LOGO-generic-web-eps-converted-to.pdf

Monte Carlo Localization of mobile robot

J. Sagild, J. Johansen, M. Knædal, S. Koivumaki

***Abstract*— This project investigates real time estimation of a mobile robot's position and orientation (pose) by the Monte Carlo Localization (MCL) algorithm. The method has been implemented with Robot Operating System (ROS) tested on a mobile robot, equipped with a laser range finder. Five experiments were carried out to evaluate the implementation. The experiments gave satisfying results regarding the accuracy and robustness of the MCL implementation.**

***Index Terms*— Mobile robots, Localization, Particle filter, Monte Carlo**

I. Introduction

Localization is one of the fundamental problems that has to be solved for autonomous systems. The project described in this report aims to solve the global localization problem using the Monte Carlo Localization algorithm (MCL). The problem is solved using Robot Operating system (ROS), with a Pioneer 3DX robot and a Hokuyo URG-04LX laser range finder (LRF) equipped on the top of the robot. Firstly, MCL and the different sub-algorithms and methods used in the project will be described. Secondly, different experiments performed to test the implementation will be presented. At last, the results, a discussion, and a conclusion will be given.

II. Methods and Algorithms

A. Monte Carlo Localization

MCL is used to solve local and global localization problems in robotics. It can be considered as a recursive Bayes filter which estimates posteriors over robot poses by using particle filters. When implementing MCL it is essential to understand the main principles of the Bayes filter and particle filters in general. In the localization problem the robot's state x_t can be considered as the robot's pose at time t as defined in (1)

$$x_t = (x_t, y_t, \theta_t), \tag{1}$$

where x_t and y_t is the robot's position and θ_t is the robot's orientation. The Bayes filter represent the belief $bel(x_t)$ of the robot's current state x_t based on functions of the previous state x_{t-1}, actions u_t and measurements (z_t). The belief of the state can be expressed in two steps: prediction (2) and update (3):

$$\overline{bel}(x_t) = \int_{x_{t-1}} p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \tag{2}$$

$$bel(x_t) = \mu p(z_t|x_t)\overline{bel}(x_t) \tag{3}$$

The prediction of the belief of the current state $\overline{bel}$ is based on the probability of getting to the current state x_t when executing actions u_t to the the previous state x_{t-1}. In our case, action vector u_t includes the robot's odometry information. The current belief of the state is defined in the update step. The prediction of the current belief is updated by the probability that the particular state x_t belongs to the current measurements, z_t. In our case measurement data z_t is received from the LRF.

The particle filter is an non-parametric implementation of the Bayes filter in which the belief of the current state is represented by a set of multiple random samples X_t (4):

$$X_t := x_t^1, x_t^2, ..., x_t^M, \tag{4}$$

where M is the number of particles and each particle represent a possible location of the robot. Like in the Bayes filter, the particle filter consists of steps that can be related to the prediction and update step iterated for M particles x_t in the particle set X_t as follows:

$$x_t^m \approx p(x_t|x_{t-1}^m, u_t) \tag{5}$$

$$w_t^m = p(z_t|x_t^m) \tag{6}$$

The set of particles resulting from iterating equation (5) M times is the filter's representation of $\overline{bel}(x_t)$. The importance factor, represented by the weight w_t, is defined for each particle in that set by computing the probability given by equation (6). The set of weighted particles is the filter's representation of $bel(x_t)$. The calculated set of particles and weights related to them forms a temporary set of particles $\overline{X_t}$ that might be used for resampling. In the resampling the particle filter algorithm draws particles with replacement from the temporary set $\overline{X_t}$. The probability of drawing each particle is given by its weight. Particles drawn in the resampling forms the current set of particles, X_t, representing the belief of the current state.

MCL is a particle filter in which equation (5) is calculated with a motion model and equation (6) with a measurement model, represented more detailed in the following sections. [1]

Monte-Carlo-Localization(X_{t-1}, u_t, z_t m)**:**
$\overline{X}_t = X_t = 0$
for $m = 0$ *to* M **do**
  x_t^m = **sample-motion-model-odometry**(u_t, x_{t-1}^m)
  w_t^m = **measurement-model**(z_t, x_t^m, m)
  $\overline{X}_t = \overline{X}_t + (x_t^m, w_t^m)$
end
for $m = 0$ *to* M **do**
  draw i with probability $\propto w_t^m$
  add x_t^m to X_t
end
return X_t

Algorithm 1: Monte-Carlo-Localization algorithm

III. Implementation

Implementation is done using ROS. ROS provides packages and tools for developing. A brief explanation of self-implemented and built-in packages that have been used will be given in this section.

- Self-implemented packages:
  - `tf_laser_to_base_link`
  - `monte_carlo`
  - `tf_mocap_to_pose`
- Built-in packages:
  - `slam_gmapping`
  - `hokuyo_node`
  - `rosaria`
  - `mocap`

– `teleop_twist_keyboard`

First of all, the self-implemented packages are written in Python and C++, while launch-files used to launch nodes are written in XML. Maps have been made using `slam_gmapping`, and a necessary transform from the LRF's frame to the robots frame, to create the map, is implemented in `tf_laser_to_base_link`.

Connecting the LRF and the robot has been done using the `hokuyo_node`- and the `rosaria` package, respectively. Final tests has been done using the Motion Capture system (MoCap), provided in the `mocap` package. To eliminate a static deviation in the `mocap` system, a static transform using a launch-file was made. Teleoperating the robot has been done using `teleop_twist_keyboard` package, to collect data through rosbags.

The MCL algorithm is implemented in the `monte_carlo` package. A computation graph while running the MCL algorithm is shown in figure 1. Visualization and plotting is done using Rviz and MATLAB.

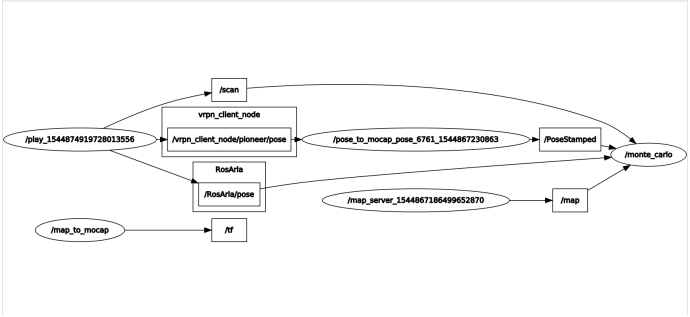

Fig. 1. Computation graph while running MCL algorithm. Generated using the built-in tool `rqt_graph`.

A. Motion model

We are using odometry as our motion model. Another option would have been to use a velocity model but it is generally considered more erroneous. On the other hand the advantage of using a velocity model would have been the information gained about the robot future positions. For the localization task that's not as relevant as reliability. The odometry information is obtained by integration the rotation of the wheels and it includes the information of robot's pose in its local coordinates. To simplify the problem we consider the odometry information as a control signal even though it is only available in retrospect. Another alternative would have been to model the odometry information as a measurement, but then we would also need to include velocity as a state variable, and this would lead to a higher dimension state-space, and thus increase the complexity.

The sample-motion-model-odometry algorithm (Algorithm 2) defines the prediction of the current position of the particle x_t based on the data of the previous particle x_{t-1} and the odometry data u_t and u_{t-1} which includes the pose information in the robot's local coordinates ($\overline{x}$, $\overline{y}$, $\overline{\theta}$) as follows 2 [1].

The movement of the robot is calculated using the rotation and translation where the rotation is divided in two parts: half of the total rotation angle before the change of robot's position, and the other half after the change of the robot position.

B. Measurement model

The measurement model is defined as a conditional probability distribution $p(z_t|x_t, m)$, where x_t is the robot pose given by the motion model, z_t is the measurements at time t and m is the map of the environment. This probability distribution describes how well

sample-motion-model-odometry (u_t, u_{t-1}, x_{t-1})**:**

$\delta_{rot1} = arctan2(\overline{y} - \overline{y*}, \overline{x} - \overline{x*}) - \overline{\theta*}$
$\delta_{trans} = \sqrt{(\overline{x} - \overline{x*})^2 + (\overline{y} - \overline{y*})^2}$
$\delta_{rot2} = \overline{\theta} - \overline{\theta*} - \delta_{rot1}$
$\overline{\delta_{rot1}} = \delta_{rot1} - \textbf{sample}(|\alpha_1\delta_{rot1} + \alpha_2\delta_{trans}|)$
$\overline{\delta_{trans}} = \delta_{trans} - \textbf{sample}(|\alpha_3\delta_{trans} + \alpha_4(\delta_{rot1} + \delta_{rot2})|)$
$\overline{\delta_{rot2}} = \delta_{rot2} - \textbf{sample}(|\alpha_1\delta_{rot2} + \alpha_2\delta_{trans}|)$
$x = x* + \overline{\delta_{trans}} \cos(\theta* + \delta_{rot1})$
$y = y* + \overline{\delta_{trans}} \sin(\theta* + \delta_{rot1})$
$\theta = \theta* + \overline{\delta_{rot1}} + \overline{\delta_{rot2}}$
return $x_t = (x, y, \theta)$

Algorithm 2: sample-motion-model-odometry

a set of measurements fits a given particle in an environment, and represent the weight each particle is given in the MCL-algorithm.

We used the beam-range-finder-model algorithm (Algorithm 3) to compute this probability distribution.

beam-range-finder-model(z_t, x_t, m)**:**
$q = 1$
for *each measurement* z_t^k *in measurements* z_t **do**
  z_t^{k*} = ray-casting(x_t, θ_t^k, m)
  $p = a \cdot p_{hit} + b \cdot p_{max}$
  $q = q \cdot p$
end
return q

Algorithm 3: beam-range-finder-model algorithm

Our beam-range-finder-model algorithm incorporates two types of error measurements: local measurement error, p_{hit}, and error due to failing to detect objects, p_{max}. We could include other types of error measurements, but our algorithm worked very well only using these two so we did not bother.

1) Local measurement error: If the measurement value z_t^k is greater or equal to the maximum laser range, z_{max}, the local measurement error, p_{hit} is set to zero. If not, the measurement error is modeled using a narrow Gaussian with mean z_t^{k*} and standard deviation σ:

$$p_{hit} = \eta \mathcal{N}(z_t^k; z_t^{k*}, \sigma^2).$$

where

$$\eta = (\int_0^{z_{max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma^2)\, dz_t^k)^{-1}$$

and works as a normalizer. The mean, z_t^{k*}, denotes the "true" range of a measurement z_t^k . This "true" range is computed using ray casting (Algorithm 4).

ray-casting(x_t, θ_t^k, m)**:**
$grids$ = beresenham(x_{start}, y_{start}, x_{end}, y_{end})
for *each grid in grids* **do**
  if *grid is occupied* **then**
    return distance to grid
  end
end

Algorithm 4: ray-casting algroithm

The ray-casting method calculates the distance from a particle's position to the closest occupied grid given the angle, θ_t^k, of a measurement z_t^k. The closest occupied grid is found using a variation

of Bresenhams algorithm (Algorithm 5). Our Bresenham variation draws a line between two grids, (x_{start}, y_{start}) and (x_{end}, y_{end}), and returns the set of grids this line run through. In this set the first grid is (x_{start}, y_{start}) and the last grid is (x_{end}, y_{end}) (Algorithm 4). The ray-casting method then returns the first, and also closest, occupied grid in this set.

(x_{start}, y_{start}) is derived from the pose, x_t, and represent the position the particle. (x_{end}, y_{end}) is computed using θ_t^k and z_{max}. It represent the grid with the biggest distance it's possible to measure for that measurement.

bresenham(x_{start}, y_{start}, x_{end}, y_{end}):
$dx = x_{start} - x_{end}$
$dy = y_{start} - y_{end}$
$swapped = False$
if $abs(dy) > abs(dx)$ **then**
  x_{start}, y_{start} = y_{start}, x_{start}
  x_{end}, y_{end} = y_{end}, x_{end}
end
if $x_{start} > x_{end}$ **then**
  x_{start}, x_{end} = x_{end}, x_{start}
  y_{start}, y_{end} = y_{end}, y_{start}
  calculate new dx and dy
  $swapped = True$
end
for *each* x *between* x_{start} *and* x_{end} **do**
  if $abs(dy) > abs(dx)$ **then**
    grid = (x, y)
  else
    grid = (y, x)
  end
  add $grid$ to $grids$
  $error = error - abs(dy)$
  if $error < 0$ **then**
    $y = y + y_{step}$
    $error = error + dx$
  end
end
if $swapped$ **then**
  reverse $grids$
end
return $grids$

Algorithm 5: Bresenham algorithm

The Bresenham algorithm first calculates the slopes, dx and dy. It then uses these values, as well as x_{start} and x_{end} to determine if and how to transform the coordinates so that we can iterate along the x-axis. For each iteration of the for-loop a $grid$ is added to $grids$ and the $error$ is updated. The $error$ has the initial value of $\frac{dx}{2}$ and indicates whether or not we should update y.

2) Failure to detect objects: The error related to failure to detect objects is much easier to compute than the one related to local measurement error. This error occurs when the laser fails to detect objects, which might occur when sensing black, light absorbing objects or when measuring objects in bright light. It will also occur when there are no objects within the laser range measure distance. In the eighth and fifth floor, which is where we have tested, there are a lot of open spaces so the LRF will fail to detect any objects quite often. The LRF will then return the maximum laser range value, z_{max}. This error is modeled using a point-mass distribution centered at z_{max}:

$$p_{hit} = \begin{cases} 1 & if\ z_t^k = z_{max} \\ 0 & otherwise \end{cases} \quad (7)$$

The intrinsic parameters of Algorithm 3, a and b, represents the weight of each error calculation, and the sum of these are equal to one. We quickly learned that the local measurement error is the most important, and a should therefor have a much higher value than b.

C. Estimating position from particles and error calculation

In this subsection the way of estimating the robot's pose and calculating the error is described. These methods are used in the experiments described later. The estimation of the robot's pose for each iteration is calculated as a mean of the pose of the ten best weighted particles before re-sampling. The position error between the estimated position and the ground truth position is calculated as absolute Pythagoras error. From now on this error is called the position error. The orientation error is the absolute between the estimated position and the ground truth. The orientation error is scaled between $[\pi{:}-\pi]$

IV. Experiments

To measure performance and evaluate the MCL algorithm, different tests have been performed. We used ground truth provided by the MoCap system to evaluate our estimated position. The data was obtained by running the robot in the eight floor of the North Tower. We collected three rosbags, one used to create a map, and two for conducting experiments of the localization. The trajectory of the robot is different in the different rosbags.

The experiments consist of running the MCL algorithm on a given rosbag 10 times. All the experiments are conducted with the same parameters. The parameters are tuned to minimize the error of rosbag 1. Five experiments are executed.

A. Experiment 1

Experiment 1 uses rosbag 1. The particles are randomly initialized in a box around the starting position. The box's center is at the initial ground truth position provided by MoCap. See figure 2. The length of the sides of the box is 1 meter. This experiment is performed to confirm that the MCL algorithm is able to localize with small error when given a good initial guess.

B. Experiment 2

Experiment 2 uses rosbag 1. The particles are randomly initialized within the free-space of the map. See figure 3. The goal of the experiment is to prove that the algorithm is able to solve a global localization problem with a initial uniform distribution of the particles.

C. Experiment 3

Experiment 3 uses rosbag 2. The particles are randomly initialized in a box around the starting position. The box's center is at the initial ground truth position provided by MoCap. The length of the sides of the box is 1 meter. This experiment is similar to experiment IV-A. The goal is to prove that the chosen parameters are not overfitted to only perform well on rosbag 1.

D. Experiment 4

Experiment 4 uses rosbag 2. The particles are randomly initialized within the free-space of the map. This experiment is similar to experiment 2. The goal is to show that MCL can solve a localization problem with global initialization with a new trajectory.

E. Experiment 5

Experiment 5 uses a new map, which is a map of fifth floor (figure 4) in North Tower, and a complete ”unseen” rosbag. This means that the robot have never been tested on this rosbag before. Since we use a new environment, the ground truth provided by MoCap cannot be used. To know the actual position, we recorded and took notes of the approximate position. This experiment is performed to show that the parameters are not overfitted for the map of the eight floor, and to show that it is able to converge in different environments. Finally it is to show that MCL is able to solve a general global localization problem.

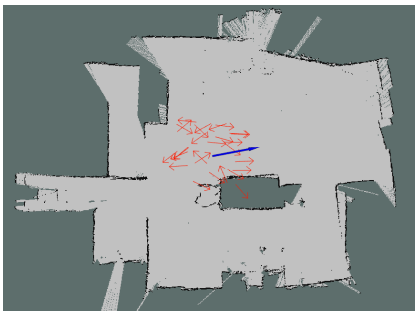

Fig. 2. Box initialization used in experiment 1 and 3. Print screen from Rviz. 500 particles are randomly initialized in a box around the starting position. The box's center is at the initial ground truth position provided by the MoCap.

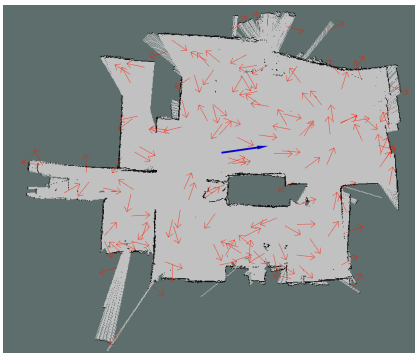

Fig. 3. Initialization used in experiment 2 and 4. Print screen from Rviz. The particles are randomly initialized within the free-space of the map.

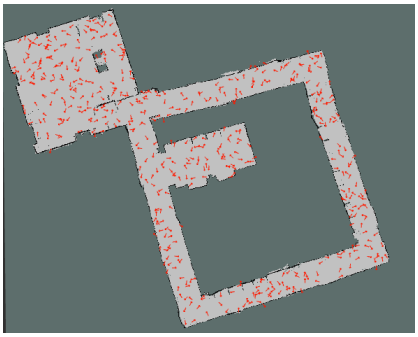

Fig. 4. Initialization used in experiment 5. Print screen from Rviz. The particles are randomly initialized withing the freespace of the map

F. Evaluation of the experiments

To evaluate the experiments we can examine the mean and standard deviation of the position relative to the ground truth provided by MoCap, of the 10 runs we performed for each experiment. The reason each test is performed 10 times is because of the randomness introduced by MCL. By looking at the mean error we can conclude on how accurate it is, and by looking at the standard deviation we can conclude on how consistent it is.

By comparing the results from Experiment 1 and 3, and 2 and 4, we can conclude whether the parameters are overfitted. By looking at experiment 2 and 4 we can conclude whether we are solving a general global localization problem or not. Finally, by looking at experiment 5 we can conclude on whether the MCL algorithm is able to operate in a new unseen environment.

V. Results

In this section, final parameters used to obtain the results from the experiments, as well as results from the different experiments, described in IV will be represented.

TABLE I

Final results of the Mean pose error calculated from running the respective rosbag used in each experiment 10 times. STD is standard deviation. Position is presented in meters, while orientation is presented in radians.

	Position		Orientation	
	Mean	STD	Mean	STD
Experiment 1	0.37	0.10	0.04	0.20
Experiment 2	2.20	0.52	-0.09	0.42
Experiment 3	0.40	0.18	0.25	0.50
Experiment 4	1.77	0.98	0.25	0.66

A. Parameters

The extrinsic parameters and the intrinsic parameters tuned on rosbag 1 are represented in table II.

TABLE II

The given parameters refer to constants introduced section III-A and III-B. $\boldsymbol{\alpha_1}$, $\boldsymbol{\alpha_2}$, $\boldsymbol{\alpha_3}$ and $\boldsymbol{\alpha_4}$ refers to constants in the odometry model, and sigma refers to a constant in the measurement model.

Extrinsic parameters	
Number of particles	100
Number of measurements	4
Loop time	0.3
Intrinsic parameters	
α_1	0.00025
α_2	0.30000
α_3	0.30000
α_4	0.00025
sigma	1.50000
threshold	90.90910

B. Experiment 1

The results obtained from Experiment 1 is shown in figure 5.

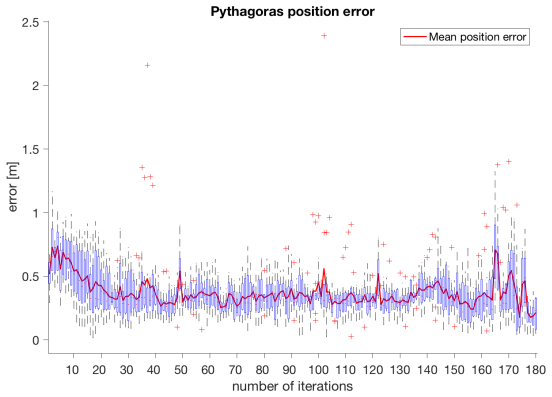

Fig. 5. The figure shows data from Experiment 1. It shows a boxplot of position errors plotted against the number of iterations, running rosbag 1, 10 times. Iteration time equals 0.3 seconds. There is a unique box for each iteration. The blue boxes represents 75% of the data (3 interquartile range), and the red crosses are outliers. The red line is the mean position error.

C. Experiment 2

The results obtained from Experiment 2 is shown in figure 6.

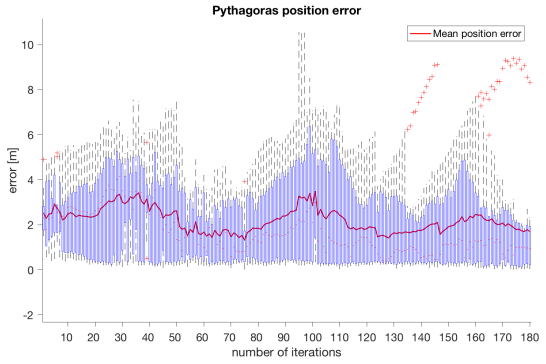

Fig. 6. The figure shows data from Experiment 2. It shows a boxplot of position errors plotted against the number of iterations, running rosbag 2, 10 times. Iteration time equals 0.3 seconds. There is a unique box for each iteration. The blue boxes represents 75% of the data (3 interquartile range), and the red crosses are outliers. The red line is the mean position error.

D. Experiment 3

The results obtained from Experiment 3 is shown in figure 7.

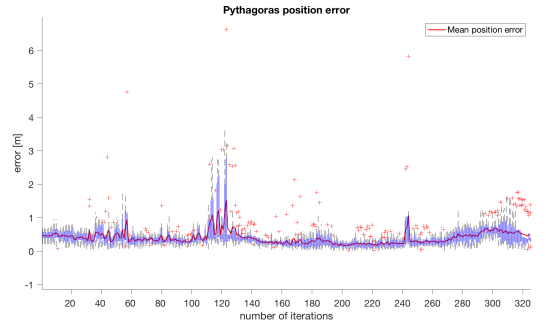

Fig. 7. The figure shows data from Experiment 3. It shows a boxplot of position errors plotted against the number of iterations, running rosbag 1, 10 times. Iteration time equals 0.3 seconds. There is a unique box for each iteration. The blue boxes represents 75% of the data (3 interquartile range), and the red crosses are outliers. The red line is the mean position error.

TABLE III

The table shows the time used to converge for each run. If the error is less than 0.5 meters for more than 10 iteration, it has converged in our terms. The time is given in seconds. 'Converged' is True if it converged, and False if it did not. Data is obtained from bag 1, using global initialization.

	Convergence time	Converged
Run 1	0	False
Run 2	40	True
Run 3	0	False
Run 4	43	True
Run 5	0	False
Run 6	0	False
Run 7	4	True
Run 8	0	False
Run 9	6	True
Run 10	16	True

E. Experiment 4

The results obtained from Experiment 4 is shown in figure 8.

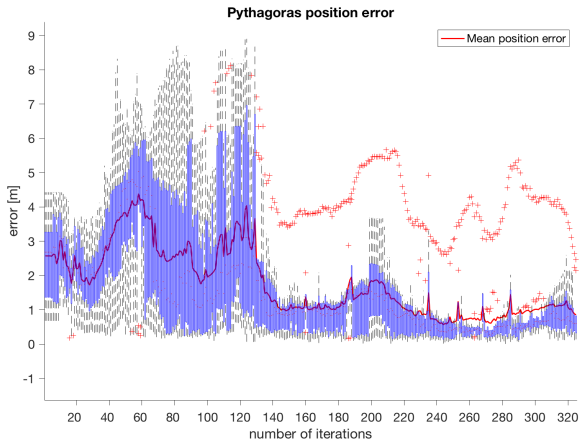

Fig. 8. The figure shows data from Experiment 4. It shows a boxplot of position errors plotted against the number of iterations, running bag 2, 10 times. Iteration time equals 0.3 seconds. There is a unique box for each iteration. The blue boxes represents 75% of the data (3 interquartile range), and the red crosses are outliers. The red line is the mean position error.

TABLE IV

The table shows the time used to converge for each run. If the error is less than 0.5 meters for more than 10 iteration, it has converged in our terms. The time is given in seconds. 'Converged' is True if it converged, and False if it did not. Data is obtained from rosbag 2, using global initialization.

	Convergence time	Converged
Run 1	0	False
Run 2	4	True
Run 3	98	True
Run 4	38	True
Run 5	152	True
Run 6	146	True
Run 7	91	True
Run 8	0	False
Run 9	155	True
Run 10	158	True

F. Experiment 5

The results obtained from Experiment 5 is shown in table V.

TABLE V
The percentage of correct convergence after 10 runs when the number of particles varies.

Number of particles	Percentage convergence
100	0.2
500	0.8

VI. Discussion

In this section the results represented in section V will be discussed and evaluated.

A. Experiment 1 and 3

We can see from figure 5 and figure 7 that our MCL algortihm is able to localize with a small error when given a good initialization. As shown in table III the average position error for all iterations is 0.37 meters for experiment 1 and 0.4 for experiment 3. The standard deviations is 0.1 meters and 0.18 meters for experiment 1 and experiment 3, respectively. This shows that our method is consistent. It also shows that our algorithm predicts similar positions each run. The difference between the values for the two experiments might be a result of the length and trajectory of the respective rosbags.

In addition to the position, the error of the orientation needs to be reviewed. In experiment 1 both the average orientation error and it's standard deviation are very small, only 0.04 radians and 0.20 radians respectively, as shown in table III. For experiment 3 those are remarkably higher: the average is 0.25 radians and the standard deviation is 0.50 radians.

Rosbag 2 is longer than rosbag 1, and the trajectory of rosbag 2 is more challenging. Rosbag 2 includes many turns, which can be challenging for the odometry. It also moves into every part of the map, which contains a lot of open space. This can be challenging for the measurement model. It is therefore reasonable that the mean and standard deviation of the position and the orientation error is bigger in experiment 3 than in experiment 1.

This result verify that our algorithm is capable of providing a good localization estimation on two different rosbags, and that it is not overfitted for one rosbag.

With the given mean positional errors and standard deviations, it is reasonable to assume that we have a small static error. This might be due to inaccurate transformations of measurements, or inaccuracies in the setup of MoCap.

B. Experiment 2 and 4

From table III we can see that MCL is able to localize 50% of the times when given a global initial uniform distribution of the particles, as seen from experiment 2. The average position error for all iterations is 2.20 meter and the standard deviation is 0.52 meters. Both are higher than for experiment 1 and 3, shown in figure 6. The average orientation error in experiment 2 is -0.09 radians and it's standard deviation is 0.20 radians. These results are surprisingly good when compared to the position error of the same experiment and the orientation error in experiment 3.

In experiment 4, we can see from table IV that MCL is able to localize 80% of the runs. The average position error for all iterations is 1.77 meter and the standard deviation is 0.98 meters. The Mean value of the position error is lower than in experiment 2, as shown in figure 7. This is surprising, since the parameters are tuned for rosbag 1. The average orientation error is 0.25 radians and its standard deviation is 0.66 radians. These results are remarkably weaker than in experiment 2 but correlates with the difficulties with the orientation localization in the experiment 3.

The reason why it only converges 50% of the runs in Experiment 2 can be because of the amount of particles used is to small. MCL localizes to the correct positions 30% more often with rosbag 1 than 2. The reason might be because rosbag 2 is longer than 1, respectively 320 and 180 iterations each, and it therefore has a larger probability to localize.

Based on the percentage of times MCL are able to localize in Experiment 2 and 4, we can conclude that it is not always able to localize it self with the given amount of particles. From figure 8 we can see that it is often able to localize after enough iterations (approximately 140 iterations). Then the mean position error stabilizes around 1 meter, and the standard deviation shrinks in. We can also see that there are some outliers, referring to the red crosses. This is an indicator that MCL is able to do global localization given enough time and particles.

C. Experiment 5

We can see from table V that in a new environment our algorithm struggles to localize with the same number of particles. There could be several reasons behind this. The rosbags duration is shorter, giving the robot less time to localize. Also, the map has different characteristics. Eighth floor contains more open space than fifth floor.

The algorithm has a higher performance if the number of particles is increased. With a higher number of particles, the probability of some particles being initialized with a pose close to the actual position of the robot is larger. Increasing the number of particles to 500 increases the probability of succeeding to 0.8. From this, we can conclude that MCL is often able to converge given enough particles. Note that the average run time of MCL increased from 0.1 to 0.5 when the amount of particles was increased.

VII. Conclusions

Through various experiments we have concluded that MCL can solve the local localization problem with a small error but the global localization problem is more challenging to solve. If the initial pose of some of the particles is close to the pose of the robot, the robot is able to localize consistently with a small error. On the other side, it will localize 50% to 80% of the times when none of the particles are initialized near the robot's initial pose, and it will take several iterations to converge. The solution, as concluded from experiment 5, is more particles to make it converge quicker and more often.

References

[1] S. Thrun, W. Burgard and D. Fox, "Probabilistic Robotics,", 2nd ed., Early Draft, 1999-2000. pp. 9–219.