Efficient Monte Carlo Localization for Mobile Robots: Implementation and Evaluation

Kevin Eppacher, BSc.

***Abstract*—This paper presents an implementation and evaluation of a Monte Carlo Localization (MCL) algorithm for a Differential Drive Mobile Robot (DDMR), with a comparison to the widely used ROS Adaptive Monte Carlo Localization (AMCL) algorithm. The MCL algorithm demonstrates efficient localization using only 100 particles, significantly fewer than the 500 to 3000 particles typically used by AMCL. A novel resampling strategy, which includes generating 80% random particles, enhances the MCL's ability to quickly find the initial pose and recover from localization loss. Despite these improvements, MCL remains computationally intensive with increasing particle numbers. Future work includes averaging the strongest particle positions for smoother pose estimation and applying machine learning techniques to optimize sensor and motion model parameters.**

I. Introduction

Monte Carlo Localization (MCL), also known as the Particle Filter, is a probabilistic approach for estimating the pose of a robot within a given map [7]. Unlike other methods such as Bayesian filters, Kalman Filters (KF), and Extended Kalman Filters (EKF), MCL can handle non-linearities and multi-modal distributions, making it highly effective in complex environments [1].

The primary advantage of MCL over Bayesian filters, KF, and EKF lies in its ability to represent a diverse range of possible poses through a set of weighted particles, allowing for greater flexibility and robustness in localization tasks [3]. This adaptability makes MCL particularly suitable for dynamic and unpredictable environments.

However, MCL also faces challenges, particularly in defining accurate sensor models for different types of robots, such as Differential Drive Mobile Robots (DDMR). Developing precise sensor models is crucial for the performance of MCL, as inaccuracies can lead to significant localization errors [5].

II. State of the Art

The Monte Carlo Localization (MCL) algorithm relies heavily on the accuracy of its sensor models. While Thrun's beam range finder model is widely used and effective [7], several other sensor models have been developed to enhance the performance of Particle Filters.

One notable model is the likelihood field model proposed by Thrun et al., which precomputes a likelihood field for range measurements. This model improves computational efficiency and accuracy by using precomputed values to quickly evaluate the likelihood of measurements [7].

Another advanced sensor model is the end-point model, which evaluates the likelihood of the end points of the laser beams instead of the entire beam path. This model simplifies the computation and is particularly useful in environments with sparse obstacles [5].

Additionally, machine learning techniques have been applied to develop data-driven sensor models. For instance, a support vector learning-based particle filter has been proposed for underwater acoustic sensor networks, which enhances localization accuracy by dealing with distorted data using least-square support vector regression (LSSVR) [6]. Another approach uses convolutional neural networks (CNNs) to learn the sensor model parameters, providing a more adaptable and accurate model for various environments [2].

These advanced sensor models, along with Thrun's beam range finder model, contribute to the robustness and adaptability of MCL in various robotic applications. This work contributes to the further improvement in the implementation of the particle filter, regarding localization accuracy, computation and robustness.

III. Materials and Methods

This section explains the methods used to carry out the experiments. The inputs for the Monte Carlo Localizer (MCL), also known as Particle Filter, are the initial particles, the map, the motion commands and the laser scan measurement from the mobile robot. In the first line of the algorithm 1, the particles must be initialized with a position and a weight. In this work, the position of the initial particles was randomly generated on unoccupied grids of the map. After the first loop, the initial particles are the resampled particles. The pose of the particles is randomly predicted based on the motion command and the previous position of the particles (Algorithm 1, line 3). The particles are then evaluated by the sensor model in line 4.

The more likely the randomly predicted particle corresponds to the scan measurement, the greater the weighting. The predicted position and the corresponding particle are added to a list of particles. In the resampling step, a new set of particles X_t is created by selecting particles from $\bar{X}_t$ proportional to their weights (Algorithm 1, lines 7-10). This ensures that particles with higher probability are selected more often, which concentrates the particle set on the most probable positions. In addition, random particles are inserted to diversify the particle distribution (Algorithm 1, lines 11-15). This is done by randomly selecting cells c and drawing orientations θ from a uniform distribution between 0 and 2π. These random particles are added to the set of resampled particles to make the model more robust against uncertainties. Finally, the set X_t, which represents the estimated position of the robot, is returned (Algorithm 1, line 16). The MCL Algorithm was taken and modified from [7].

Algorithm 1 MCL(X_{t-1}, u_t, z_t, m)

1: $\bar{X}_t = X_t = \emptyset$
2: **for** $m = 1$ to M **do**
3: $\quad x_t^{[m]} =$ sample_motion_model$(u_t, x_{t-1}^{[m]})$
4: $\quad w_t^{[m]} =$ measurement_model$(z_t, x_t^{[m]}, m)$
5: $\quad \bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: $ResampledParticles$ = Resampling($\bar{X}_t$)
7: **for** $m = 1$ to $ResampledParticles$ **do**
8: $\quad$ draw i with probability $\propto w_t^{[i]}$
9: $\quad pose_t^{[i]} \leftarrow pose_t^{[i]} + \mathcal{N}(0, \Sigma)$ ▷ Add normal distributed noise to pose
10: $\quad$ add $pose_t^{[i]}$ to X_t
11: **for** j = 1 to $numRandomParticles$ **do**
12: $\quad$ Sample a cell: $c \sim \mathcal{U}$
13: $\quad$ Generate random orientation: $\theta \sim$ Uniform$(0, 2\pi)$
14: $\quad$ Create particle $p = (c_x, c_y, \theta, w)$, where $w = \frac{1}{\text{numParticles}}$
15: $\quad$ Add p to X_t
16: **return** X_t

Algorithm 2 predicts the pose of the particles. First, random noise is added to the linear and angular velocities from the motion commands using a normal distribution sampling function (Algorithm 2, lines 1-3). [7] did not consider the division by zero if $\hat{\omega}$ converges to zero in his Motion Model. Therefore, if $\hat{\omega}$ is significantly greater than zero, the motion model results in angular motion (Algorithm 2, lines 4-5); otherwise, the robot's motion is linear.

To evaluate the predictions of the particles, a Sensor Model is required to assess the accuracy of the particles' poses. In this work, the beam range finder model was used from [7]. Algorithm 4 calculates the probability of each scan measurement, by multiplying the probabilities of each laser

Algorithm 2 sample_motion_model_velocity(u_t, x_{t-1})

1: $\hat{v} = v +$ sample$(\alpha_1|v| + \alpha_2|\omega|)$
2: $\hat{\omega} = \omega +$ sample$(\alpha_3|v| + \alpha_4|\omega|)$
3: $\hat{\gamma} =$ sample$(\alpha_5|v| + \alpha_6|\omega|)$
4: **if** $|\hat{\omega}| >> 0$ **then**
5: $\quad x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t)$
6: $\quad y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t)$
7: **else**
8: $\quad x' = x + \hat{v} \cos\theta \Delta t$
9: $\quad y' = y + \hat{v} \sin\theta \Delta t$
10: $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
11: **return** $x_t = (x', y', \theta')^T$

beam of the scan measurement. In order to calculate the probabilities of each laser beam, four different probability density functions are needed. The mean of the probability density functions (PDF) are the range of the particles and the random variable is the range of the scan measurement. Imaginable by using the scan measurement laser beams as a template and comparing it with each particle. If the scan measurement and the particles raycast measurement overlap, the probability is high. The raycast Algorithm 3 calculates the ranges of each scan with the pose of the particle and the map. Depending on the map resolution, the range increments until ich hits a occupied cell or is out of map boundary.

Algorithm 3 raycasting(z_t, x_t, m)

1: δ = $map\_resolution$
2: $W = map.info.width$, $H = map.info.height$
3: $x_0 = x_t\_x$, $y_0 = x_t\_y$
4: **for** $k = 1$ to K **do**
5: $\quad \theta_k = \theta_p + z_t.angle\_min + k \cdot z_t.angle\_increment$
6: $\quad$ **while** $(r < r_{max})$ **and** $hit\ is\ false$ **do**
7: $\quad\quad r = r + \delta$
8: $\quad\quad x = x_0 + r\cos(\theta_k)$
9: $\quad\quad y = y_0 + r\sin(\theta_k)$
10: $\quad\quad map_x = \left\lfloor \frac{x - map.info.origin.position.x}{\delta} \right\rfloor$
11: $\quad\quad map_y = \left\lfloor \frac{y - map.info.origin.position.y}{\delta} \right\rfloor$
12: $\quad\quad$ **if** $map_x < W$ **and** $map_y < H$ **then**
13: $\quad\quad\quad index = map_y \cdot width + map_x$
14: $\quad\quad\quad$ **if** $mapatindexoccupied$ **then**
15: $\quad\quad\quad\quad hit = true$
16: $\quad\quad$ **else**
17: $\quad\quad\quad$ **break**
18: $\quad Rays \leftarrow Rays \cup r$
19: **return** $Rays$

The probability density function in Equation 1 evaluates the likelihood of a measured laser scan distance z_t^k given the particle position $\mathbf{x}_t$ and the map m.

Algorithm 4 beam_range_finder_model(z_t, x_t, m)

$q = 1$
for $k = 1$ to K **do**
  compute z_t^{k*} for the measurement z_t^k using ray casting
  $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
  $+z_{\text{max}} \cdot p_{\text{max}}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
  $q = q \cdot p$
return q

$$p_{\text{hit}}(z_t^k \mid \mathbf{x}_t, m) = \begin{cases} \eta \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) & \text{if } 0 \le z_t^k \le z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Here, $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)$ is a normal distribution with mean z_t^{k*} (expected measurement) and variance σ_{hit}^2. The factor η in Equation 2 normalizes the distribution so that the total probability sums to 1. In this work, the integral is solved numerically by the trapezoidal rule, which is demonstrated in Algorithm 5.

$$\eta = \left(\int_0^{z_{\text{max}}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)\, dz_t^k \right)^{-1} \tag{2}$$

This function is relevant only in the range $0 \le z_t^k \le z_{\text{max}}$ and models the hit probability of a laser scan. Changing σ_{hit} affects the spread of the normal distribution: a larger σ_{hit} results in a wider distribution, indicating more uncertainty in the measurements, while a smaller σ_{hit} results in a narrower distribution, indicating higher confidence in the measurements.

The probability density function $p_{\text{short}}(z_t^k \mid \mathbf{x}_t, m)$ evaluates the likelihood of a measured distance z_t^k being shorter than the expected measurement z_t^{k*}.

$$p_{\text{short}}(z_t^k \mid \mathbf{x}_t, m) = \begin{cases} \eta \lambda_{\text{short}} e^{-\lambda_{\text{short}} z_t^k} & \text{if } 0 \le z_t^k \le z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Here, λ_{short} is a parameter of the exponential distribution, and η normalizes the distribution:

$$\eta = \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}} \tag{4}$$

Increasing λ_{short} makes the distribution decay faster, implying that shorter distances are more likely, while decreasing λ_{short} makes the distribution decay slower, implying that longer distances are more likely within the range $[0, z_t^{k*}]$.

The probability function $p_{\text{max}}(z_t^k \mid \mathbf{x}_t, m)$ models the likelihood that the measurement z_t^k equals the maximum range z_{max}.

$$p_{\text{max}}(z_t^k \mid \mathbf{x}_t, m) = I(z_t^k = z_{\text{max}}) = \begin{cases} 1 & \text{if } z_t^k = z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

This function is a simple indicator function that assigns a probability of 1 if the measurement is exactly the maximum range and 0 otherwise. Changing z_{max} directly affects the threshold at which this function assigns a non-zero probability.

The probability density function $p_{\text{rand}}(z_t^k \mid \mathbf{x}_t, m)$ models random measurements within the range $[0, z_{\text{max}})$.

$$p_{\text{rand}}(z_t^k \mid \mathbf{x}_t, m) = \begin{cases} \frac{1}{z_{\text{max}}} & \text{if } 0 \le z_t^k < z_{\text{max}} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This function is a uniform distribution within the range $[0, z_{\text{max}})$. Increasing z_{max} decreases the probability density (since the total probability must sum to 1), while decreasing z_{max} increases the probability density within the specified range.

The parameters for the MCL are given in Table 1, in which the particles quantity, sensor model parameters and motion model parameters are compared with the AMCL algorithm from ROS [4]. These parameters are empirically tested, in regard of computational performance and localization performance.

IV. Experimental Results

In this experiment, a Turtlebot navigates in a circular path using MoveBase and AMCL for localization. Concurrently, our MCL (Particle Filter) implementation runs, visualizing all particles and publishing the pose of the most probable particle. The objective is to compare the accuracy of our MCL against the ground truth and AMCL.

Figure 1 illustrates the visualization of the raycast laser beams and the measured scan beams. Even within the simulation environment, laser scan measurements exhibit errors. These errors manifest when the laser beams either miss an occupied cell, such as a wall, or incorrectly pass through a wall. The scan serves as a reference template, against which the raycasts from each particle are compared. When the simulated raycasts align with the actual scan measurements, the probability weight assigned to the corresponding particle is maximized.

The performance of the Particle Filter is further evaluated by comparing the estimated positions with the ground truth. Figure 2 shows the x and y positions of the Turtlebot over time. The red and orange lines represent the ground truth positions, the green and purple lines indicate the estimated positions by the MCL, and the blue lines show the Euclidean error between the estimated positions and the ground truth. The mean deviation observed

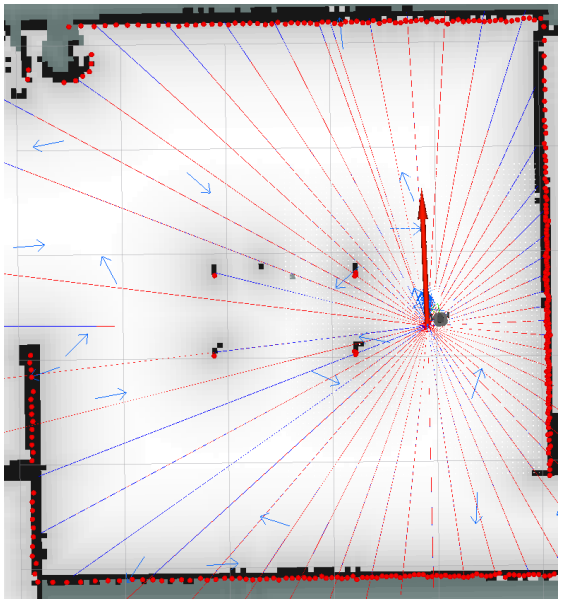

Fig. 1: Visualization of the raycast beams and the scan beams.

is 0.3 meters, with a minimum deviation of 0.03 meters and a maximum deviation of 0.6 meters.

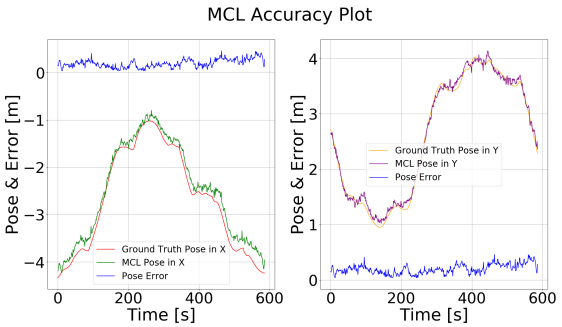

Fig. 2: MCL accuracy plot showing the ground truth positions, estimated positions, and the Euclidean error.

Figure 3 shows the comparison between the ground truth and estimated pose in a 2D space, highlighting the accuracy of the estimated path.

The particle distribution at a specific instance is illustrated in the histogram in Figure **??**, highlighting the spread and concentration of particles. This example histogram demonstrates the distribution of weights across 10 particles, showing the number of particles that fall into each weight bin.

A key highlight of our MCL implementation is the efficient use of particles. Compared to AMCL, which utilizes between 500 and 3000 particles, our MCL achieves accurate localization with only 100 particles. This efficiency is enhanced by our resampling strategy, where 50% of the particles are randomly generated on the map and only 50% are resampled. This approach helps the algorithm

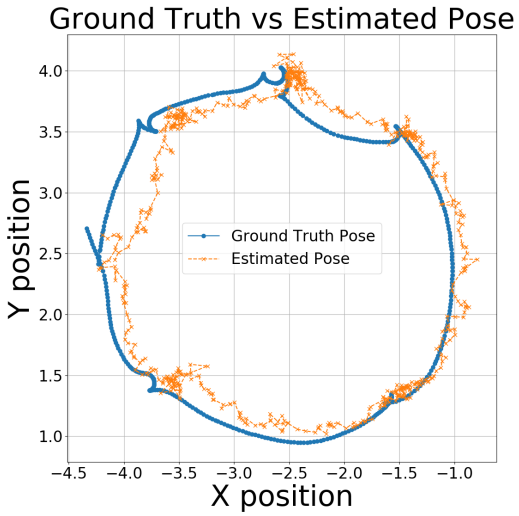

Fig. 3: Ground Truth vs Estimated Pose.

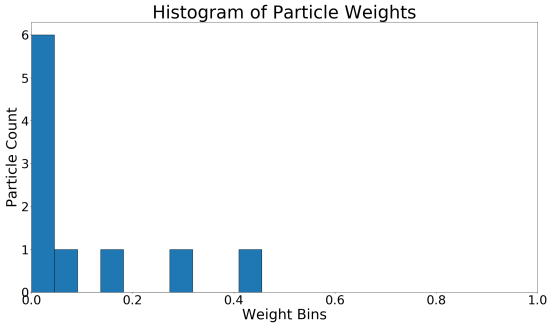

Fig. 4: Particle Histogram.

quickly find the initial pose due to the random particles and recover rapidly if the position is lost.

Lastly, 5 visualizes the DDMR with a set of pose arrays, calculated from the particle filter. The red arrow represents pose of the DDMR, which is calculated with the mean of the resampled particles.

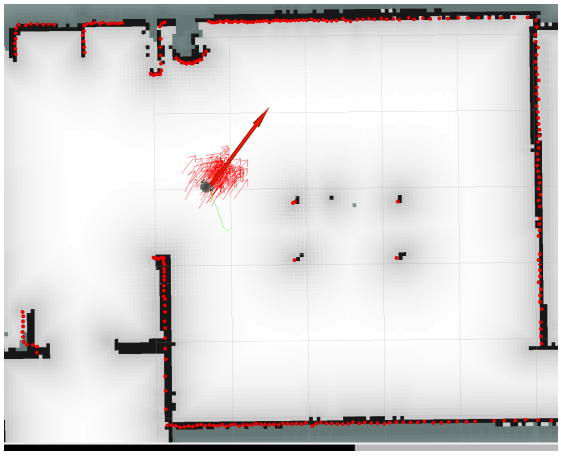

Fig. 5: Visualization of the pose array from the particle filter

V. Summary and Outlook

In this work, a MCL algorithm was implemented and evaluated for a Turtlebot, with its performance compared against the widely used ROS AMCL algorithm. Despite the efficiency improvements in the MCL implementation, particularly with the use of only 100 particles compared to AMCL's 500 to 3000 particles, the particle filter remains computationally intensive, especially as the number of particles increases beyond 500.

To further enhance the accuracy and smoothness of pose estimation, future work could involve averaging the positions of the strongest particles. This approach would result in a more accurate and smoother pose estimation, which is particularly beneficial for applications such as Model Predictive Control (MPC), where smoother predictions can lead to better performance.

Additionally, machine learning techniques could be employed to optimize the sensor model and motion model parameters. By using data-driven approaches, it is possible to more accurately identify these parameters, potentially improving the overall performance and robustness of the localization algorithm.

References

[1] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1322–1328 vol.2.

[2] M. Eder, M. Reip, and G. Steinbauer, "Using particle filter and machine learning for accuracy estimation of robot localization," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, ser. Lecture Notes in Computer Science, F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali, Eds. Springer, Cham, Jun. 2019, pp. 700–713, 32nd International Conference on Industrial, Engineering amp; Other Applications of Applied Intelligent Systems, IEA/AIE 2019 ; Conference date: 09-07-2019 Through 11-07-2019.

[3] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," 10 2001.

[4] W. Garage, "Adaptive monte carlo localization (amcl)," http://wiki.ros.org/amcl, 2016, accessed: 2023-06-29.

[5] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[6] X. Li, C. Zhang, L. Yan, S. Han, and X. Guan, "A support vector learning-based particle filter scheme for target localization in communication-constrained underwater acoustic sensor networks," *Sensors*, vol. 18, no. 1, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/1/8

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

APPENDIX

Algorithm 5 Numerical integration using the trapezoidal rule

function NUMERICALINTEGRATION(mean, z_max_range, num_steps)
  $step\_size = \frac{z\_max\_range}{num\_steps}$
  $integral = 0.0$
  for $i = 0$ to num_steps **do**
    $z = i \cdot step\_size$
    $weight = \begin{cases} 0.5 & \text{if } i = 0 \text{ or } i = num\_steps \\ 1.0 & \text{otherwise} \end{cases}$
    $integral = integral + weight \cdot \text{normalDistribution}(z, mean)$
  $integral = integral \cdot step\_size$
  return $integral$

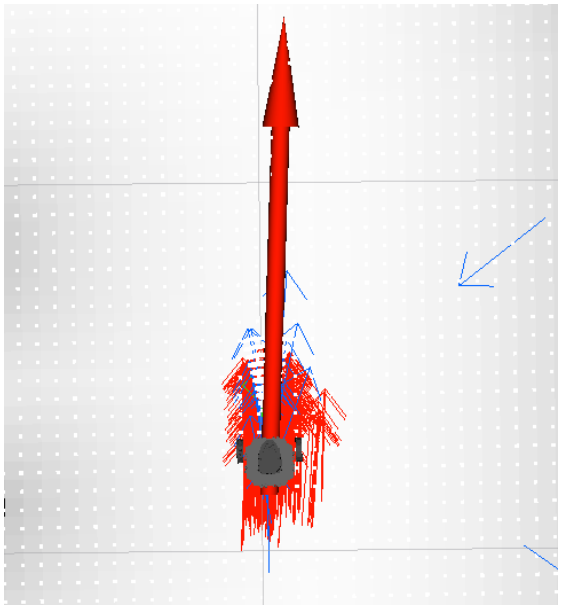

Fig. 6: Visualization of the particles from ROS AMCL and our MCL implementation.

Parameter	MCL	AMCL
z_hit	0.5	0.5
z_short	0.05	0.05
z_max	0.05	0.05
z_rand	0.5	0.5
sigma_hit	0.2	0.2
lambda_short	0.1	0.1
Particle Quantity	100	min_particles: 500, max_particles: 3000
Percentage random particles	0.5	N/A
perc_rays	20	N/A
alpha1	0.1	0.1
alpha2	0.1	0.1
alpha3	0.1	0.1
alpha4	0.1	0.1
alpha5	0.1	N/A
alpha6	0.1	N/A
laser max range	N/A	3.5
laser max beams	N/A	180

Tab. 1: Comparison of MCL and ROS AMCL [4] parameters