University of Montpellier - LIRMM

Matlab report

Localization by particle filter Off-line and Online localization performance evaluation

Supervisor : - Mr. Didier CRESTANI - Mr. Philippe LAMBERT

Authors : - BENALI Abderahmane - BOUZIT Zakaria

Contents

List of Figures

1 Introduction

Mobile robots have remarkable achievements in industrial manufacturing, medical science, social science, agriculture, education, games, space research, etc. Navigation is a challenging problem for autonomous mobile robots. To perform successful navigation, the robot passes through different phases such as perception, localization, cognition, and motion control. In the perception phase, the robot extracts meaningful data by interpreting its sensors. In the localization phase, the robot estimates its current location in the employed environment using information from external sensors. In the cognition phase, the robot plans the necessary steps to reach the target. The motion control phase lets the robot achieve its desired trajectory by modifying its motor outputs. In the past decade, localization has received the greatest research attention (Siegwart et al., 2011). The rest of this paper focuses on the localization problem and different strategies developed for positioning autonomous robots.

2 Localization Problem

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's location is an essential precursor to making decisions about future actions.

Based on the information of the initial position, we classify the localization problem into position tracking, global positioning/localization, and kidnapped robot problem. In position tracking, the robot's initial position is known and the objective is to track the robot at each instance of time during its navigation in the environment. During navigation, the localization algorithm utilizes the robot's previous position to update its current location. This is possible by continually monitoring the route of the robot. Position tracking utilizes odometry and sensor data. However, in case of large uncertainty, the robot might not be localized. Hence, in position tracking, the position uncertainty of the robot is required to be small. Moreover, in position tracking, the robot's initial belief (best guess about the initial state) is a normal distribution.

Relocation deals in mobile robot tracking without any information on its

initial pose. This gives rise to another class of localization problems: global localization. In this category, the robot does not have any knowledge about its initial position. This signifies that the robot is able to locate itself globally within the environment (Negenborn, 2003). In some situations, the robot is tracked at an arbitrary place sometimes during pose tracking or abducted (kidnapped) to an unidentified place. This issue arises in kidnapped robot problem where, the robot knows that it is being kidnapped. Hence, Kidnap recovery is necessary for any autonomous robots. In most of the situations, the current sensor data is utilized in estimating its pose. Basically, the best match between known data and sensor data solves the issue of relocation. An autonomous robot should be able to handle pose monitoring and relocation simultaneously. It should be able to recognize that; it is being kidnapped and it should recover its pose by applying the relocation method. [1]

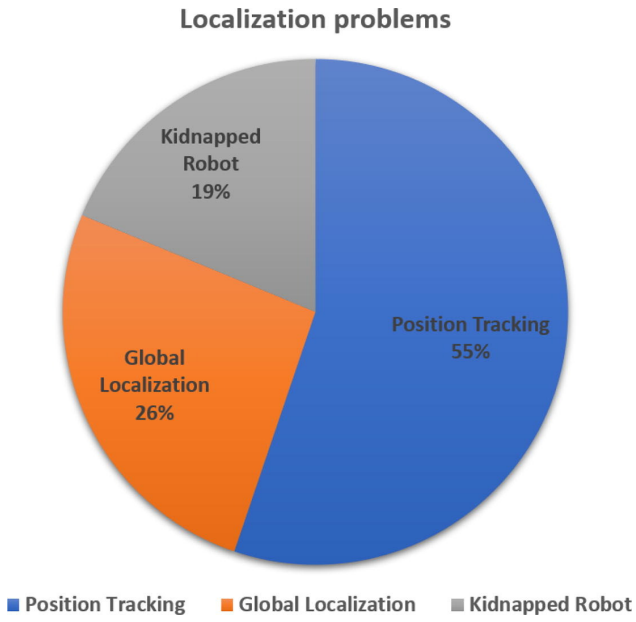

Figure 1: Localization problem partition [1]

One of the key developments in robotics has been the adoption of probabilistic techniques which integrate imperfect models and imperfect sensors through probabilistic laws. In this work, we'll focus on a localization probabilistic approach called Particle Filter. [2]

The problem developed consists rather in exploiting an already existing map of the environment and determining the position and orientation of the robot relative to this map using Particle Filter, then to qualify in simulation the performances of this localization method global in terms of uncertainties and adaptability to our environment.

3 Localization using particle filter

Given a map of the environment, Particle filter estimates the position and orientation of a robot as it moves and senses the environment. For that, it uses a swarm of points called 'particles' to represent the distribution of likely states, with each particle representing a possible state, i.e., a hypothesis of where the robot is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive Bayesian estimation, i.e., how well the actual sensed data correlate with the predicted state. Ultimately, the particles should converge towards the actual position of the robot. [3] [4]

4 Particle filter algorithm

The PF-Localisation algorithm proceeds as follows [5]:

1. **Initialization:** we initialize N particles in our environment, since we are estimating the position of a mobile robot, each particle is a 3×1 state vector with x, y coordinate for position and θ for orientation.

2. **Control:** We apply the same control law applied on the robot for all the sets of particles, this will result in having a lot of different versions of the state vector that we can compare to the robot state vector.

3. **Measurements and likelihood:** For each particle, predict the observation using a model of the sensor, and compare this value to the actual measured value of the robot. This is done using a likelihood function to evaluate the likelihood of the observation given the state presented by each particle, this will be a scalar that is associated with each particle, it's referred to as "weight" or "importance".

4. **Estimation:** After calculating the weights of each particle, we calculate the estimated position of our robot based on all the particles.

5. **Selection:** Select the particles that best explain the observation, there are many ways of doing this, the obvious one is simply to sort and select the top candidates (the particles with the biggest weights) but this would reduce the number of particles. One solution to this would be copying the best candidates until having the N particles again, the consequence of this is reducing the diversity and spread of the particle set. Instead, we randomly sample from the samples biased by the importances values. for this two methods are presented with different approaches. In both, there is a chance to pick a particle that didn't necessarily explain the observation well, but this may turn out to be a good choice because it may do a better job for the next observation.

6. **redistribution:** if the weights are too concentrated, we redistribute the particles on all the environment, this step has two advantages, first, the particles may be concentrated around the wrong position and the redistribution will correct this. Second, this step is important in the case of a kidnapping phenomenon.

7. go back to step 2.

the following scheme resume all the steps of the particles filter algorithm:

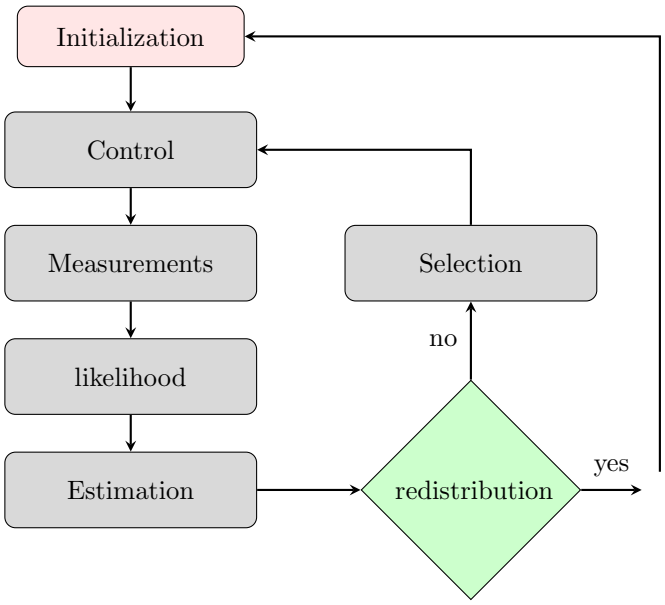

Figure 2: Particle filter algorithm

5 Paticle Filter implementation in Matlab

5.1 Particle Filter functions

5.1.1 Intialization

We generate N number of particles in our environment, for that, we use the function **"particle-generation.m"** that randomly generates N particles in a given rectangle. In our case, we divide our map into two rectangles, and we generate $N/2$ particles in each of them which the orientation going from $-\pi$ to π, we suppose that a particle can't go closer than $0.5m$ to the wall, the function keeps generating particles and checking if there are inside our environment using **"isinBoxmap.m"** function until reaching the desired number of particles.

5.1.2 Measurements

for this we have two types of sensors [6]:

laser sensor

we have two laser sensors that cover 240° each, the first one is oriented to the front of the robot and the second to the back of the robot, each of them sends 511 laser light in different directions and get the distance from obstacles in that direction. This function is implemented in the Matlab functions **"Mesure-act.m"** and **"USPATCH-act.m"** with the number of points used as an input, since using all of the points is time-consuming.

the following figures represent the laser lights sent by the sensors:

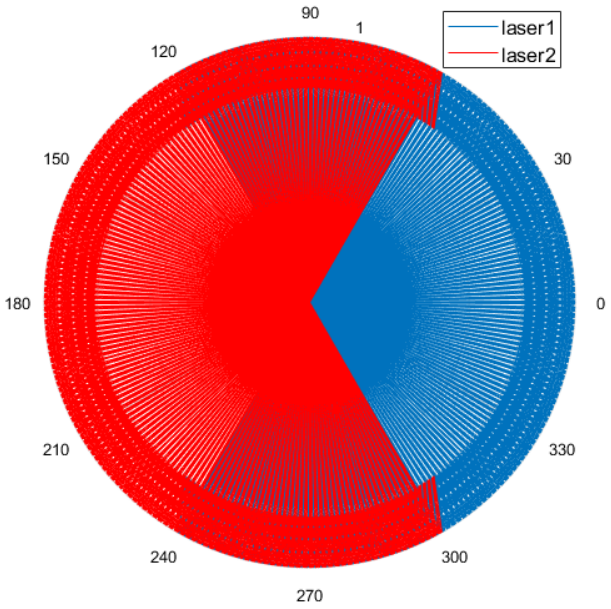

Figure 3: laser lights with all the points

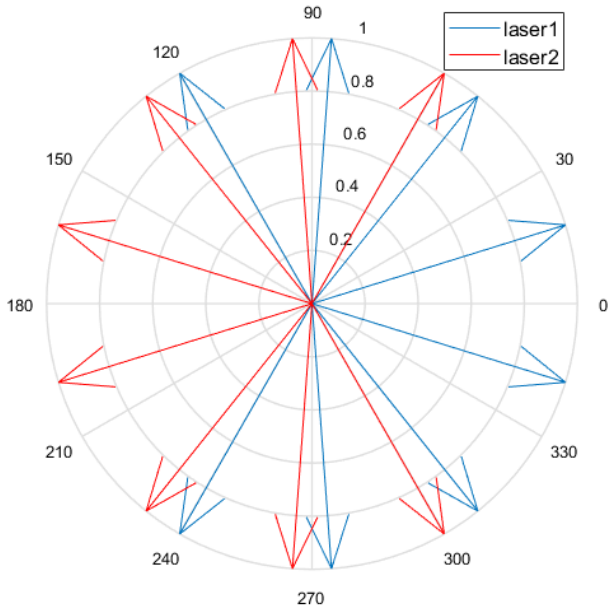

Figure 4: laser lights with 16 points

US sensor

16 ultrasounds sensor is used, with their repartition illustrated in figure 5. The US sensor calculates the distance by measuring the time of flight of an acoustic wave, the limits of the US are from $0.1m$ to $4m$. This function is implemented in the Matlab functions **"Mesure-act.m"** and **"USPATCH-act.m"**

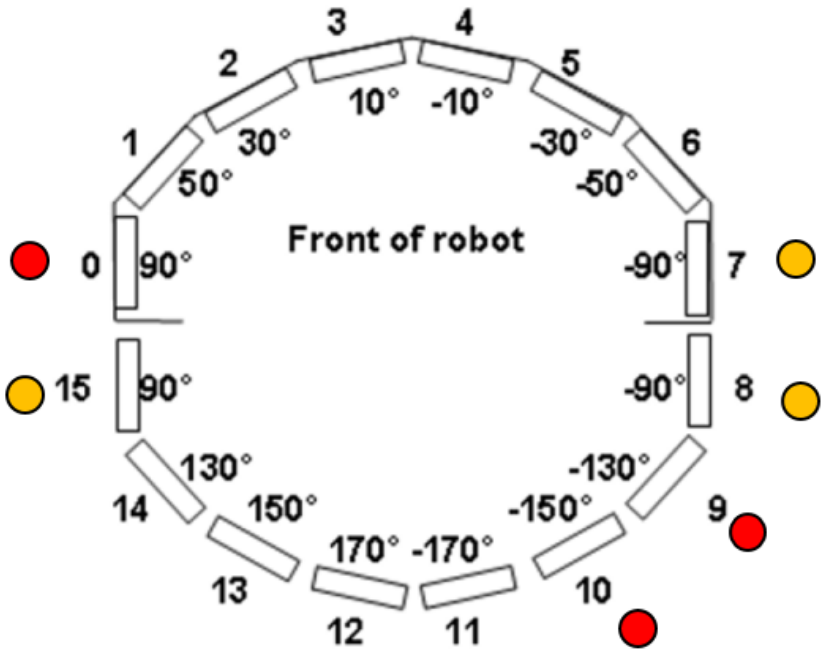

Figure 5: US sensors in the robot

5.1.3 Likelihood

For this we suppose that a measurement z that we are given is in some5 way related to the state we wish to estimate, we also supposed that the measurement of the robot is not precise and is noise corrupted. the distribution of a particle is the conditional probability of the measurement z given a robot measurement of x. a gaussian in our case which gives us the weight of the particle[5][7]. This function is implemented in the Matlab function **"likelihood.m"**.

$$\delta = rob_{measurement} - particle^{i}_{measurement} \tag{1}$$

$$\sigma = \sqrt{var(\delta)} \tag{2}$$

$$E = (\delta - mean(\delta))/sigma \tag{3}$$

$$w^{i} = \frac{1}{\sqrt{2.\pi}}.e^{-0.5.E'.E} \tag{4}$$

5.1.4 estimation

To estimate the position we take the sum of all particles multiplied by their normalized weights[8]. This function is implemented in the Matlab particle filter core function **"ParticleFilter.m"**.

$$x_{estimated} = \sum_{i=1}^{N} \overline{w}(i) \ . \ x_{particle}(i) \tag{5}$$

$$y_{estimated} = \sum_{i=1}^{N} \overline{w}(i) \ . \ y_{particle}(i) \tag{6}$$

$$\theta_{estimated} = \sum_{i=1}^{N} \overline{w}(i) \ . \ \theta_{particle}(i) \tag{7}$$

with $\overline{w}(i)$ is the normalized weight ($\overline{w}(i) = \frac{w(i)}{\sum w}$)

5.1.5 Selection

in the part two methods are presented:

Fitness proportionate selection(roulette wheel)

Roulette selection is a stochastic selection method, where the probability for the selection of an individual is proportional to its weight. The method is inspired by real-world roulettes but in this case, the roulette is weighted. The selection is based on the binary search algorithm.
Each time we randomly select a particle for the next generation until we select N particles for the next generation[7]. The particles with more weights have more chance to appear in the next generation.

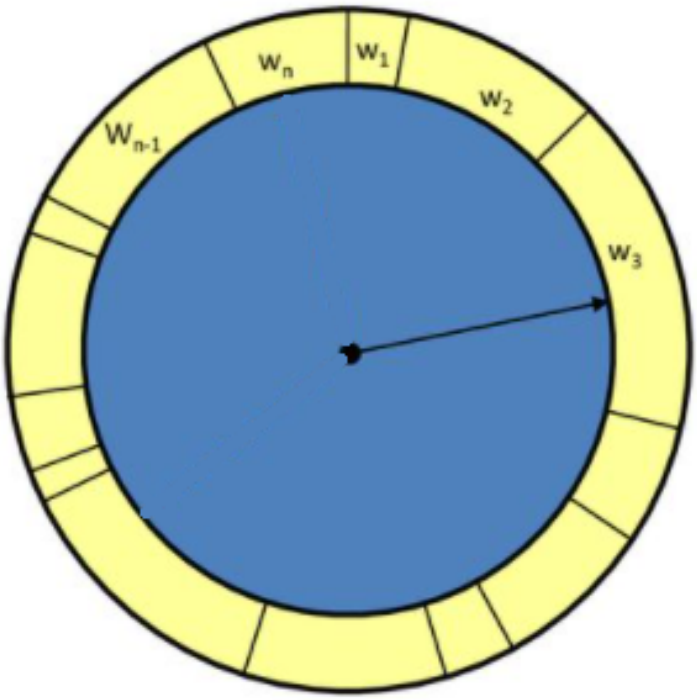

Figure 6: roulette wheel selection

Stochastic universal sampling

this method is a development of fitness proportionate selection (FPS) which exhibits no bias and minimal spread (minimal variance). the previous method choose several solutions from the particles by repeated random sampling. On the other hand, this method uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals. This can give the particles with lower fitness more chance to appear in the next generation.[9] [10]

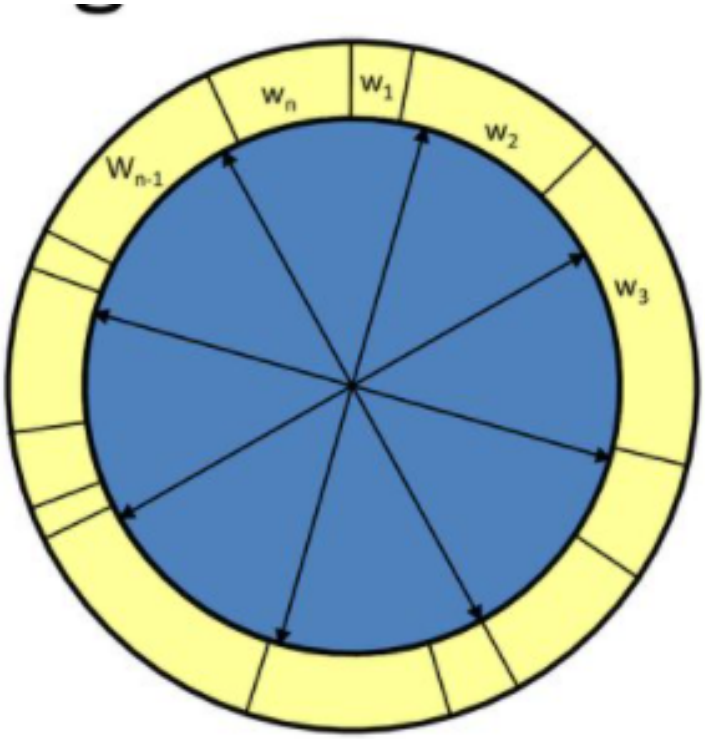

Figure 7: Stochastic universal selection

These functions are implemented in the Matlab function **"selection.m"**.

5.1.6 Redistribution

when the particles are too focused around a single point we redistribute them to all the environment, this is helpful for two reasons, the particle filter may be converging for a false position (if this position is similar to the real one) and since the selection of particles for the next generation is done from the current distribution, the false position will remain in the next generations, the second problem can be the kidnapping problem, if we lose the robot position and we have all particles focusing on a single point, we can't find the new position of the robot, that's why we introduce two conditions for particles redistribution in the matlab function **"resambling.m"**.:

- the first is related to the wheighted standard deviation: we calculated for all particles for x, y, and θ and if it's smaller than a certain ceil we redistribute the particles. [7] [8]

- The second condition is related to the difference between the distance of the robot (calculated using odometry) and the distance of the estimated robot (we calculated in simulation). It's proven by simulation, that if the particle

filter converges to a false position, we'll have a difference in the distance between the robot and the estimation. [7][11]

5.2 Other functions

5.2.1 Trajectory generation

This function takes three arguments; the start point, the endpoint, and the number of intermediate points NPP (assuming that the trajectory between two points is a line), the function takes the distance between the start and end point and divide it into NPP rectangle, then uses the particle generation function to generate a random point in this rectangle assuming that we can't get too close to the wall (we keep a distance of $0.75m$).

- in the next example: we generate a random trajectory between the starting point $(28, 50, \pi)$ and the endpoint $(22, 0.5, \pi)$ with 10 points in between:

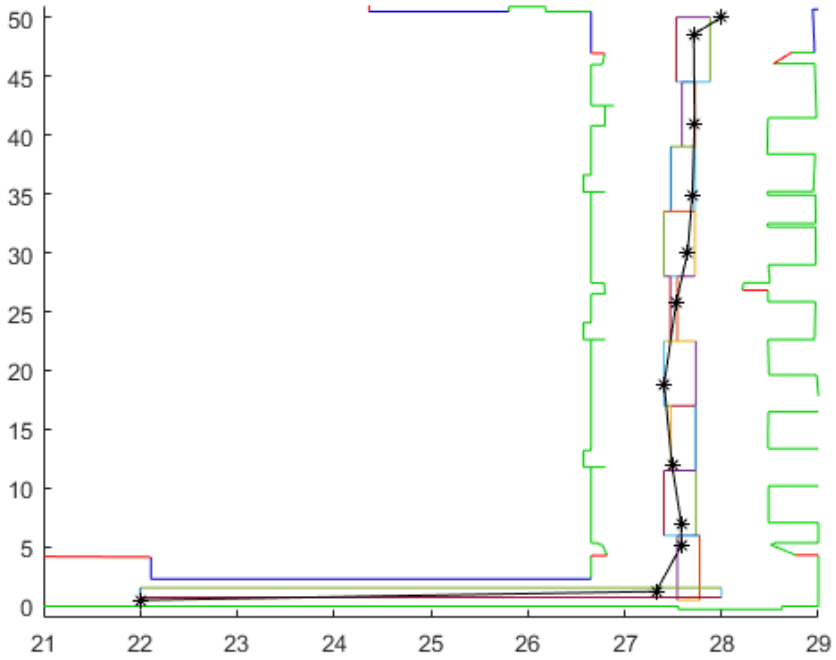

Figure 8: trajectory generation example

5.2.2 Environment evulation

We generate multiple trajectories in the enivronment, and test the particle filter performances depending on the starting point. The goal is to identify where the particle filter performance well in order to implemented in the same regions with the real robot.

5.2.3 Particle filter

This function takes as input the options for the simulation. Namely, the selection function choice, the redistribution function choice, the number of particles, the first distribution of the particles (this can be helpful to run multiple simulations with the same first distribution of the particles), the number of rayons used, the type of sensor used in the simulation, and the simulation trajectory. After the simulation is done, the function returns a struct "*Data*" with all the necessary information about the simulation (the number of particle (during each iteration in case we change the number of particles), the particles distribution in each iteration, the number of iterations, the time of simulation for each iteration, the convergane and redistribution flag in each iteration and the trajectory, the estimation position at each iteration and the absolute error at each iteration).

Note

The function can be stopped after convergance, after a certain number of iterations or after one iteration.

6 Perspectives

- Change the control of particles and use transformation matrix of the robot (see C++ code).
- In our simulations, we supposed that particle filter converge always to a the right position (when enough particles are used). This was proven to not be always true when we reduce the number of particles, that's whay we should also check the errors and not only the iteration to convergance.
- We should simulate not using the particle filter at each iteration (we assume here that a particle filter iteration takes longer than the control iteration).

References

[1] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University-Computer and Information Sciences*, 2021.

[2] Frank Dellaert and Wolfram Burgard. " computer science department, carnegie mellon university, pittsburgh pa 15213* institute of computer science iii, university of bonn, d-53117 bonn.

[3] Jeong Woo, Young-Joong Kim, Jeong-on Lee, and Myo-Taeg Lim. Localization of mobile robot using particle filter. In *2006 SICE-ICASE International Joint Conference*, pages 3031–3034. IEEE, 2006.

[4] H Koeslag. *Multi robot SLAM pose estimate enhancement*. PhD thesis, Faculty of Science and Engineering, 2007.

[5] Paul Michael Newman. C4b—mobile robotics. *Accessed*, 2:2011, 2003.

[6] Philippe Lambert. *Contribution à l'autonomie des robots: vers des missions à garantie de performance incluant l'incertitude de localisation en environnement intérieur connu*. PhD thesis, Université Montpellier, 2021.

[7] David Filliat. *Robotique mobile*. PhD thesis, EDX, 2011.

[8] Christian Musso. *Contributions aux méthodes numériques pour le filtrage et la recherche opérationnelle*. PhD thesis, UNIVERSITE DE TOULON, 2017.

[9] James E Baker et al. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, volume 206, pages 14–21, 1987.

[10] John J Grefenstette. *Genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms*. Psychology Press, 2013.

[11] Yiploon Seow, Renato Miyagusuku, Atsushi Yamashita, and Hajime Asama. Detecting and solving the kidnapped robot problem using laser range finder and wifi signal. In *2017 IEEE international conference on real-time computing and robotics (RCAR)*, pages 303–308. IEEE, 2017.