tests with this map and using the parameters in Table III, it was observed that the algorithm needed a higher number of particles to cover the large map adequately. However, the complexity of the **raytracing** algorithm imposes limitations on the particle count that can be utilized. To tackle this challenge, the adopted approach was based on an assumption made about the initial location of the robot, and the particle distribution is confined to that specific area. This strategy eliminates the requirement for an increased number of particles.

Parameters	**Values**
Number of particles, N	1000
Resample threshold, N_{thresh}	300
Number of laser rays, L	90
laser variance matrix, Q[L x L]	$\begin{bmatrix} 60 & \dots & 0.00 \\ \dots & 60 & \dots \\ 0.00 & \dots & 60 \end{bmatrix}$

TABLE III: Particle filter parameters

Additionally, the map of the fifth floor exhibits significant symmetry due to the corridors. The starting position of the *rosbag* data being at the center of the intersection caused the particles to frequently diverge towards each corridor, resulting

Variance	1	5	15	60	90	120	160
Count	9	12	13	14	14	13	11
Time (s)	1.21	7.77	10.20	16.32	30.11	39.57	53.37

TABLE IV: Number of times it converged in 15 runs

in inaccurate outcomes. To address this, the initial orientation of the particles were assumed to be uniformly distributed between $[-\pi/2; \pi/2]$ relative to the orientation of the robot. Figure 10 shows the initialization of the position of all particles in the restricted area, along with the starting point of the robot.

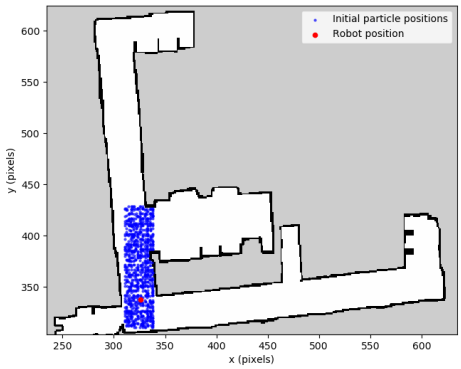

Fig. 10: Initial particle positions

In Figure 11, the **RMSE** is consistent in values below 1.1 meter, only rising above that when, in the beginning, due to the symmetrical implications of the map in Figure 10, while starting in the corridor, the algorithm has difficulties converging to the right position, but as it enters the room above, it becomes easier for it to find.

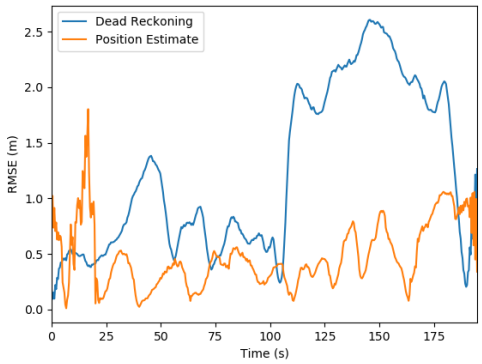

Fig. 11: RMSE of real data

It is important to acknowledge the sudden decrease of the dead reckoning in the final instants of the route, this decrease is due to probability caused by the route the robot takes because it needs to drive back through the same corridor instead of driving around the floor, as shown in Figure 9.

In Table IV, we performed the same tests as in Table II, but now using real data. The only difference is that we consider convergence when the **RMSE** value is stable and less than 1 meter.

Despite limitations, the results in Figure 11 show satisfactory performance considering the given restrictions. In this situation, the algorithm exhibits slightly reduced accuracy and longer convergence time. Initializing particles further right or up, as shown in Figure 10, would introduce additional errors due to the symmetrical nature of the corridors in the starting area. This symmetry poses a challenge for accurate localization, contributing to the observed error. However, considering the constraints of the environment, the algorithm still performs well.

Kidnapping experiments were not induced in real data due to not having significant results that show a good performance because of the big dimensions of the map and the restrictions on the number of particles that could be used, so it was turned off throughout all real data tests.

VII. Conclusions

In this paper, we addressed the problem of robot localization using the Monte Carlo algorithm in a previously known map. The proposed solution uses odometry and scan data (using raytracing) to update the probability of each particle. In the simulated environment, absolute localization has been achieved as the error was low. However, these results may vary depending on the amount of artificial noise added. If this noise has a very low value, the error tends to increase. Furthermore, in the real environment, the map tested was much bigger, and because of limitations of the algorithm, absolute localization could not be achieved. Because of that, a different approach was taken, focusing on relative localization, assuming an area that contained the first position of the robot.

Future work will mainly focus on absolute localization in larger maps, including particle initialization throughout the whole map to improve the results with real data, this can be achieved by optimizing the overall algorithm but mainly raytracing, that proved to be a computationally demanding solution. Furthermore, the algorithm could also be optimized by handling a kidnapping situation while performing absolute localization. These improvements will enable a broader coverage of the environment and a finer estimation of position, resulting in a more precise and reliable localization.

References

[1] P. Gonzalez de Santos E. Garcia M. A. Jimenez and M. Armada. “The Evolution of Robotics Research”. In: (2007), pp. 1–5.

[2] Burgard Fox Thrun and Dellaert. *Particle Filters for Mobile Robot Localization*. 2001.

[3] M. Powell et al. “Targeting and Localization for Mars Rover Operations”. In: (2006), p. 1.

[4] Ioannis M. Rekleitis. “A Particle Filter Tutorial for Mobile Robot Localization”. In: (2002).

[5] W. Burgard S. Thrun and D. Fox. *Probabilistic Robotics*. 2005.