Robot Localization with Particle Filters

An Information Theory Design Report

Sarah Brown
College of Engineering
University of Oklahoma
Norman, OK

***Abstract* — An information theory project that features the use of a small robot run on an ESP32 system. This robot uses particle filters to localize itself on a small and simple map. By locating itself on a map, the robot then would be able to path plan from its starting location. By continuing to localize, a path can be more accurately planned.**

Keywords—Filtering, Particle Methods, Resampling, Particle Filter

I. Introduction

A. Description of Problem

Robots are now used for many areas and applications. Many areas where robots are used, path planning is required to complete its task. However, path planning requires not only that the robot knows where it is trying to get to, but also where it is starting from. Robot localization can many various sensors and data sources. Some of these are GPS, lidar, and ultrasonic sensors. In addition, odometry data can also be used to help localize a robot. When a robot is localized, it can then continue on its path towards its goal.

B. Problem Solution

Particle filters are a type of Monte Carlo algorithm that can be used for signal processing. By simulating particles and sampling with these particles, an approximate solution can be found to a complex model. By repeatedly resampling, a more accurate representation of the system can be found. Over time, this representation can converge to the robot's location. With the use of a simple known map and a robot platform, it was possible to localize a robot.

C. Differences Between Orignal Plan and Implemented Plan

In the original project scope, the focus of robot localization was to be applied to the IGVC competition for path planning. IGVC is a robotics competition centered on navigating an autonomous vehicle through a course marked with lanes. However, while reviewing literature on filters over the semester, a smaller scope was selected to ensure completion. By refocusing on a smaller robot than the IGVC bot, the robot was localized on a smaller scale and the lessons learned can be applied on similar projects in the future.

II. Robot Platform

A. Hardware

The robot used for testing and localization was constructed with a small robot prototype chassis. This chassis held batteries, motors, a motor controller, motor encoders, an ultrasonic sensor, and an ESP32. The ESP32 used was the ESP32-WROOM-32. This part was selected for its Wi-Fi capabilities and its size to fit on the robot chassis. The ultrasonic sensor is used to relay distance information to be used for localization. In addition, the motor encoders are used to control and measure how far the wheels have turned.

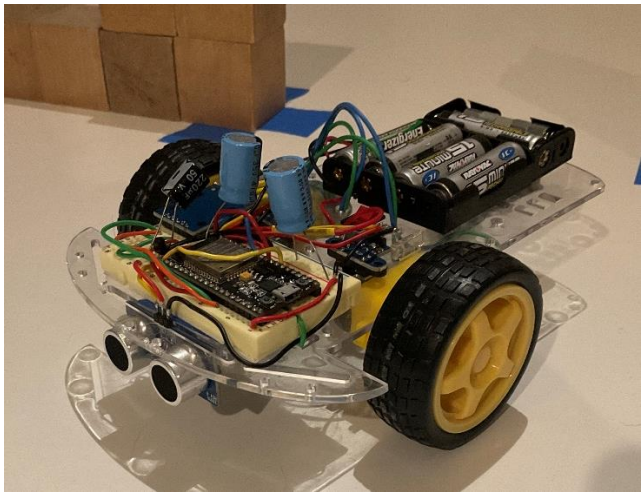


Figure 1. Small robot featuring an ESP32 to connect to computer running Python code via Wi-Fi

B. Software

The software for the ESP32 board was created using the Arduino IDE with the use of ESP32 and Wi-Fi libraries. The ESP32 was configured as a Wi-Fi client that the computer could then connect to. Once connected, it was possible to send and receive data. This data included distance data from the ultrasonic sensor and motor position data from the two motor encoders. The motors were in turn controlled with PWM and interrupts triggered by the motor encoders. This allowed for controlled positioning of the motors instead of deadreckoning by the amount of time the motors had been running. Once the

data was configured to be transferred, a filter could be developed with Python and run on a system with more power.

III. Particle Filters

A. Overview

A particle filter uses a known map or a system of known probabilities to estimate the outcome of the system. In the case of robot localization, a particle filter uses a known map and the pose of the robot. The pose of the robot is the position (x and y coordinates) and the orientation (reference angle theta) of where it is pointing.

B. Process

For a particle filter to converge, the algorithm repeatedly follows a series of steps. To begin with, an initial belief of the system is created. The simplest initial belief of a system is for uniform distribution across the known map. This is represented by created a number of particles and uniformly distributing them on the map. A higher number of particles increases the accuracy of the representation of the model, but it also increases computation complexity and time required. Following the initial belief, additional measurements are made using sensors and that information is used to adjust the weight of the generated particles. In addition to updating the weights of the various particles, the motion of the robot must also be taken into account as it moves to collect more data. The use of a motion model is used to change the positions of the particles with a degree of error to represent error in the robot system. New samples are then taken by using the newly weighted particles as a distribution model and using a gaussian around it. This process is then repeated and overtime the particles converge if the necessary data is received properly.

IV. Application of Particle Filter

A known map was created with the use of a wooden table and wooden blocks to create a diving wall. Measurements were then taken of this setup to complete the map.

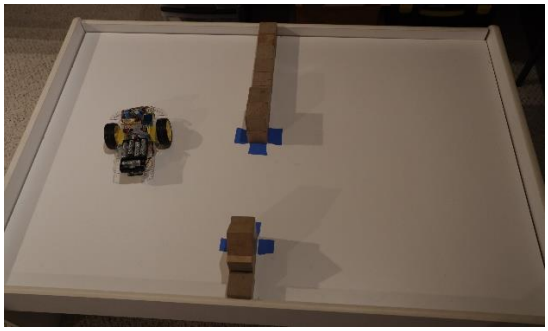


Figure 2. The map setup with robot on the table.

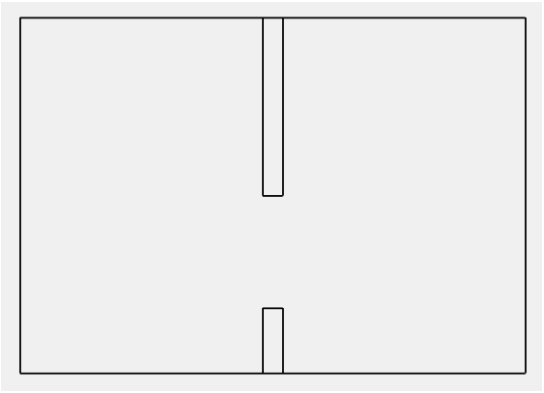


Figure 3. A graphical representation of the known map

Using this setup, an initial uniform distribution of particles was created with the robot represented as a point.

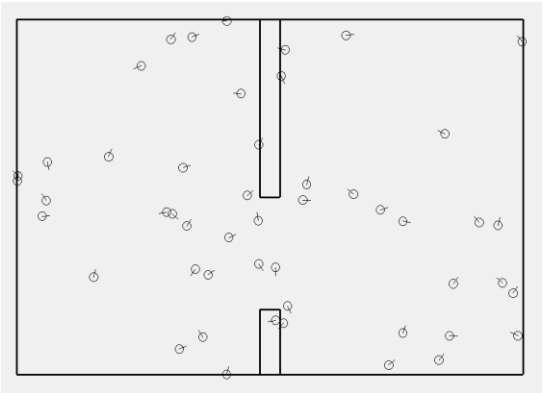


Figure 4. Initial uniform distribution of particles with 50 particles. Shows position and angle that the robot was facing.

This setup was tested with a lower number of particles, but the final number of particles was increased to 5000 particles. Following a measurement from the ultrasonic sensor (a set distance of 8cm was used for testing purposes before use of the robot), the weights of each point were updated based on how likely that point was based on the distance. This was done with a series of trigonometry calculations and with the use of the map as bounds. To localize the robot, additional data is then required. To collect this data, the robot turns in place towards the right. This allows more information to be collected, but the position of the particles must also be updated as the particle moves. As the robot is represented as a point, only the theta value must be updated indicating which direction the robot is pointing. Theta is updated based on how far the motors are turned which corresponds to how far around a circle the robot turned. Once the particles are updated, the new weights are used to resample and obtain new particles. This can be done as the new weights represent the distribution in space of how likely the robot is to be at the location of any of the particles. Resampling is with via weighted selection with replacement. By placing a gaussian around each point, new points can be selected and tuned using sigma values for x, y and theta.

V. Results

This particle filter was run on both test data with a constant ultrasonic distance of 8 centimeters as well as on the constructed robot field. The test data did not fully converge to a point due to only one measurement used repeatedly. However, it did allow points to be eliminated and a distribution can be seen in the results showing the distance away from the various walls in the map.

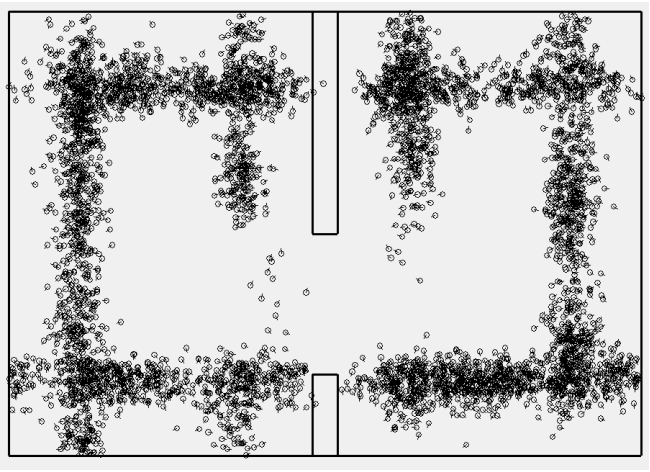
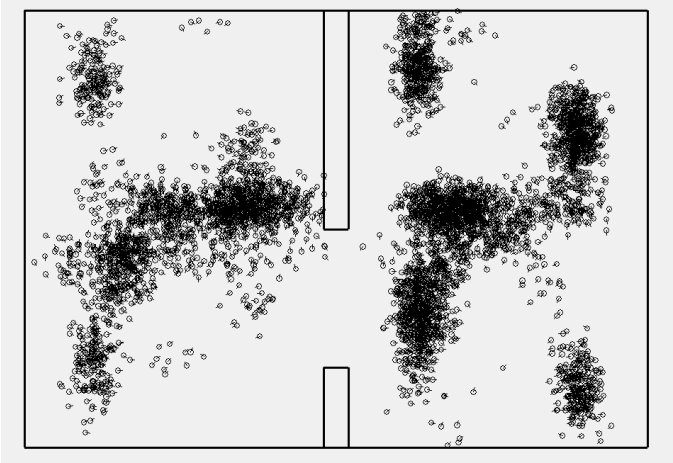


Figure 5. Screencap of result part way through a test run with ultrasonic distance set to a constant 8 cm.

During the robot run on the constructed field, the points converged reliably with only a few issues. Over a series of measurements, the points converged to where the robot was located on the map. However, in some instances, the robot converged in spot that mirrored its actual location due to similarities in the constructed course between the two sides.



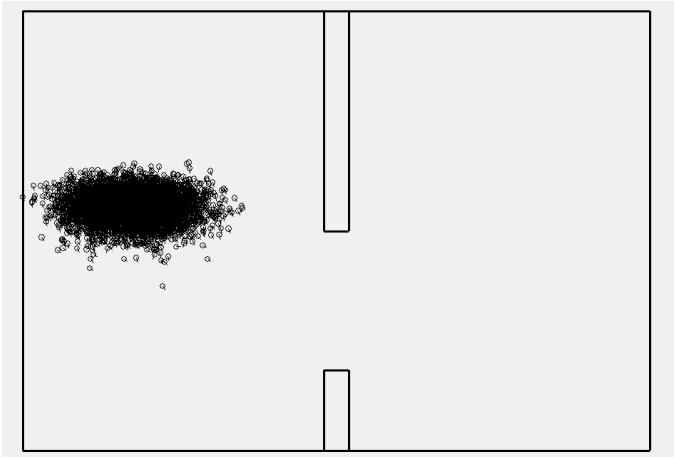


Figure 6 &7. Screencaps of particles converging during a robot run.

VI. Conclusions

A. Challenges

There were several challenging aspects while tackling this project. Once the project scope was revisited, this project became much more manageable. However, there were still issues while interfacing between the robot and the computer to run the particle filter. In particular, there were connection issues due to the ESP32 resetting due to sensitivity with voltage dips caused by motor movement, the ultrasonic sensor pinging, and Wi-Fi transmissions. This issue was solved by adding capacitors to the voltage rails to limit voltage dips. In addition, the graphics library that was used to display the particles throughout the filtering process slowed down the processing time vastly until adjustments were made. The robot faced some issues correctly converging due to the near symmetry of the map. While one side of the field was larger than the other, this still posed issues as they were very similar. This similarity sometimes resulted in the robot location converging in the mirroring position on the table.

B. Further Work

This work can be further developed by using the known robot position due to localization to path plan and navigate around the table. In addition, the knowledge gained with regards to filtering with this project can be applied to other robotics projects. In specific, it will be applied to improve path planning for an IGVC robot.

Code

Code for this project can be viewed at:
https://github.com/SarahBrown/InfoTheory

References

[1] "L9110 2-Channel Motor Driver - NCUT." [Online]. Available: http://me.web2.ncut.edu.tw/ezfiles/39/1039/img/617/L9110_2_CHANNEL_MOTOR_DRIVER.pdf. [Accessed: 16-Dec-2021].

[2] Leimao, "Leimao/particle-filter: Robot localization in maze using particle filter," GitHub. [Online]. Available: https://github.com/leimao/Particle-Filter. [Accessed: 16-Dec-2021].

[3] "Localize TurtleBot using Monte Carlo localization," Localize TurtleBot Using Monte Carlo Localization - MATLAB &

Simulink. [Online]. Available: https://www.mathworks.com/help///nav/ug/localize-turtlebot-using-monte-carlo-localization.html. [Accessed: 16-Dec-2021].

[4] “Numpy.random.normal,” numpy.random.normal - NumPy v1.21 Manual. [Online]. Available: https://numpy.org/doc/stable//reference/random/generated/numpy.random.normal.html. [Accessed: 16-Dec-2021].

[5] Rlabbe, “RLABBE Kalman-and-Bayesian-filters-in-python,” GitHub, 13-Oct-2020. [Online]. Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/12-Particle-Filters.ipynb. [Accessed: 16-Dec-2021].

[6] “A tutorial on particle filtering and smoothing: Fifteen ...” [Online]. Available: https://www.stats.ox.ac.uk/~doucet/doucet_johansen_tutorialPF2011.pdf. [Accessed: 16-Dec-2021].

[7] “Understanding the particle filter.” [Online]. Available: https://www.youtube.com/watch?v=NrzmH_yerBU. [Accessed: 16-Dec-2021].