Performance Analysis of Monte Carlo Localization Algorithm

Mert Gokbulut
Responsible Professor: Ranga Rao Venkatesha Prasad
Supervisors: Ashutosh Simha, Suryansh Sharma
EEMCS, Delft University of Technology, The Netherlands
22-6-2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

One of the key problems of swarm robotics is how the mobile robots can navigate accurately in a given environment. To achieve this, the mobile robots need to accurately determine where they are globally, or relative to other robots and landmarks. This paper is going to be an investigation of the Monte Carlo Localization algorithm in an environment containing 3 anchors. In addition, an alternative modification is suggested to this localization algorithm to improve the localization performance. An experiment is devised to assess different aspects of localization performances of these algorithms. With the experimental results, a quantitative analysis of performance figures are compared and analyzed.

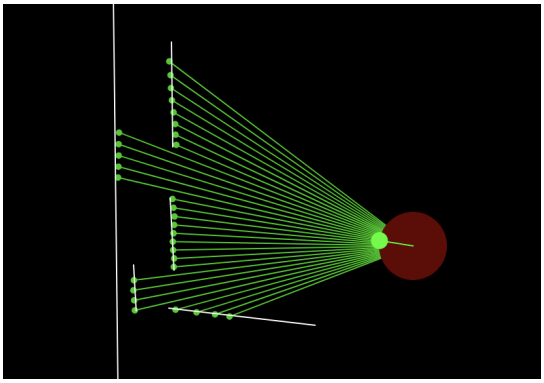

Figure 1: Visualization of LIDAR sensor distance data

1 Introduction

Swarm robotics is an actively researched topic that is interested in the coordination of a system containing multiple robots. The main challenge of swarm robotics is solving a problem where robots coordinate cooperatively [2]. Before defining the behaviour of the swarm, there are certain sub problems that needs to be addressed. The sub problem this paper is going to be focusing on is localization of robots.

Localization is the problem of a robot estimating its current position within the environment it is in. For now, we can split localization of a robot into two different scenarios. First being globally localizing the robot which means that the robot can estimate its position within a known environment. Second scenario is the relative localization which is concerned with the robot identifying itself by understanding the relative distance between itself and the objects within the unknown environment.

There are multiple ways that the robot can be localized. For the robot to comprehend and analyze the environment, there are various different types of proprioceptive sensors that can be used to get a position estimation. That being said, it is quite uncertain to get a accurate position estimation with raw sensor data. The uncertainty in the position estimation can be risen by sensor noises, the approximate nature of the used algorithm and the inaccurate model of the environment. The robot needs multiple sources of information which can be analyzed by probabilistic robotics (statistical robotics) [7] approaches while estimating its position to reduce the uncertainty.

In this paper, we will investigate globally localizing a robot using noisy light detection and ranging (LIDAR) sensor [10], an Inertial Measurement Unit (IMU) [6] combined with a particle filter [3]. In addition, we will investigate the particle filter performance using 3 anchors [3]. The goal of the investigation is to definitively localize the robot using minimum number of steps up to some uncertainty.

1.1 LIDAR Sensor

A LIDAR sensor [10] measures the distances between the obstacles, and the sensor itself. A 2D representation of a LIDAR sensor projected by a robot is given in figure 1. The white lines represent the obstacles (walls). Each green line represents the distance between the landmark and the robot at a given angle. The figure given represents a noiseless LIDAR sensor, however, this is hardly the case in real life which is being discussed in the Sensor Noise section. In this paper, the virtual robot in the test environment is going to be sensing 3 anchors using a virtual LIDAR sensor.

1.2 IMU

An IMU sensor [6] is a widely used sensor in robotics to keep track of quantities such as distance traveled, velocity, acceleration, rotation data, etc. In real-life applications, just like a LIDAR sensor, it can be noisy. IMU sensors are usually paired with another sensor measurement to produce an accurate position estimation. A Kalman filter [11] is generally used to combine these measurements and produce a position estimation. In this paper, the virtual robot that is being tested is going to have an IMU sensor which has an odometer and rotation data.

1.3 Sensor Noise

In the real world, it is unrealistic to expect a sensor to behave perfectly. This is why when analyzing the performance of the filters mentioned, different sensor noise values need to be considered. Two sensors will be utilized in the simulation (LIDAR, IMU), must be noisy to make the experiments as realistic as possible. These sensors can have a random noise behavior or can be modeled using Gaussian distribution. For instance, the LIDAR reading between the boundary and the sensor itself could be a bit further away or closer to the robot itself in reality. By amplifying this randomness, the simulator can be tested with high and low noise in separate experiments. The sensor noise does not have to be random, it could be progressive (random but smooth) and could follow a pattern that can be modeled with Perlin noise [5]. In addition, the sensors could progressively get less accurate so they could be prone to drift.

1.4 Localization using a Particle Filter

In this subsection, how a robot can utilize a particle filter to localize itself in a known environment is going to be discussed. At the start of the localization process, the robot is located at a random position on the map and it does not have any prior clue about where it is located. This means it is equally probable for the robot to be at any position and direction within the known map.

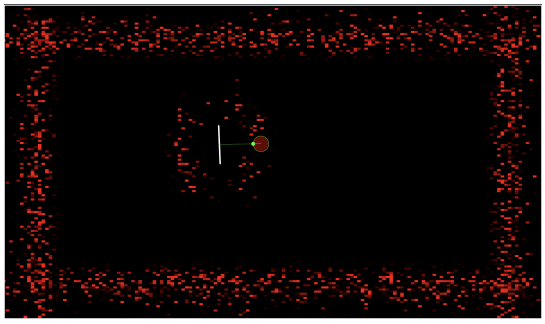

Figure 2: Probability Density Map of Robot Location (color red is highly probable)

The robot starts by taking an initial LIDAR measurement. From this, the robot gains information on how far it is located from the obstacles (or landmarks, anchors) on the map. This measurement alone is not enough to estimate where the robot is since the map might contain multiple obstacles and the robot could be sensing any of them. That being said, the robot can deduce that it is at a point that is some distance away from the obstacle and therefore can eliminate possible positions and direction combinations on the map that do not satisfy these criteria. This measurement constrains the robot to be at specific locations.

An example of a probability distribution of robot position is given as a heat-map at figure 2. The given map has a wall on the outline and a single, white-colored wall segment in the middle. The red points represent the possible robot locations given that the robot is measuring the distance between itself and the wall segment in front of it. The density map for this figure is generated by taking random particles (position and direction) and comparing them to what the LIDAR sensor sees. Figure 2 shows that the robot can be at a position and direction in which the LIDAR sensor reads a particular distance. It can either be near the borders of the room looking towards the borders at the same distance the actual robot is viewing the wall segment in the middle, or it can be near the wall segment in the middle. As mentioned, it is highly unlikely for the robot to be anywhere else on this simple map therefore robot may already have a rough idea of its position and direction. The reason why the red clusters in the figure 2 are widely spread is because of the sensor noise. This can be modeled as Gaussian noise and allow distance measurements up to some multiple of the standard deviation of the distribution.

To converge these possible locations to a single location, the robot can move around the map to observe new features of the map. Doing this for multiple steps helps eliminate positions that were initially possible. By recursively updating the weights of the possible locations, the robot can narrow down the possible locations it can be in.

1.5 Relevant Limitations of Particle Filter

In some cases, the known map might have rooms that are quite similar to each other or the map might be too simple to analyze. In such cases, it is harder for the particle filter to converge to position and direction estimation in a small number of steps. In this paper, a step is defined as the single time frame motion of the robot which contains both position and direction change.

Even tough particle size could be varied, to have a good position estimation, generally, a large number of particles need to be used which is computationally expensive as the particle size grows.

The particle filter is non-deterministic. This means that the behaviour might not be reproducible. The systems taking the same inputs may result in different position estimations. This is not the case if the same random seed is used for particle sampling.

2 Formal Problem Description

2.1 Problem Description

This paper is going to be an investigation of the performance of the Monte Carlo Localization algorithm in a known environment containing 3 anchors through simulations. The simulated robot is going to contain a LIDAR sensor and an IMU. In terms of performance, the minimum step size & sample size required for the robot to be localized below 5% error are going to be assessed. In addition, the algorithm is going to be modified and tested to reduce minimum step size and minimum percentage error while position estimation.

2.2 Monte Carlo Localization

Monte Carlo Localization algorithm [8] is a particle filter based localization algorithm in which the positions and directions of the particles (with preset particle size) are initially uniformly randomized. It is a recursive algorithm meaning it has a prediction and correction step when doing a position and direction estimation. For the particle set with J particle size, X is the weighted samples where x^j is the state hypothesis which is the position vector and w^j is the importance weight which is a real number.

$$X = \{\langle x^{[j]}, w^{[j]} \rangle\}_{j=1,2..J}$$

The samples represent the posterior where $\delta_{x^{[j]}}$ is the Dirac impulses.

$$p(x) = \sum_{j=1}^{J} w^j \delta_{x^{[j]}}(x)$$

Initially, if there is no prior information about where the robot could be in the map, the particle sample is randomized. One way of selecting samples is by re-sampling the particles with a sample probability proportional to the importance weight. This technique is called importance sampling [9].

The algorithm steps are as follows

1. Prediction Step, generating a sample particle set by using a proposal distribution. This step aims to estimate where the original robot position could be in the environment.

   $$x_t^{[j]} \sim proposal(x_t|...)$$

   Particles generated are going to be treated as a position hypothesis where the proposal p is the observation

model, u is the movement & rotation command, x_{t-1} is the previous belief on where the robot was, and x_t is the current belief on where the robot is.

$$x_t^{[j]} \sim p(x_t|x_{t-1}, u_t)$$

2. Correction Step, by using importance sampling, the algorithm calculates the importance weights of the particle set generated in step 1. The weights calculated are proportional to the outcome of the observation model given x_t, z_t which is the measurement at time t using LIDAR and m as the map of the environment.

$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})} \propto p(z_t|x_t, m)$$

3. Re-sampling Step, which pools a sample from the proposal distribution by its importance weight. This step replaces the weighted samples into frequencies so that sample j is going to have a probability of $w_t^{[j]}$. This process is repeated J times for re-sampling. This means that the re-sampled distribution is going to contain highly probable samples.

Algorithm 1 MCL Algorithm derived from [8]

MCL(X_{t-1}, u_t, z_t, m)**:**
- $\bar{X}_t = X_t = \emptyset$
- $j = 1$
- **while** $j \neq J$ **do**
  - $x_t^{[j]} \sim p(x_t|x_{t-1}^{[j]}, u_t, m)$ ▷ sample particle
  - $w_t^{[j]} = p(z_t|x_t, m)$ ▷ calculate weight
  - $\bar{X}_t = \bar{X}_t + \langle x^{[j]}, w^{[j]} \rangle$
  - $j = j + 1$
- **end while**
- $w_t = w_t / J$ ▷ Normalize the weights
- $j = 1$
- **while** $j \neq J$ **do** ▷ Importance Sampling
  - draw sample $x_t^{[j]}$ with probability $\sim w_t^{[j]}$
  - $X_t = X_t + x_t^{[j]}$
  - $j = j + 1$
- **end while**
- return X_t

3 Modifying MCL Algorithm

In this section, a modified version of the MCL algorithm is given to increase the minimum step size and lowest possible error performance.

3.1 Proposed Optimization

The proposed algorithm aims to reduce the minimum number of steps and overall lower uncertainty by modifying the importance sampling part of the Monte Carlo Localization (MCL) algorithm. Particle-based filters such as the MCL algorithm takes the positions of the particles into account. The original Monte Carlo Localization algorithm proposed is taking position data into account while localizing. The suggested algorithm presented is also will take direction and particle spread into account while localizing.

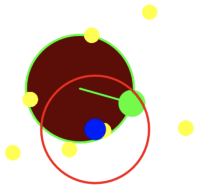

Figure 3: re-sampling based on prediction

The idea behind the proposed modification is based on knowing the robot's motion model, which is described in the Setup section. Knowing how much the robot travels in a single step, and how much direction it can change in a single step would allow us to make predictions, and allow us to manipulate the position and directions of our proposed particle distribution.

In the re-sampling step of the MCL algorithm, the importance sampling principle associates the weights of the particles with frequencies. This means that only the most highly probable particles are going to be considered for the next step in the localization process. This results in sampling that only consists of a small percentage of particles with high weights for the next generation which results in a slow rate of decrease in percentage error in small step sizes if the sensor noise is high.

To improve the particle sampling for the next step, an algorithm can disregard a portion of the initially re-sampled particles. The idea is that based on the position prediction, we can generate new particles that are some Gaussian noise away from the predicted position. This achieves a highly dense particle sample that is around the previous position prediction, which increases the rate of decrease in percentage error.

The figure 3 shows the robot itself outlined with a green color. The head of the robot is represented by the point circle which indicates the heading direction. The yellow points represent the particles after importance sampling. The blue point represents the prediction and the red circle around represents the outer bound of where the actual robot can be based on the LIDAR sensor noise. As it can be seen in the figure 3, importance sampling does not necessarily produce a sample set that is particularly close to the robot's position. That being said, the position estimation (blue point) is fairly accurate.

To add variance and accuracy to the model in later steps, the suggested algorithm makes use of this red circle and re-samples some portion of the already re-sampled particle set.

Another aspect in which the re-sampling section of the algorithm could be improved is by taking the motion model into account. The yellow-colored particles in figure 3 may

have accurate positions however they may not have accurate direction. At the next step of the simulation, some of these highly probable particles may become irrelevant. By storing the previous position estimate g_{t-1} and calculating the current position estimate g_t, the algorithm can roughly determine the direction of the robot. Knowing the maximum radius which the robot can turn in a single step and the noise in the rotation, would limit the possibilities of where the robot is heading. So $rotationNoise =$ Maximum rotation in a single step + sensor noise.

By combining steps 1 and 2, the algorithm can generate particles that are some Gaussian distance away from the prediction, and also would incorporate the position estimation into account. By re-sampling using this algorithm, the particles would be spread over an area that is highly probable and would move towards where the actual robot would be moving. This is why, in the next step, the particles that are moved have a higher chance of being near the actual position of the robot.

Gaussian(μ, σ^2) defined in the algorithm 2 returns a random sample of a Gaussian distribution of mean μ and variance σ^2. The function $centerPoint(X_t)$ returns the central position of the particles in set X_t.

Algorithm 2 Modified MCL Algorithm

MMCL$(X_{t-1}, u_t, z_t, m, g_{t-1})$**:**
- $\bar{X}_t = X_t = \emptyset$
- $j = 1$
- **while** $j \neq J$ **do**
  - $x_t^{[j]} \sim p(x_t | x_{t-1}^{[j]}, u_t, m)$ ▷ sample particle
  - $w_t^{[j]} = p(z_t | x_t, m)$ ▷ calculate weight
  - $\bar{X}_t = \bar{X}_t + \langle x^{[j]}, w^{[j]} \rangle$
  - $j = j + 1$
- **end while**
- $w_t = w_t / J$ ▷ Normalize the weights
- $j = 1$
- **while** $j \neq J$ **do** ▷ Importance Sampling
  - draw sample $x_t^{[j]}$ with probability $\sim w_t^{[j]}$
  - $X_t = X_t + x_t^{[j]}$
  - $j = j + 1$
- **end while**
- $X_t = sort(X_t)$ ▷ Sort using weights (high to low)
- $j = J * k$ ▷ k is a constant, from 0 to 1
- $X_t = X_t[0 : j]$ ▷ disregard k portion of X_t
- $g_t = centerPoint(X_t)$
- $\theta = g_t - g_{t-1}$ ▷ Estimated Direction
- **while** $j \neq J$ **do** ▷ Generate new particles
  - $n_t = g_t + Gaussian(g_t, lidarNoise)$
  - $n_t.dir = \theta + Gaussian(g_t, rotationNoise)$
  - $X_t = X_t + n_t$
  - $j = j + 1$
- **end while**
- return $\langle X_t, g_t \rangle$

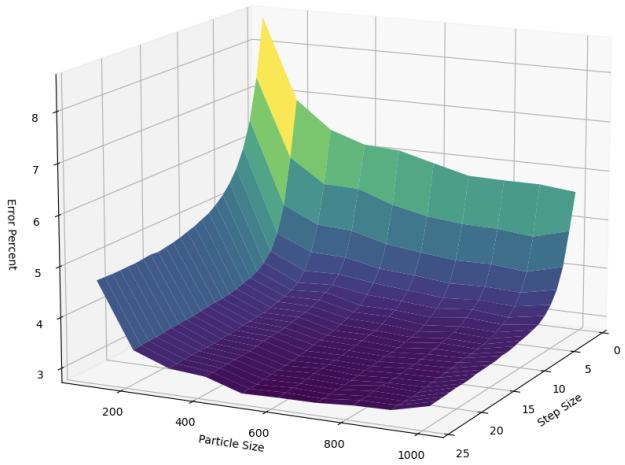

Figure 4: (MCL) Error percent based on particle size and step size

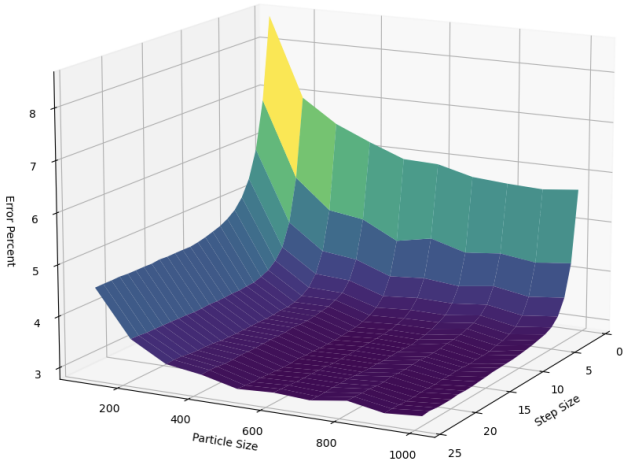

Figure 5: (MMCL) Error percent based on particle size and step size

4 Experimental Setup

In order to test the localization performance of the Monte Carlo Localization algorithm, a square map with 3 randomly placed anchors is going to be used. In each test, the robot is going to have a random initial position with random direction. In each of these tests, the mean distance of particles to the robot's position is going to be calculated and compared.

The tests are going to be conducted using a simulation. The performance criteria are going to be assessed based on the minimum number of steps that the robot needs to take before it is localized to %5 uncertainty. Additionally, the average uncertainty that is achieved in 5 steps is going to be assessed. This experiment is going to be repeated 10000 times for every setup to accurately assess the performance.

The simulation is going to have an environment of size 1000 by 1000 pixels which represents a room of size 100m^2. Initially, the robot's LIDAR sensor is assumed to have a 2% error on distance measurements on the anchors. A step is de-

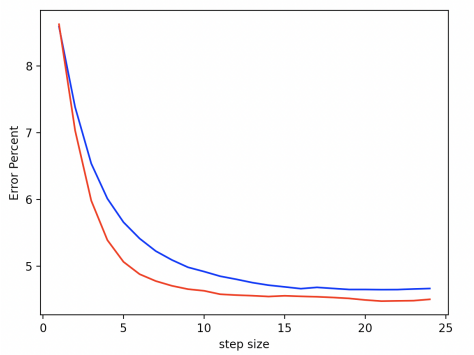

Figure 6: MCL vs MMCL, 100 particles

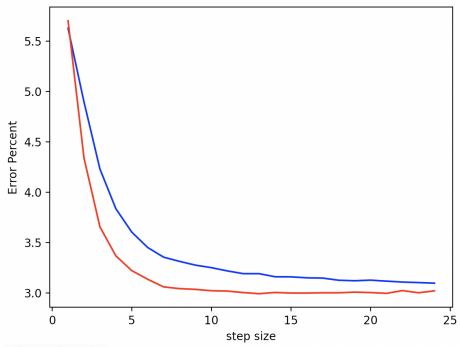

Figure 7: MCL vs MMCL, 1000 particles

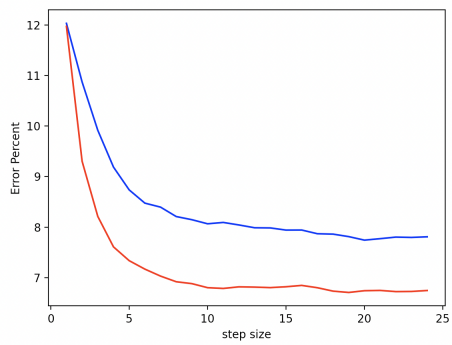

Figure 8: MCL vs MMCL, high noise, 100 particles

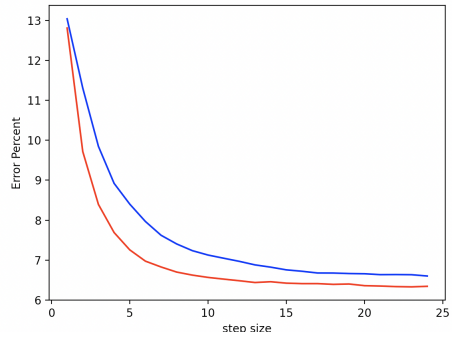

Figure 9: MCL vs MMCL, high noise, 1000 particles

fined as a 50cm movement in the environment. The robot's optometry and rotation are going to have a 1% error in a single step.

The robot's motion is based on steps. In a step, a robot travels roughly 50 cm and can rotate from $-\frac{\pi}{6}$ radians to $+\frac{\pi}{6}$ radians. The rotation is calculated by using Perlin noise [5]. This ensures that the robots change direction progressively, not instantaneously while discovering the environment. The directions are random however it is more predictable which is more representative of an actual robot's direction change model.

The simulation is written in typescript paired with the p5.js drawing library which allows for visualization of the simulation. A standard Monte Carlo Localization algorithm is initially implemented. Then the Modified Monte Carlo Localization algorithm is implemented. The performances of these two algorithms are compared and discussed in section 5.

5 Results

The performance criteria in this section is measured by calculating the mean distance of the particle set X_t to the actual robot position. The algorithm is run for 10^5 times with random anchor & robot positioning, up to 25 steps for both MCL and MMCL. The direct comparison for 100 and 1000 particles is given in figures 6 and 7 respectively, blue line representing MCL, red line representing MMCL.

In 5 steps, on average with 100 particles, MCL achieves 5.64%, MMCL achieves 5.05% error. At the end of 25 steps, MCL approaches 4.65% and MMCL approaches 4.47% error.

In 5 steps, on average with 1000 particles, MCL achieves 3.59%, MMCL achieves 3.21% error. At the end of 25 steps, MCL approaches 3.10% and MMCL approaches 3.00% error.

The figures 6 and 7 shows that the the rate of decrease of error percentage in MMCL is considerably higher than MCL.

Another aspect that can be analyzed is the fail rates of the algorithm. We are going to define a failing experiment as having a ϵ uncertainty and higher after α number of steps. Using 100 particles with $\alpha = 10$ and $\epsilon = 5\%$, MCL has a fail rate of 14.28%, MMCL has a fail rate of 13.31%. When it comes to a 1000 particles using the same α and ϵ, the fail rate of MCL and MMCL is 7.15% and 6.36% respectively.

That being said, in case of increasing the sensor errors, LIDAR from 2% to 5%, and optometry (both position and direction) from 1%, 3%, figures 8 and figure 9 still shows that the rate of decrease of error percentage in MMCL still are considerably lower than the MCL. Looking at the graphs, MMCL performs quite similar with lower particles where as in MCL, it can be clearly seen that increasing the particle size increases its performance. In both, 100 and 1000 particle cases, the MMCL approaches an uncertainty lower than 7% where as the MCL approaches this uncertainty in 1000 particle case only.

Taking Step size and particle size into account, the figure 4 and 5 show that for both of the algorithms, as the step size and particle size increase, the localization performance in terms of the minimum number of steps to achieve lower than 5% error, and achieving lower overall percentage error increases.

6 Discussion and Future Work

Robust Monte Carlo Localization for Mobile Robots [8] is the state-of-the-art algorithm that is used for indoor localization. MCL combines the advantages of accuracy and efficiency of a Kalman filter[11] and Markov localization[4] which is a grid-based localization algorithm.

Robots MINERVA, was being used as a museum tour guide in the Smithsonian's of American History using the Monte Carlo Localization algorithm for position estimation [8].

That being said, the Monte Carlo Localization algorithm is introduced in 1999, which is why there are existing specialized Monte Carlo localization algorithms for specific environments and sensor data. Some of the mainline advances are given in paper [1].

Another modified Monte Carlo Algorithm that takes position estimation and tracking into account is given in paper [12] which is tested in more complex environments to show how its performance scales compared to the original Monte Carlo Localization Algorithm.

The algorithm presented in this paper is specifically tuned for 3 anchors, this is why in a real-life situation where there are more landmarks or anchors, it may be beneficial to compare how a specialized Monte Carlo Localization algorithm behaves.

7 Responsible Research

The experiments devised and done on the research parts are evaluated using a statistical analysis. Since the localization algorithms are tested are non-deterministic, it is crucial to point out that the tests are reproducible. The repository used in this project contains a typescript project in which one can visually test how the algorithms behave, also generate multiple data sets to be analyzed. Also, python script is used to generate the graphics used in some of the figures, which is also included in the repository [1].

8 Conclusions

This experimental study discusses the performance of the original Monte Carlo Algorithm 1. In addition, a modified version of the Monte Carlo Algorithm 2 is presented for an environment containing 3 anchors. The parts in which the presented algorithm improves on the original algorithm are explained. The performances of these algorithms are quantitatively compared with figures. The real-life implications of these algorithms are questioned and debated. Finally, the repeatability aspect of the experiments conducted is explained.

References

[1] Olivier Cappe, Simon J. Godsill, and Eric Moulines. An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5), 2007.

[2] Marco Dorigo, Guy Theraulaz, and Vito Trianni. Reflections on the future of swarm robotics, 2020.

[3] Jos Elfring, Elena Torta, and René van de Molengraft. Particle filters: A hands-on tutorial. *Sensors (Switzerland)*, 21(2), 2021.

[4] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 11, 1999.

[5] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker. A survey of procedural noise functions. *Computer Graphics Forum*, 29(8), 2010.

[6] Alwin Poulose, Odongo Steven Eyobu, and Dong Seog Han. An indoor position-estimation algorithm using smartphone IMU sensor data. *IEEE Access*, 7, 2019.

[7] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3), 2002.

[8] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2), 2001.

[9] Surya T. Tokdar and Robert E. Kass. Importance sampling: A review, 2010.

[10] Mial E. Warren. Automotive LIDAR Technology. In *IEEE Symposium on VLSI Circuits, Digest of Technical Papers*, volume 2019-June, 2019.

[11] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *In Practice*, 7(1), 2006.

[12] Lei Zhang, René Zapata, and Pascal Lépinay. Self-adaptive Monte Carlo localization for mobile robots using range finders. *Robotica*, 30(2), 2012.

[1] https://github.com/mgokbulut/MCL-Research