Mobile robot position estimation and navigation

Raphaël Cherney Frédéric Wilhelm

I. Introduction

This report describes the implementation of a position estimation and navigation scheme using the e-puck mobile robot. The robot is placed in an arena with internal walls and senses the environment using a rotating distance sensor. Because the internal geometry is non-trivial, the robot can estimate its position by comparing its sensor readings to an internal map. Tests were done using the Enki simulator and with physical hardware. The robot was able to avoid obstacles, determine its position in the arena, and travel to user-defined waypoints.

II. Locomotion

A. The e-puck platform

The e-puck robot is a small, differential wheeled mobile robot developed at the Swiss Federal Institute of Technology in Lausanne (EPFL). For our experiments, the robot was remotely controlled through an off-board Linux computer using a Bluetooth connection. The robot has a number of on-board sensors, but for this project we only used the IR proximity sensors and a rotating distance sensor to control both stepper motor wheels.

B. Braitenberg controller

In order to make the robot avoid obstacles, we implemented a simple Braitenberg-style controller. Both wheels start with a forward bias, and a each wheel speed is adjusted by subtracting weighted values for the IR proximity sensors on the opposite side (see Figure 1). In this way, obstacles sensed on the right side of the robot will cause the left wheel to slow down or reverse, resulting in the robot turning to the left, away from the obstacle. Similarly, obstacles on the left side cause the robot to turn right. This simple controller allows the robot to effectively avoid running into walls or other obstacles.

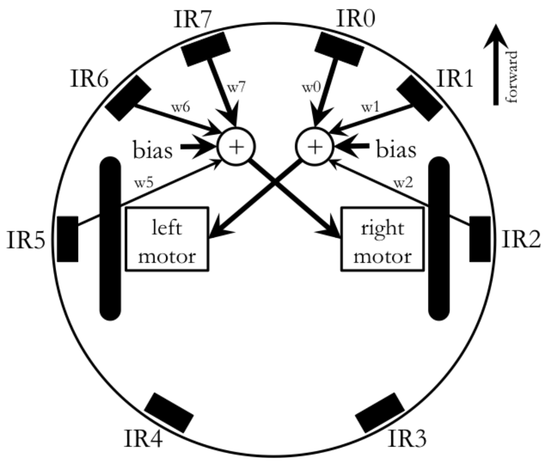

Figure 1: Braitenberg controller.

A problem with Braitenberg controllers is that they can often cause the system to get stuck in corners or dead-ends. This happens when the forward bias and weighted inputs cancel each other out. In this case, the robot stops moving, the sensor inputs remain the same, and the robot remains stuck. In order to alleviate this problem, we added some noise to controller which allowed the robot to (randomly) get out of these dead-end situations.

Note that we are using the IR proximity sensors to avoid obstacles instead of the rotating distance sensor. This is because the proximity sensors have a faster update rate than the rotating sensor. Since the proximity sensors are statically mounted to the robot, we do not have to wait for a turret to check a given direction. This allows us to have a quicker reaction time and helps us avoid obstacles even at higher speeds.

C. Gradient ascent

In addition to simple obstacle avoidance, we implemented a path planning algorithm to guide the robot to waypoints. This was accomplished using a gradient ascent approach. The user defines a waypoint using the map from the Enki interface. A *gradient map* of the entire arena is then created using a gradient propagation algorithm often referred to as wavefront expansion or grassfire. In our implementation, the goal cell is given a value of 255 and all other connected cells are given a value based on the minimum Manhattan distance to the goal though accessible cells ($value = 255 - distance$). The maximum (positive) slope at a given point gives the desired direction (since it will, by definition, lead to a cell closer to the goal). Figure 2 shows an example of the gradient map and the resulting path. Note that what we are calling a gradient map does not actually contain the gradients at each point; rather, it contains the information from which the gradient can be found.

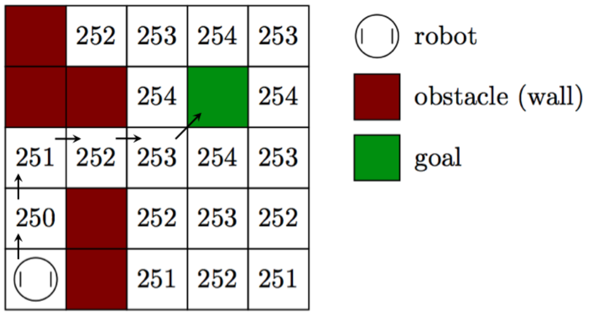

Figure 2: Gradient ascent algorithm leading to goal.

III. Localization

A. Distance histogram matching

The robot is equipped with a rotating distance sensor that enables the detection of obstacles in 64 different directions around the robot. The result is a distance histogram represented as a vector of 64 dimensions (Figure 3).

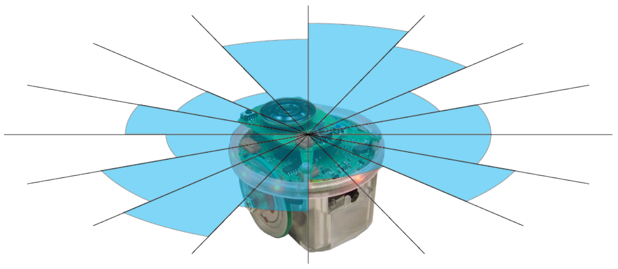

Figure 3: Proximity histogram obtained though the rotating distance sensor. The real resolution is 64 scans per turn.

The key principle of our localization model is to see this histogram as a characteristic of any point in the map, given the obstacle positions. Said differently, for each accessible point of the map, we can associate the histogram that would be read by the sensor when the robot is at that point with direction zero (i.e. in the x-axis direction). For simplicity as well as for computational reasons, the map is sampled with a grid, each cell having it's own corresponding histogram.

B. Probability distribution

Given the proximity histogram of each cell of the grid, we can describe the estimation of the robot's position as a *probability map* defined over the accessible space. The probability of being in a given cell will increase when the distance between the two histograms (for the best direction of that cell) decreases. Our algorithm has several different steps:

1) **Compute the best direction for each location and fill in the probability map**
For each accessible point (x, y) (whose corresponding histogram is known), compute the robot's orientation for which the matching between the measured histogram and the emplacement histogram is the best (using the Euclidian distance). The resulting direction is the *most probable direction supposing the robot is in* (x, y).
The matching (the Euclidian distance) between the two histograms is stored for each position, resulting in a probability map (which is not explicitly stored; what is really stored is the distance between the histograms, which has to be minimized over the space, which corresponds to the maximization of the location probability).
2) **Update the new probability map taking into account the prior knowledge of the position and direction**
For the cells in the neighborhood of the previous position, the probability is increased. This corresponds to a decreased distance in the histogram-distance map. We implemented this by multiplying the cells within one cell of the previous estimated position by a factor $\alpha < 1$ (experimentally, $\alpha = 0.3$ was a good value). This prevents the robot from *jumping* from one position evaluation to another when

there are several maxima of the probability map.

As an additional step to avoid jumps in the position estimation, we penalize the difference between the previous and proposed orientations: if the best direction of a cell (x, y) (computed in step 1) is far away from the previous direction (in the angular sense), it is very unlikely to be the current position given the time needed by the robot to turn. This probability is thus penalized. We implemented this by adding a term to the histogram-distance map, proportional to the (absolute) distance between the previous direction and the cell's best direction, multiplied by a weight (experimentally fixed at 1000).

3) **Evaluate the most probable position**
By minimizing the modified histogram-distance map, we can find the most probably position allow with the corresponding orientation. *We thus have a complete estimate of our robot's position.*

IV. Results

We began our testing using the Enki simulator. Using this tool, we were able to try out our controllers and assess various methods for improving our position estimate. Because we designed the controllers to work in this simulated environment, we got very good and consistent results once our system was tuned. The Enki simulator includes visualizations for the Probability map (really a histogram-distance map) and the Gradient map (see Figure 4). The Probability map shows the difference between the measured histogram and the known histogram (also taking into account the prior position knowledge). This gives us some measure of the probability of being in a particular position. The green square represents the most probable position and the blue line shows the corresponding orientation. The Gradient map is used to find the path to the waypoint marked in red (lighter cells are "higher"). These visualizations are also used when testing the physical hardware, explaining why Enki is still present in the physical robot testing code.

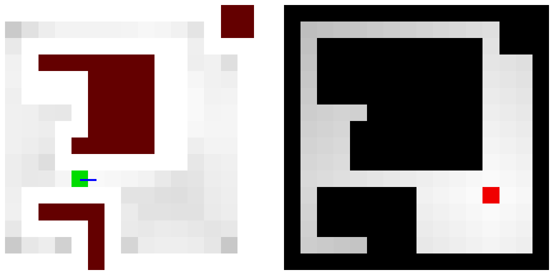
(a) Probability map (b) Gradient map

Figure 4: Visualizations for the robot position and desired direction.

When we moved from simulation to the physical hardware, we encountered several issues. While the robot was still able to avoid obstacles, the way-point navigation was extremely poor. The position estimates were not consistent, causing jerky and inconsistent behavior. This is due to the fact that the real rotating distance sensor has a relatively low refresh rate (compared to the speed of motion). This means that the distance histogram was not fully updated each time we tried to estimate the position. Instead, the robot moved while in the middle of a measurement, meaning that the histogram matching was always off.

In order to alleviate this problem, we implemented a simple finite state machine with two states: a *scanning state* and a *moving state*. The robot simply alternated between these two states based on predefined timing. During the scanning state, the robot would remain stationary to complete a distance scan and update its histogram. It would then have a brief moving state where the motors were active before stopping and completing another scan. This resulted in slower movements but dramatically improved position estimation. Because the robot was able to have a current and correct histogram each time it estimated its position, we were actually able use the same code from the simulator to navigate around the arena.

V. Conclusion

Localization and navigation are important topics in the field of robotics. Here we presented a simple setup to control an e-puck mobile robot. By comparing sensor values to a grid of expected sensor values, we are able to estimate the position and orientation of the robot within an known arena. Using this position estimate and a user-defined waypoint, we determine the path using a gradient ascent algorithm. This is an interesting experiment, but real-world applications would likely need to be more advanced and robust. The scheme presented here requires a well-known and controlled environment that is not always available. More advanced techniques such as simultaneous localization and mapping (SLAM) along with better controllers can deal with less-controlled, unknown, or changing environments. Better sensors with faster update rates, sensor fusion with Kalman or particle filters, more fine grain maps, more efficient search algorithms, active beacons, and the use of multiple robots can all help real-world robots achieve better position estimation and path-planning, enabling a wide range of user scenarios.