Monte Carlo Localization: Efficient Position Estimation for MyThymio

Jose Ezequiel Castro Elizondo, AI Master Student, USI Lugano
Maria Andromachi Kolyvaki, AI Master Student, USI Lugano
Initial Submission

***Abstract*—This paper presents an implementation of the Monte Carlo Localization algorithm that is responsible for the localization of different types of mobile robots. The objective of our implementation is to achieve the localization of MightyThymioX model, in a maze, without knowing its initial pose (meaning the Cartesian coordinates and its orientation). We are going to use a simulated low-level MyT controller, on a world of maze that we constructed on Gazebo simulator. Our implementation demonstrates two flavors of models. One that omits the orientation of MyThimio and another one that takes into consideration of the sensors readings regarding the orientation.**

***Index Terms*—Monte Carlo Localization, MyThymioX, Particle Filtering, Robot motion, Sensor Readings, Bayes Rule, Randomized Sampling, Global Localization, Maze, Labyrinth.**

I. Introduction

One of the most salient subjects, surrounding the field of navigation in Robotics, has undoubtedly been the struggle for Map-based Localization of mobile robots, in a given indoor or outdoor environment. The notion of Map-based stands for our prior knowledge about the layout of the world we are moving in, with the exception of not knowing our exact position on that given map. In this paper we are going to use a rudimentary simple maze, which we have constructed in the Gazebo simulator, using grid of equidistant blocks. The scale of the maze was chosen in such a way so that the dimensions of MyThymio would not cause any further problems of manoeuvering in the given world.

Throughout the last years, many different approaches have been introduced regarding the reliable localization of mobile robots, with probabilistic approaches being the most prominent and in parallel promiscuous for the aforementioned objective. The building block of Monte Carlo Localization (MCL) is approximating the current belief that the robot has about its state, by keeping track of a subset of all states (which are called particles). The algorithm comprises by two stages (SEE and ACT), which we are going to analyze extensively in the Algorithm section. In general, Particle filtering provides a robust way of robot localization, although some times no matter how accurate and detailed the probabilistic model that we implement is, there are some some difficulties, because the robot's decision are made instantly and on real-time.

II. Overview of the Algorithm

In order to whet our appetite, for deeper understanding towards the MCL algorithm, it would be high time to provide the basic layout of the algorithm in parallel with a profound analysis of the steps it entails.

We will start from the basis of the Particle Filters which lies in the property of them, being a family of Sequential Monte Carlo algorithms. Monte Carlo modeling, relies on Bayesian formulation of the movement of MyThymio having in mind the prior knowledge regarding its previous position but none kind of knowledge regarding the topological properties of the environment. This aforementioned knowledge enables to formulate a continuously alternating subset of possible states of the robot, which are called particle filters. We start by a subset of particle filters that are randomly uniformly distributed throughout the whole map, that is already known, with all of them being equally likely candidates for the real pose for our robot. As a second step, the robot has to move and sense the environment which in our case is a maze that is comprised of equally scaled walls and two ways out of the maze. We are going to partition the algorithm that we followed in three distinct and well defined steps: MOVE, UPDATE, RE-SAMPLE. In the MOVE part, the robot starts moving for a given distance, acquiring a given random orientation and also the particles are moved in accordance with, the reached position of the robot, based on euclidean distance measurements. As a next step, we have to UPDATE the weights of the corresponding particle in such a way that they will demonstrate their relative position with respect to the robot, and as a logical consequence we should proceed to the RE-SAMPLE step. In this last step of our iterative algorithm, we have to choose only those particle filters that meet the range of a predefined threshold which determines the model's tolerance for keeping the fittest candidates.

III. MATHS

It would be high time to familiarize with the mathematical background of the aforementioned computations in order to make the transition from the theoretical infrastructure to the code implementation, as smooth as possible.

In reality, Monte Carlo methods are trying to approximate the partially observable Markov Chains that are produced while the robot is moving, with the use of Particle filters. These particle filters represent probabilistic hypothesis of the robot's position within the maze and it could be easily modeled in the following way.

Let us consider, at time t that the given state of the robot is x_t and after a unit of time Δt, which is a number that ranges between different implementations, the robot is at state $x_{t+\Delta t}$. By applying basic probabilistic probabilistic law, we conclude that the probability of the robot being at state $x_{t+\Delta t}$, after following instruction u_t given that it started in the position x_t, is given by: $P(x_{t+\Delta t}|u_t, x_t)$. As we are moving, exploring the space we could clearly understand that the Markovian states that we pass through are not observable and for that reason

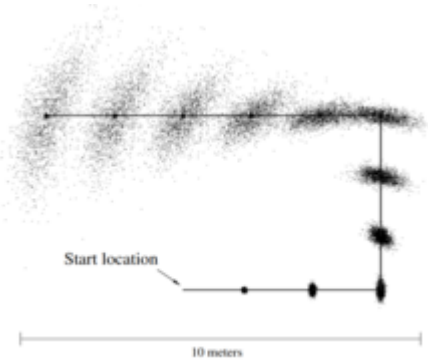

Figure 1. Sampling-based approximation of the belief of the robot about its position, without using sensors.

we should seek for a measurement that is going to provide us some rigid knowledge about our possible positions. This aforementioned knowledge comes form the sensor readings z_t which provide us with a summary of distance with respect to the obstacles, in order to formalize our belief with respect to the given map. Our belief could be derived via the basic Law of Probabilities $P(z_t|x_t)$.

Our initial objective was to compute the posterior probabilities of each state given both the odometry and the sensor readings. For that reason, in order to compute them we are going to base our computations to the Bayes Filter, since we already acquire knowledge about the prior and the initial probabilities according to the following formula.

$$P(x_t|u_t, z_t) = \alpha P(z_t|x_t) \int_{x_{t-1}} P(x_t|x_{t-1}, u_t) \\ P(x_{t-1}|u_{t-1}, z_{t-1}) dx_{t-1}$$

Where $P(x_{t-1}|u_{t-1}, z_{t-1})$ is the prior belief of its state. The motion model as the probability of $P(x_t|x_{t-1}, u_t)$ is referring to the odometry based reading as we mentioned previously.

IV. IMPLEMENTATION

We are going to start this detailed part of getting an insight on our implementation, by making a brief explanation of how we constructed the maze for MyThymio localization.

Firstly, we would like to mention that for our project, we have used Gazebo simulator, which is a pretty useful simulator for implementing our algortihm and checking the implementation on Mighty Thymio 10. To begin with, we used the *Gazebo model Database* that contains tailor made walls, which we modified further in order to have a fiducial copy of the initial labyrinth picture, we have found online and is represented in Fig. 2. The we exported the world in Gazebo's models directory as an .sdf file and we made the corresponding XML launch files in order to be able to launch our world from the terminal, exactly in the way that we did when we were launching any other default world contained in our initial directory of thymio_course_sceleton. As we are going to show later we had to create later a mapping of the world to a grid that had the same scale as our world (an 1-to-1 mapping from the 300x300 ppx picture to our virtual world which was 300x300 cm)

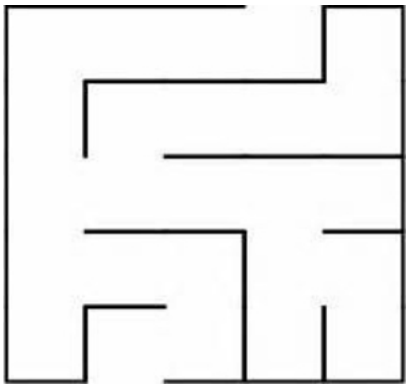

Figure 2. The labytinth that was chosen for our objective of localizing MyT inside it

A. Initial Model

In this first model we are basically solely based on the coordinates of the robot and the particles, omitting the orientation of the before-mentioned.
It would be high time to to begin our foray into the exploration of our first model. We will start with our `move` function which basically describes the movement of the robot for some units of time, in order to reach a given goal. In the `move` function we implement a random movement inside the maze and in case that we have to stop, in case of the observation of an obstacle we compute the euclidean distance of our starting point until the point we reached.

Most importantly, we have constructed a class named `Particle` which demonstrates the spawning of particles throughout the maze that basically are comprised of an id, in order to identify which of them we are going to keep in later steps, a pose, including the coordinates and the orientation, and lastly the weight that is assigned to each one of the particles. The last field is going to be of primal importance in later stages when we implement the re-sampling.

We can now proceed to the point that we are going to talk about the `Program` class which comprises the the crux of the matter of the first model. Our initial concern was to create a mapping form the picture of the maze to a grid in order to formulate a mathematical representation of the spawning of the particles and also in order to ensure the best control of the the coordinates of both the final pose of the robot and the pose of the particles. For that reason the function `buildWorld` implements the visualization of the picture with a predefined grid on top of that.

For the realization of the particle filtering we should initially spawn the the particles following a uniform random distribution with mean zero and a variance analogous to the images dimensions. In other words, if we denote the particles as p_i then we should have that the initial probability of each particle representing the belief of the robot's possible states could be

described by the following

$$x_i^{[t]} \sim \mathcal{N}\left((0,0),(\frac{width}{2},\frac{height}{2})\right)$$

As a second step, we have to map the particles to a corresponding grid square, for implementing the localization iteratively, by applying the Bayes filtering that we introduced before.

As a next step, we introduced the two following functions `check_for_obstacles` and `get_distance_to_all_obstacles` that are going to check for the presence of an obstacle and finding the distances to all these obstacles, which in our cases are translated as the walls of the maze. As we have based our whole implementation on a mapping from the given picture to the grid, we also translated all the walls as an array comprised of arrays of coordinates corresponding to the starting and the ending points or each wall of our world.

Since we "axiomatized" our implementation with some initial steps we should now proceed to the building block of our it, where we get our hands on the application of the three steps of the Monte Carlo Filtering.

We should start with the the MOVE part which is implemented in the function `model_proposal`. The particles are moving along with the robot's locomotion, by adding also some Gaussian noise, in order to explore a little more other places. So we are assuming that the particles are not moving perfectly, so there is of course a factor of the error. The corresponding weights of our initial particles are going to be randomly chosen from a Gaussian distribution as we mentioned formerly. But in each step of the iterative application of Bayes Filtering, they need to be updated, which automatically makes us move to the following step of the UPDATE, or `model_correction` as it is referred in the code the related function.

In this part being armed with the knowledge of the weights in the current step, we are going to update our belief about possible candidate states of our robot. This procedure is going to be realized by measuring the distance of the particles to the obstacles using the sensors readings. After returning the corrected value of the range measurement, by inserting some Gaussian noise to the error of the sensors, we compute the new weights.

The last part of the algorithm is the RESAMPLE, which is implemented in the function `model_resample`. As a first step we implement a normalization of the weights, in order to have a probabilistically meaningful context and then we proceed to the choice of a fixed amount of particles randomly. Each entry in the list of the above probabilities represents the probability of each particle surviving to the next step and replicating itself as a candidate state of our robot.

As this recursive procedure continuous indefinitely, the set of the particles are going to converge in a uniform manner to the desired posterior. The first model finally manages to make the particles find MyThymio with a small error, almost converges, as can be seen from the figure below

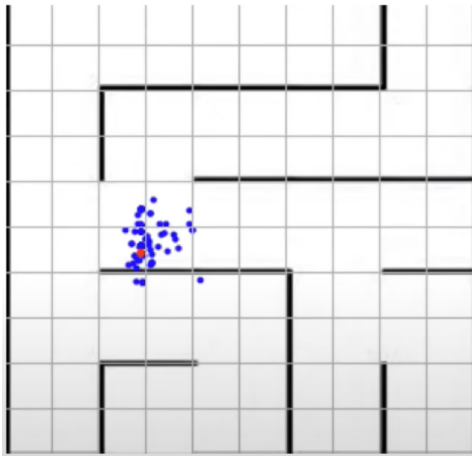

Figure 3. Our robot is the red dot and the blue dots are the particles that survived after many steps.

B. Model with additional knowledge of orientation

Having all that in mind, we concluded with a model that was working properly but the orientation of both the robot and the particles was not taken into consideration. This was something that we should fix, because we are referring to real-life robots that the orientation in the 3-dimensional virtual world is of primal importance. For that reason, we found it really important to extend our initial model to one that could provide information regarding the relative orientation of the particles with respect to the maze and as a result concluding to a more precise overall belief that entails precious information about the posterior attitude of the robot.

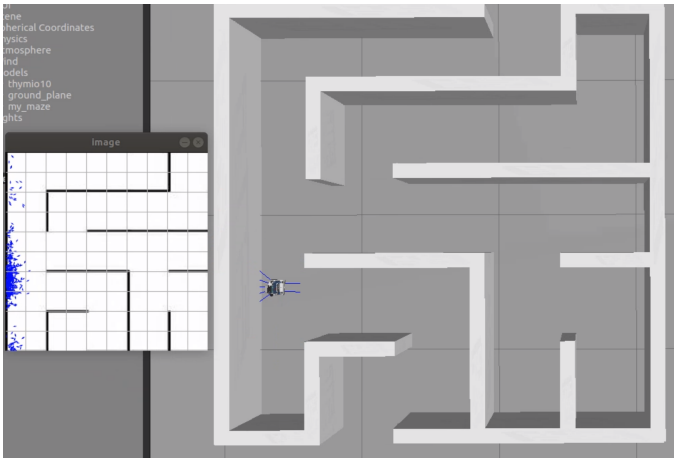

Figure 4. Our robot position and the particles belief about its current state

The incorporation of the orientation to the initial model, led us to some initial difficulties of updating the particles properly, but these obstacles where partially eliminated, while trying different ways of approaching the input of the sensor readings about the particles orientation, and its true attitude.

Lastly, in order to get more clear information from the sensor readings, we proceed in implementing some specific movements of MyThymio, as for example, make it move in a

90°direction and we also made MyThymio omit reading the sensors, unless it is perpendicular to an obstacle.

V. Conclusion and Future Work

This project presented a succinct implementation of the Monte Carlo Localization (MCL), providing means for localizing MyThimio in a maze, with no apriori knowledge about its relative position with respect to the world.
As we could understand by the progression of our model so far, the implementation is a promiscuous baseline to start with, in order to enrich it further with more useful characteristics.

Our project could be extended in such a way, that the particles describe in a more clear and precise way the real position of the robot. As we mentioned previously, in the second version of our model, we experienced some difficulties given the complexity that the orientation posed. However, we managed to conclude to a model that had a solid performance regarding the localization, managing after many iterations to specify a distinct region surrounding the actual robot.

Despite this progress, there exist plenty of opportunities for further extensions that aim to extend our model and also augment the efficiency and the accuracy of the aforementioned. We could extend our research to the sensor readings, in order to find a lucrative way of updating the current belief of the particles given the current orientation. The iterative implementation of Bayesian filtering algorithm with particle-based density has led us to the exploration of many fields of robotics such as SLAM, and we would really like to extend as a future work our model to the concurrent exploration of the world and its localization within it.

VI. Source code

Our code is available at this GitHub repository.

References

[1] Sebastian Thrun, "Particle Filters in Robotics" *In Proceedings of Uncertainty in AI (UAI)*, 2002.

[2] Dieter Fox, Wolfram Burgard, Frank Dellaert, Sebastian Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots", *To appear in Proc. of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, Florida, 1999.

[3] Arnaud Doucet, Nando de Freitas, Neil Gordon, "An Introduction to sequential MOnte Carlo Methods"

[4] Cyrill Stachniss and Luciano Spinello, "Introduction to Monte Carlo Localization", *Practical Course WS12/13* .

[5] Luis Payá,* Lorenzo Fernández, Arturo Gil, and Oscar Reinoso, "Map Building and Monte Carlo Localization Using Global Appearance of Omnidirectional Images", *Departamento de Ingeniería de Sistemas Industriales, Universidad Miguel Hernández, Avda. de la Universidad s/n, 03202, Elche (Alicante), Spain*.

[6] Daniel Lowd, "Monte-Carlo Localization: from Tragedy to Triumph!", *CS 154: Robotics, Spring 2003*.

[7] Liang Chen , Peixin Sun , Guohua Zhang , Jie Niu2 , and Xiaodong Zhang, "Fast Monte Carlo Localization for Mobile Robot.", *School of Information Science Engineering, Shenyang Ligong University, 110159 Shenyang, China, Shenyang Artillery Academy, 110162 Shenyang, China*.

[8] Rainer Kümmerle, Rudolph Triebel, Patrick Pfa, and Wolfram Burgar, "Monte Carlo Localization in Outdoor Terrains Using Multi-Level Surface Maps", *CS 154: Robotics, Spring 2003*.

[9] Anas W. Alhashimi, George Nikolakopoulos, Thomas Gustafsson, "Observation model for Monte Carlo Localization", *CS 154: Robotics, Spring 2003*.

[10] Fox D., "Adapting the sample size in particle filters through kld-sampling", *The international Journal of robotics research*, 22(12), 985–1003 (2003).

[11] Lenser, S. and Veloso, M., "Sensor resetting localization for poorly modelled mobile robots", *In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, 1225–1232. IEEE.

[12] Pfaff, P., Stachniss, C., Plagemann, C., and Burgard, W., "Efficiently learning high-dimensional observation models for monte-carlo localization using gaussian mixtures. ", *In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3539– 3544.

[13] Thrun, S., Fox, D., Burgard, W., and Dellaert, F., "Robust monte carlo localization for mobile robots", *Artificial intelligence*, 128(1), 99–141.

[14] D. Avots, E. Lim, R. Thibaux, and S. Thrun., "A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments.", *In IROS-2002*.

[15] R. E. Kalman., "A new approach to linear filtering and prediction problems.", *Trans. ASME, Journal of Basic Engineering,* 82, 1960.

[16] J. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems.", *Journal of the American Statistical Association,* 93, 1998.

[17] S. Majumder, H. Durrant-Whyte, S. Thrun, and M. de Battista., "An approximate Bayesian method for simultaneous localisation and mapping.", *Submitted for publication, 2002*.

[18] A. Doucet, J.F.G. de Freitas, and N.J. Gordon, editors, "Sequential Monte Carlo Methods In Practice.", Springer, 2001

[19] Kitagawa, G. , "Monte carlo filter and smoother for non-gaussian nonlinear state space models.", *Journal of Computationaland Graphical Statistics*, 5(1).