Improving Monte Carlo Localization Performance Using Strategic Navigation Policies

Henrique José dos S. Ferreira Júnior[1], **Daniel Ratton Figueiredo** [2]

[1] Escola Politécnica (POLI), Departamento de Engenharia Eletrônica (DEL)
Universidade Federal do Rio de Janeiro (UFRJ) – Brazil

[2]Programa de Engenharia de Sistemas e Computação (PESC)
Universidade Federal do Rio de Janeiro (UFRJ) – Brazil

henriquejsfj@poli.ufrj.br, daniel@cos.ufrj.br

Abstract. *An important problem in robotics is to determine and maintain the position of a robot that moves through a known environment with indistinguishable reference points. This problem is made difficult due to the inherent noise in robot movement and sensor readings. Monte Carlo Localization (MCL) is a frequently used technique to solve this problem and its performance intuitively depends on how the robot explores the map. In this paper, we evaluate the performance of MCL under different navigation policies. In particular, we propose a novel navigation policy that aims in reducing the uncertainty in the robot's location by making a greedy movement at every step. We show that this navigation policy can significantly outperform random movements.*

.

1. Introduction

An important problem in robotics is to determine and maintain the position of a robot moving through a previously known environment that has identifiable but indistinguishable reference points. In this problem, the robot has a complete map of the environment as well as the exact location of reference points (landmarks) that are scattered throughout the environment.

However, such reference points are indistinguishable, that is, the robot can not distinguish between one point and another. For example, consider a house with reference points in the living room and in the bedroom. When detecting a reference point, the robot does not know if it is in the living room or bedroom. Another challenge in this problem is the inherent noise in robot movement and identification of reference points. In particular, the robot does not move or detect a reference point accurately (e.g., when trying to move the north, the robot can remain in the same place, or the robot may fail to detect the presence of a reference point).

The robots start in a given place of the map that is unknown to the robot. Thus, the robot has no information about its current position, when it starts. Its purpose then is to navigate the environment and determine (and maintain) its position on the map, based on the reference points encountered during navigation. This is a global localization problem, which differs from the tracking or navigation problem in which the robot starts from a known location and moves to a final destination using a map of the environment.

A technique to solve this global localization problem is Monte Carlo Localization (MCL), which, although proposed in 1999 [Fox et al. 1999, Thruna et al. 2001], has been applied in more and more scenarios. Briefly, MCL uses recursive Bayesian inference, generating random samples of possible locations for the robot (called particles), giving more weight to sites that match the observations (i.e., reference points) and actions of the robot. When moving, the robot must apply the laws of motion by moving the particles and when identifying a reference point it should assign larger weights to the particles located on these reference points. In the next step, the particles are sampled again by following the weights so that the particles are more concentrated in the actual location of the robot. As the robot moves and the algorithm is repeated, the belief associated with some locations become heavier until there is a unique location heavier than all others which is interpreted as the global location of the robot.

In this work, we study the influence of the navigation (i.e., robots' movements) on the accuracy and efficiency of the MCL. How a robot navigates the environment will affect its localization: moving to a location that has no reference points is less effective than moving to a location with a reference point. Intuitively, the random navigation policy will require more movements for the robot to determine its location. Thus, what is the optimal navigation policy? Recall that the robot has a map of the environment and the current belief of its location across all positions. This information can be leveraged to build more effective navigation policies for robot localization.

This work considers environments (maps) in the form of squared bi-dimensional grids (with wrap-around) where each grid location can have a single reference point (all identical). The performance of navigation policies is characterized by two different metrics (to be presented more formally): (i) number of movements until the robot's estimated location coincides with its actual location; (ii) fraction of time where the robot's estimated location is correct. Beyond the random navigation policy, we assess the performance of a globally optimal navigation policy (not realizable in practice) and greedy heuristics that minimize the robot's uncertainty at each step.

In order to assess the performance of different navigation policies, a specific simulator has designed and implemented, that is publicly available. Our findings indicate that the proposed navigation policies can significantly outperform random navigation. Interestingly, the performance of the proposed navigation policies does not exhibit monotonic behavior with the number of reference points, in sharp contrast with random navigation. This indicates that the proposed policies can significantly extract information from the environment to improve localization.

The remainder of this paper is organized as follows. Section 2 briefly presents some related work. Section 3 presents the environment and movement model for the robot as well as the metrics to evaluate the performance of navigation policies. In section 4 we present the navigation problem and different navigation policies. Section 5 presents and discusses the numerical evaluation of such policies. Finally, a brief conclusion and future work are presented in Section 6.

2. Related Work

Monte Carlo Localization (MCL) [Fox et al. 1999, Thruna et al. 2001] is a widely studied technique for the global robot localization with many improvements like Merge-

MCL [Li et al. 2010], and many variations like coping with new sensors such as Bluetooth devices [Hou and Arslan 2017]. However, our work does not focus on improving or adapting MCL, and we assume the classic MCL [Fox et al. 1999] and the classic grid map [Elfes 1987, Milstein 2008]. The focus of this work is to study the influence of the navigation policy on localization performance, a problem known as *active localization* [Fox et al. 1998].

Common solutions to active localization are based on information theoretic metrics, such as the entropy of the robot's location distribution [Fox et al. 1998, Jensfelt and Kristensen 2001]. Such approaches can be computationally intensive in general environments. In a novel and promising approach, the navigation policy for active localization is learned using reinforcement learning [Gottipati et al. 2019]. Note that active localization is related to the more general problem of simultaneous location and mapping (SLAM) [Tovar et al. 2006, Rekleitis et al. 2006].

In comparison, this work shows how to obtain the optimal navigation policy (that minimizes the expected number of movements). Since computing this policy is unfeasible, a simple greedy navigation policy is proposed and evaluated.

Finally, the metrics to assess the performance of the navigation policy with respect to localization were recently introduced in a related work [dos S. Ferreira Júnior and Figueiredo 2018], where the goal was to assess the influence of the map on localization (and not the navigation policy). Due to the stochastic environment (in both moving and sensing), these metrics are assessed using Monte Carlo simulations, a common approach in this context [Baudry et al. 2018].

3. Models and Evaluation Metrics

We consider an environment in the form of a squared two-dimensional grid with wrap-around (i.e., torus) of size $n{*}n$. That is, each location in the environment has exactly four neighboring locations and moving exactly $n + 1$ steps in the same direction returns to the starting point. Each location in the environment corresponds to a possible location for the robot, which in this case has n^2 different locations. Exactly p reference points (landmarks) are added to the environment, all identical (this means that the robot knows it is in a location that has a reference point, but not what is this location). Unless otherwise stated, the location for the reference points is chosen uniformly at random.

Figure 1 illustrates an environment where $n = 10$ and $p = 50$. The presence of a reference point is denoted by a black square and the absence by a white square. However, the performance of MCL in this scenario is symmetric with respect to $p = 50$, such that $p = 49$ has same expected performance as $p = 51$, and the same happens to $p = 1$ with $p = 99$ (see discussion in [dos S. Ferreira Júnior and Figueiredo 2018]). Note that if $p > 50$ then the white squares (locations that do not have a reference point) can be interpreted as the reference points. Thus, the number of reference points can vary between $p = 1$ and $p = 50$, as it represents the locations that give more information about the robot's position.

Initially, in time $t = 0$, the robot is positioned randomly and uniformly in some state of the environment and it does not have any information about its position. We will consider a discrete-time model, where at each step the robot performs a movement and a reading to identify the presence of a reference point. The move moves the robot to an adjacent location: up, down, left, or right. Note that the robot is not intended to reach a predetermined target, but rather to locate (and remain localized) in the environment. Figure 1 illustrates the position of the robot in an environment with reference points (black squares) and the possible states for which it can be moved (blue squares).

To model the noise in the robot motion, we will consider that the robot has a 10% chance of overshooting, i.e. moving an additional position in the same chosen direction, and a 10% chance of undershooting, i.e., staying in the same location (not moving). Thus, the attempt to move to the chosen adjacent location occurs satisfactorily with a chance of 80%. It is important to note that the robot is not able to determine if there has been an overshoot or undershoot when moving.

To model the noise in the identification of reference points, a reading tells the robot whether the current location is black (i.e., has a reference point) or white (i.e., does not have a reference point) with a 10% error in each case. That is, the robot correctly detects the presence or absence of a reference point in its location with a 90% chance.

The variation of MCL that will be used in this work maintains a probability of the robot's position for each location of the map. In particular, $P_x(t)$ is the probability that the robot is in location x of the environment at time t. Note that $P_x(0) = 1/n^2$, because the robot has no information about its initial location. Note that $P_x(t)$ is updated at every time step according to the chosen movement and the observation of a reference point.

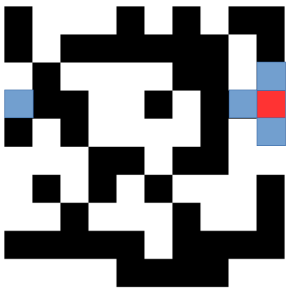

Figure 1. Example of a grid with $p = 50$ with robot in the red square.

3.1. Evaluation Metrics

We will use two metrics of [dos S. Ferreira Júnior and Figueiredo 2018] two different metrics will be used to quantify the performance of navigation policies. The first metric captures the fraction of movements (steps) that the MCL estimates the correct position of the robot. The second metric captures the number of movements (steps) required for the robot to find itself correctly for the first time (from an unknown initial state). In the following, we formalize these two metrics.

Let $R(t)$ be the actual location of the robot at time t. Let $H(t) = \{y \mid y = \arg\max_x P_x(t)\}$, where $P_x(t)$ is the probability according to the MCL of the robot being in state x at time t. That is, $H(t)$ is the set of states with the highest probability in time t. Lets define $A(t) = 1$ if $|H(t)| = 1$ and $H(t) = R(t)$, and 0 otherwise. That is, $A(t)$ indicates that the MCL correctly estimates the position of the robot at time t. Let r be the number of steps taken by the robot during an excursion in given environment G. Lets define: $E_G(r) = \sum_{t=1}^{r} A(t)/r$.

The $E_G(r)$ metric captures the intuition that MCL performance is directly proportional to the fraction of time steps in which the robot is correctly located in the environment. $E_G(r)$ is the hit rate of the MCL after r steps, which converges when $r \to \infty$, considering that the environment is finite. However, note that $E_G(r)$ (for fixed r) is a random variable since MCL is random. For a fixed r (e.g. $r = 100$), we can estimate the expected value of $E_G(r)$ by making the sample average of several independent rounds.

Another important metric is the time (measured in number of steps) to the first correct localization. Let F_G be the first instant of time t where $A(t) = 1$ for a given environment G. That is: $F_G = \min t \mid A(t) = 1$.

Notice that F_G is a random variable since it depends on the execution of the algorithm. Again, we report the sample mean of F_G by running the algorithm for many independent rounds. Note that a low value in this metric is important because it indicates that the robot can quickly localize itself in the environment without having any previous information about its position.

3.2. Simulator

A simulator of the model presented to move the robot and detect the presence of reference points was designed and implemented for this work. In addition, the MCL algorithm was also implemented to determine the location of the robot as a function of the observations. The choice of make our own simulator instead of using Gazebo on ROS, for example, comes from our model (referred as map) that is naturally discrete and without border. There no tool to simulate that and since our code was made just for that it is also efficient.

The simulator also implements different navigation strategies. It also has several parameters, such as the size of the environment (bi-dimensional grid), and the quantity and position of the reference points. The proposed performance metrics were collected by the simulator, which performs several independent rounds to estimate a sample average. The simulator is publicly available `https://gitlab.com/henriquejsfj/Monte_Carlo_Localization.git`.

4. Navigation Policies

A navigation policy determines what movement the robot is to make at each time step. Navigation policy is constructed to achieve a specific goal, usually to minimize a given cost function. Moreover, navigation policies can be static or dynamic in the sense that they can leverage and adapt to the information acquired by the robot.

In this work, the goal of the navigation policy is to effectively determine the location of a robot that is running MCL. This navigation policy is dynamic and will leverage the current belief of the robot to determine its next move.

4.1. Optimal policy

Consider a robot that wants to minimize the number of steps necessary to finds its location. How should the robot move in order to attain this minimum? The optimal policy embodies the movements to attain this minimum and is obtained as follows.

Consider a model without any movement and reading noise and a specific map of the environment. We will construct a tree where each node corresponds to a probability distribution of the robot's location as determined by MCL, namely $P_x(t)$ where t will correspond to the level in the tree. Since the robot has no information, the root of the tree is simply the uniform distribution across all locations, thus, $P_x(0) = 1/n^2$ for all x in the $n{*}n$ grid. Moreover, we assume the initial position of the robot is given by a uniform distribution across the locations.

Each node in the tree has exactly eight children which correspond to a movement followed by a sensor reading. Since there are four possible movements and two possible readings, this gives eight different children. For each child, the probability distribution of $P_x(t)$ is computed accordingly (as determined by MCL). Since the actual location of the robot is also unknown, this calculation requires conditioning on every possible location. This process starts at the root and continues on every node until it reaches a leaf. A node in the tree is a leaf when the probability distribution gives probability 1 to some location x. This means that the robot has correctly determined its location.

Note that the tree will have leaves at different levels. The leaf at the smallest level determines the minimum number of movements for the robot to find its location given that it was placed at a random location when it started. The required movements are given by the path from the root to this leaf. This is the optimal navigation policy. Figure 2 shows an example of the tree (where all leaves have been aggregated into a single node).

However, note this optimal policy was obtained in retrospect, assuming all possible movements and readings at each step. Thus, it is not realizable since in practice the robot must make a decision when navigating. Also, this optimal policy requires exploring a tree that grows exponentially with the number of steps. Moreover, computing the probability distribution of a node in the tree is a computationally expensive procedure, with complexity $O(n^4)$.

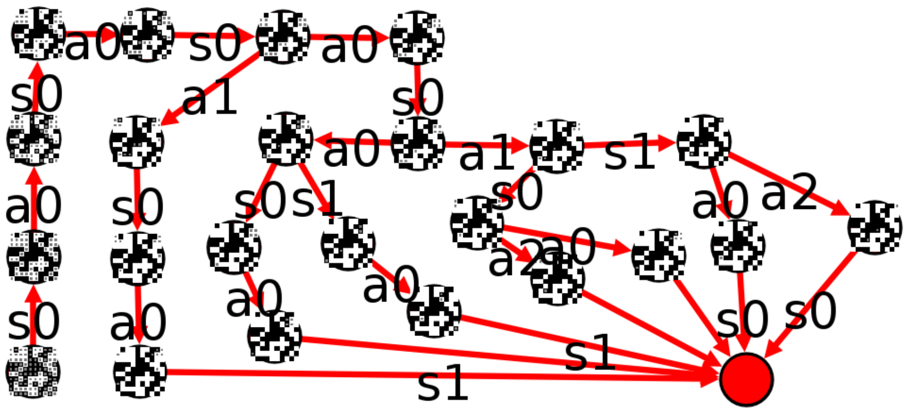

Figure 2. Example of the tree generated to construct the optimal policy. The nodes show the map where each location x with positive P_x has a gray small square. The edges with s0 and s1 are a sensor reading white and black respectively. The edges with a0, a1, a2 and a3 are action to west, north, east and south respectively. The red node denotes all nodes where the robot has localized itself (although the distribution is different).

4.2. Optimal policy for a given location

In the previous section, the initial location of the robot was assumed to be random. However, the robot does start in a given location. This information can be used to construct an optimal policy for that given location.

The procedure to obtain this optimal policy is very similar to the previous tree construction. The robot now starts in a given location, unknown to the robot, and thus, $P_x(0) = 1/n^2$ for all x in the $n{*}n$ grid. However, when each direction is explored it finds exactly what the map determines (presence/absence of a landmark). Thus, the number of children of a node in the tree is four, corresponding to each possible movement, as the reading will be determined by the location. For each child, $P_x(t)$ is computed as determined by MCL. In this scenario, no conditioning is required as the robot is in a given location (not yet known to the robot). Nodes in the trees are explored and stop on leaf nodes. Again, a leaf is a node that assigns probability 1 to a given location.

Note that the tree will have leaves at different levels. The leaf at the smallest level determines the minimum number of movements for the robot to find its location given it started from a given location. The required movements are given by the path from the root to this leaf. This is the optimal navigation policy for the given initial location.

This optimal policy is also not realizable because in practice the robot does not know its initial location. However, in retrospect, this is the best policy for navigating the robot given an initial location. Obtaining this policy also requires exploring a tree that grows exponentially and computing the probability distribution for each node has complexity $O(n^2)$.

In any case, this optimal policy provides a baseline for the localization performance as no other policy can be better. Moreover, the policy itself (what movements it choose) can provide insights into designing an efficient navigation policy. In particular, the optimal policy tends to move the robot in a way that seems to minimize the uncertainty concerning its location. This observation will be used to design the greedy navigation policies that we next describe.

4.3. Cost function for greedy navigation

The design of a good navigation policy requires determining the expected reading of a movement. Thus, given the current belief of the robot's location and the map, the robot can compute what is the probability of observing a reference point when moving in each possible direction. This probability can then be applied to a cost function that determines the expected cost (negative of the benefit) for the robot if taking this movement. Finally, the robot simply chooses the movement that minimizes the cost function. Note that the computation of the probability of observing a reference point when moving in a given direction can take into consideration both the movement noise and the sensor noise.

An adequate cost function should measure the uncertainty that remains after the movement. Thus, choosing the minimum cost movement leads to greedily choosing a movement that reduces uncertainty. An effective alternative is simply the number of possible locations in $P_x(t)$, namely the number of locations for which $P_x(t) > 0$. Intuitively, this cost function reduces the number of possible locations faster than other cost functions.

In what follows we characterize the behavior of the proposed cost function. Consider a noiseless scenario and let L_t denote the number of possible locations at time t, in particular, $L_t = |\{x|P_x(t) > 0)\}|$. Since we assume a model with no noise, we have that each possible location has probability $\frac{1}{L_t}$, in particular, $P_x(t) = \frac{1}{L_t}, \forall x \in \{x|P_x(t) > 0)\}$.

Let p_B and p_W denote the probability of reading black (a reference point) or white (no reference point) after taking a given movement. Let S be a random variable that denotes the sensor reading after the movement, such that $p_b = P(S = b)$ and $p_w = P(S = w)$. Note that $p_b + p_w = 1$. Using this information, the expected value for L_{t+1} for a given movement can be computed as follows:

$$E(L_{t+1}) = p_b * E(L_{t+1}|S = b) + p_w * E(L_{t+1}|S = w)$$

Since $p_b + p_w = 1$, we have:

$$E(L_{t+1}) = (E(L_{t+1}|S = b) - E(L_{t+1}|S = w)) * p_B + E(L_{t+1}|S = w)$$

As a movement does not change the number of possible locations, it follows that $E(L_{t+1}|S = b) + E(L_{t+1}|S = w) = L_t$, then:

$$E(L_{t+1}) = (2 * E(L_{t+1}|S = b) - L_t) * p_b + L_t - E(L_{t+1}|S = b)$$

But $E(L_{t+1}|S = b)$ is the number of possible location after sensing white, i.e. number of possible locations over white space. So p_b can be rewritten as $\frac{E(L_{t+1}|S=b)}{L_t}$, thus:

$$E(L_{t+1}) = \frac{2 * E(L_{t+1}|S = b)^2}{L_t} + L_t - 2 * E(L_{t+1}|S = b)$$

Given this expression for the cost function which captures the expected number of possible locations, we can minimize it as follows (computing its derivative):

$$\frac{d(E(L_{t+1}))}{dE(L_{t+1}|S = b)} = 0 \Longrightarrow \frac{4 * E(L_{t+1}|S = b)}{L_t} = 2 \Longrightarrow E(L_{t+1}|S = b) = \frac{L_t}{2}$$

This result indicates that the expected number of possible locations for L_{t+1} is minimized when a movement divides the prior L_t equally in black and white. Note that this result does not depend on the number of reference points on the map. Interestingly, this same result of $\frac{L_t}{2}$ can be obtained using entropy as a reward function and maximizing (instead of minimizing).

A limitation of this cost function is that the notion of possible locations is not well defined under a model with noise, as $P_x(t)$ is strictly positive for all locations. Moreover, it requires the computation of an expected value that has cost $O(n^2)$.

Another cost function is simply the absolute value of the difference between p_b and p_w, namely $|p_b - p_w|$. Note that this cost function is much simpler to compute and can be computed under noise. This cost function is closely related to the previous one, although much more general and simpler to compute.

Recall that p_b and p_w are computed considering movement noise and sensor noise, such that it reflects what the robot will read, in expectation. Using these quantities directly provides a robust while very simple cost function.

5. Experimental Results

To assess the different navigation policies, three different experiments are considered. In each scenario, random movement is used as a navigation policy, the baseline is given by the optimal policy, and the proposed navigation policy minimizes $|p_b - p_w|$, denoted here as *better split*.

5.1. Number of landmarks and time to localization

The objective of this experiment is to analyze the first time that the robot correctly determines its location (metric F_G) for the different navigation policies. This comes from the idea, that since the robot is trying to localize itself navigation where the robot self-localize faster is better. So, with this experiment, we would like to conclude which of the navigation policies is better.

This experiment consists of generating maps of $n = 10$ and positioning the robot in an initial location and then start the simulation, considering each navigation policy. At each time step the robot move and read the sensor, the robot has 300-time steps to try to localize itself (if this not happens we consider the time to first hit is 300). We analyze this metric for a different number of reference points, p, ranging from 1 to 49 with an increment of 2. An average of the first time to hit (F_G) is computed using 200 different maps for every number p. Note that the reference points are placed uniformly at random across the locations.

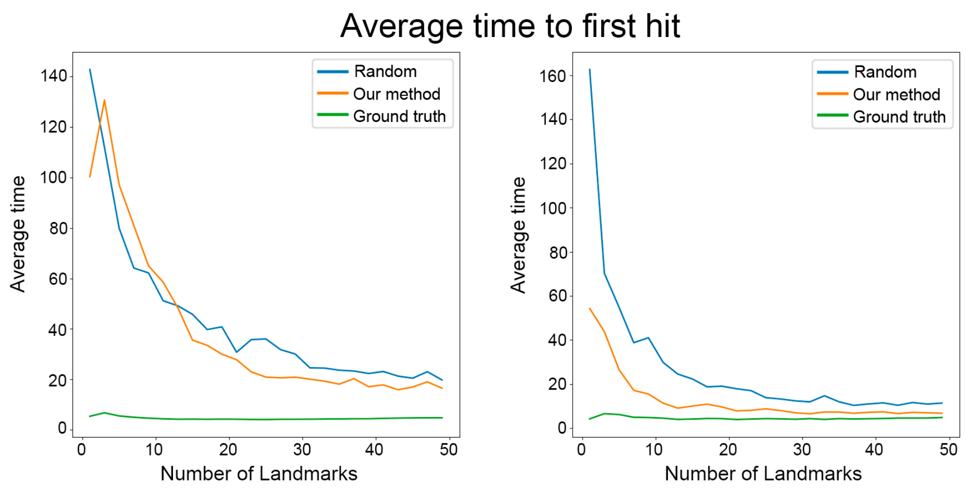

Figure 3. Performance in a scenario with noise in movement and sensor (left plot); performance in a scenario without noise (right plot).

As Figure3 shows, the proposed method is better than random navigation. In the noiseless scenario, the proposed method is much better than random, but the noise turns things more difficult making random slightly superior for a small range of p. To understand the influence of noise on performance the same experiment was made with noise just on readings and on movements, separately, and results are shown in Figure 4.

The plot in Figure 4 indicates that the noise makes the performance of our proposed navigation method be closer to the random one. In particular, that anomaly of random being better than our for a short-range, between 3 to 11 reference points, occurs only in a scenario with movement noise.

5.2. Symmetry of landmarks and time to localization

This experiment characterizes the performance under different levels of symmetry concerning the position of the landmarks while keeping their number constant. We start from a complete symmetric positioning of these landmarks and break this symmetry to analyze

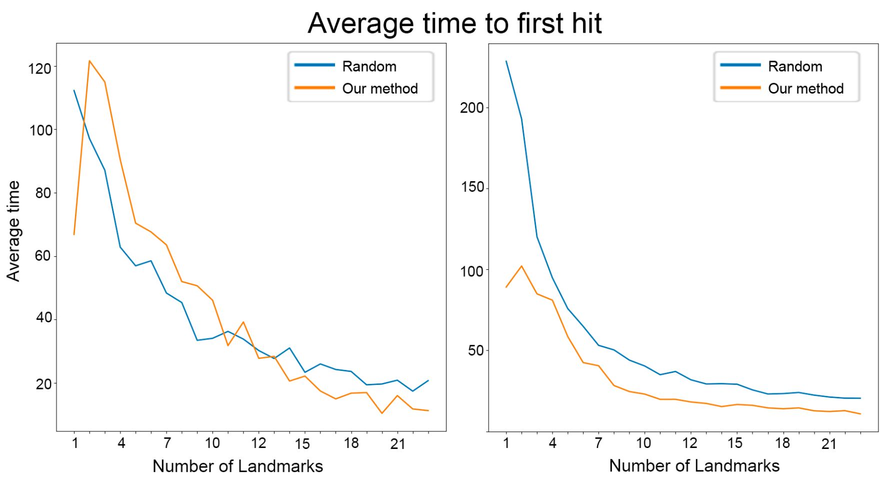

Figure 4. Performance with noise just on movement (left plot) or just on sensor reading (right plot).

the performance. The idea is to analyze the navigation policies in maps that are more difficult for the robot to find its location.

To generate different maps with different levels of symmetry, we generate a chessboard grid completely symmetric, and then change black points with the white points randomly, to break the symmetry. The number of exchanges goes from 1 to 32 on an exponential scale. An average across 30 maps for every number of exchanges was made.

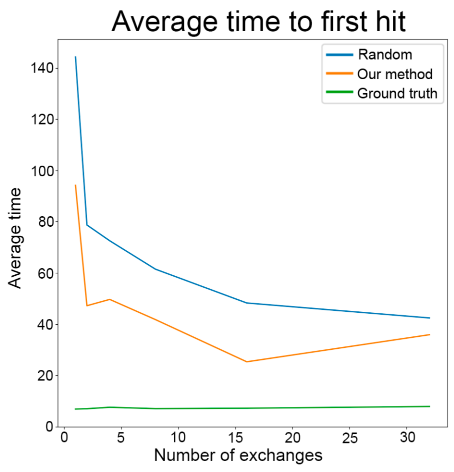

Figure 5. The graphic shows the experiment, the horizontal axis represents the number of asymmetry and vertical ones the first time to hit

The graphic on Figure 5 shows us that the proposed navigation policy is always better than the random, especially when the map has a more symmetric landmark. This means that our method explores better the existent asymmetry on the map. Note that the anomaly observed in the previous experiment does not occur.

5.3. Number of landmarks and average hit rate

The objective of this experiment is to analyze the hit rate of $E_G(2000)$ of the different navigation methods. We are interested in understanding how the robot chooses its movements to maintain its position. Considering a map with just one landmark, $p = 1$, we expect that once the robot is localized the proposed navigation policy maintains the robot going out and turning back to the landmark making the hit rate very high.

This experiment consists of generating maps of $n = 10$ with p landmarks, ranging from 1 to 99 with an increment of 2. Recall that 99 is statistically equivalent to 1 landmark. The average of 10 maps is considered for every value of p.

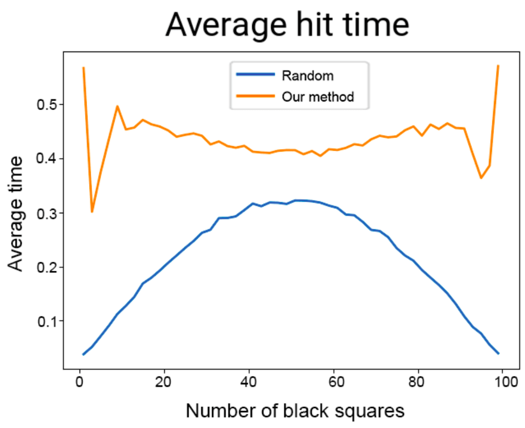

Figure 6. The vertical axis is the hit rate and the horizontal ones is the number of landmarks.

Figure 6 shows the performance of the two navigation policies. The proposed policy is superior to random navigation. Moreover, note the concavity of random policy is different from the proposed, and in particular has its peak at 50. Especially, not considering the case with just one landmark, the peak for our method occurs on 11 landmarks, that result might come from the asymmetry that our method explores cleverly. On the range from 3 to 7 landmarks, we see a low performance. This might happen because with these few landmarks the landmark positions are likely to be more symmetric.

The increase of landmarks breaks the symmetries but we find an optimal point for 11 landmarks where the robot can explore these areas that bring more information gain. Getting closer to 50 landmarks, we observe that random navigation is best (a result also showed in [dos S. Ferreira Júnior and Figueiredo 2018]) but the proposed method continues to be superior.

6. Conclusion

This work considers the investigation of navigation policies to improve the performance of Monte Carlo Localization. A procedure to compute the optimal policy has been described. While this procedure is not practical it served as a baseline to assess and provide insights to building heuristics for effective navigation policies.

We proposed a navigation policy that improves the localization performance, including a heuristic that is robust and fast to compute. The performance is assessed in different scenarios and compared to the baseline and random navigation. These experiments revealed interesting findings such as the superiority of the proposed navigation policy in many scenarios, as well as its non-monotonic behavior for the average hit rate.

Another interesting result is the relatively poor performance when the number of landmarks ranges from 3 to 7. In this regime, the movement noise and symmetry of landmarks undermine the proposed method. The investigation of navigation policies that can also be effective in this regime is left as future work.

References

Baudry, G., Macharis, C., and Vallée, T. (2018). Range-based multi-actor multi-criteria analysis: A combined method of multi-actor multi-criteria analysis and monte carlo simulation to support participatory decision making under uncertainty. *European Journal of Operational Research*, 264(1):257 – 269.

dos S. Ferreira Júnior, H. J. and Figueiredo, D. R. (2018). Influence of location and number of landmarks on the monte carlo localization problem. In *XV Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*.

Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Nat. Conf. on Artificial Intelligence (AAAI)*, pages 343–349.

Fox, D., Burgard, W., and Thrun, S. (1998). Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25(3-4):195–207.

Gottipati, S. K., Seo, K., Bhatt, D., Mai, V., Murthy, K., and Paull, L. (2019). Deep active localization. *IEEE Robotics and Automation Letters*, 4(4):4394–4401.

Hou, X. and Arslan, T. (2017). Monte carlo localization algorithm for indoor positioning using bluetooth low energy devices. In *Int. Conf. on Localization and GNSS (ICL-GNSS)*, pages 1–6.

Jensfelt, P. and Kristensen, S. (2001). Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760.

Li, T., Sun, S., and Duan, J. (2010). Monte carlo localization for mobile robot using adaptive particle merging and splitting technique. In *IEEE Int. Conf. on Information and Automation*, pages 1913–1918.

Milstein, A. (2008). Occupancy grid maps for localization and mapping. In *Motion Planning*. InTech.

Rekleitis, I., Meger, D., and Dudek, G. (2006). Simultaneous planning, localization, and mapping in a camera sensor network. *Robotics and Autonomous Systems*.

Thruna, S., Foxb, D., Burgard, W., and Dellaert, F. (2001). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141.

Tovar, B., Muñoz-Gómez, L., Murrieta-Cid, R., Alencastre-Miranda, M., Monroy, R., and Hutchinson, S. (2006). Planning exploration strategies for simultaneous localization and mapping. *Robotics and Autonomous Systems*, 54(4):314–331.