Gabriel Laverghetta

CDA 4621 Control of Mobile Robots

Lab 3

**Mathematical Computations**

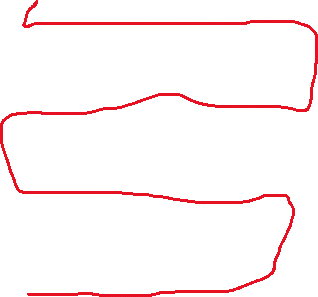
Task 1

For the first task, the robot must estimate its current position using readings from its sensors and IMU. The goal is to traverse all 16 cells in the arena, printing information about visited cells and the robot’s current pose at each cell. There are two possible initial states: starting from cell 13 oriented east and starting from an arbitrary location and orientation.

Since the robot may follow any desired path through the environment, a good strategy is to first reposition the robot at cell 13 facing east, then move the robot through each cell following a standard path. No matter where the robot starts, it can be repositioned in cell 13 using the following steps:

1. Turn until the IMU reports that the robot is facing south.
2. Move forward until the front sensor reports a wall directly ahead.
3. Turn until the IMU reports that the robot is facing west.
4. Move forward until the front sensors reports a wall directly ahead.
5. Turn until the robot is facing east.

Note that if the starting location is known to be in grid cell 13, this initial step is unnecessary. From here, the robot will move through each cell in the arena, following the general path shown in Figure 1.



|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Figure 1: Path taken by robot in Task 1 after it is repositioned in cell 13.

Each turn performed by the robot is 90 degrees. In addition, the direction of the turn alternates every two turns. When moving forward, the robot may perform “long” motions (30 inches) or “short” motions (10 inches). The length of the motion alternates after each turn is performed. The position sensors are used to determine when forward motion should end and a turn should begin.

Upon reaching a new cell, the robot should print information about visited cells along with its current pose. Each cell is stored in a 4x4 matrix, where an “X” indicates that the cell has been visited and a “.” represents that the cell has not yet been visited. To keep track of the robot’s current grid cell, two variables i and j are used. These variables are initialized to 3 and 0, respectively. For every 10 inches moved forward by the robot, the value of j should be incremented if the robot is facing east or decremented if the robot is facing west. In addition, the value of i is decremented if the robot is facing north. Whenever either of these values changes, the robot should mark the current grid cell as visited and print the necessary information.

The robot’s pose is given by s = (x, y, n, θ), where x and y represent the robot’s position in global coordinates, n represents the current grid cell, and θ is the robot’s orientation.

* The method used to determine the values of x and y varies depending on the robot’s orientation. First, note that the robot’s x value does not change while the robot is performing “short” forward motions (i.e., when the robot moving to the north). When the robot is facing east, the x value is given by (20 – front\_distance\_sensor). When the robot is facing west, the x value is given by (front\_distance\_sensor – 20). Similarly, the value of y does not change while “long” forward motion is being performed. At all other times, the value of y is given by the following (20 – front\_distance\_sensor).
* The current grid cell number can be found by manipulating the i and j values introduced earlier: n = 4i + j + 1.
* The robot’s orientation is simply retrieved from the IMU reading.

All of this information is combined in the state diagram shown below.

Task 2

This task featured the same environment and goals as the previous task, which allows many of the same results to be reused. Once the robot’s starting position is known, the robot will move to grid cell 13 and traverse the environment in the same manner as before. The change for this task is that the robot must use triangulation or trilateration to determine its initial pose. The problem is illustrated in Figure 1. The robot begins at an arbitrary location in the environment and must use the known locations of landmarks in the environment to estimate its initial pose.

Graphical user interface, application, Word

Description automatically generated

Figure 1: Trilateration in Task 2, with three known landmarks

Each of the landmarks is located at one of the corners of the arena. To perform trilateration, start by forming equations for three circles, each of which is centered at one of the landmarks.

(1)

(2)

(3)

The center of each circle (xi, yi) is known and given in the following table:

|  |  |  |
| --- | --- | --- |
| Cylinder | X | Y |
| Yellow | -20 | 20 |
| Red | 20 | 20 |
| Blue | 20 | -20 |
| Green | -20 | -20 |

The values of R1, R2, and R3 are determined using the forward distance sensor.

Expanding equations (1), (2), and (3) yields the following:

(4)

(5)

(6)

Subtract the second equation from the first and the third equation from the second:

(7)

(8)

The robot’s initial x and y values are determined as follows:

(9)

(10)

(11)

(12)

(13)

(14)

(15)

F (16)

(17,18)

Implementing this in Task 2 requires the robot to rotate in-place until it has detected three of the colored cylinders, e.g., the yellow, red, and blue cylinders, and measured its distance from each of them. In ideal conditions, the robot would always be able to use the first three cylinders it detects. However, the robot’s distance sensor has a range limit of fifty inches, so the measured distances might be inaccurate if the robot is too far from a cylinder (for example, if it begins in grid cell 16 and it is trying to measure the distance to the yellow cylinder). For this reason, the robot begins by rotating a full 360 degrees in-place and measuring the distances to each of the four cylinders. The cylinder that is farthest away is not used for trilateration. Once the appropriate cylinder trio has been determined, the previously discussed equations are used to determine the robot’s starting location.

Following this, the robot must reposition itself in grid cell 13 facing east. This is accomplished by rotating in-place until the green cylinder is in view, moving forward until the distance sensor reports that the cylinder is directly ahead, then rotating until the robot faces east. The robot then traverses all 16 grid cells using the method discussed for Task 1.

Task 3

This task changes the arena from an open environment to one containing several obstacles. The goal, again, is to visit every cell in the arena, printing information about visited cells and the robot’s pose at each cell. For this task, the robot will always start at an unknown location and must employ the particle filter to estimate its starting position. The cells are organized into a 16x4 matrix, with each entry indicating if a wall is adjacent to the cell. “W” indicates a wall is present, and “O” indicates no wall. Since the maze will be given in advance to the robot, this wall configuration matrix should be included in the controller for the task and used by the robot to complete the task. In addition, if any modifications are made to the maze, the matrix should be updated accordingly.

The particle filter algorithm requires the robot to keep track of the number of particles that have been assigned to each cell. A 4x4 matrix is used to store particle information, with each entry in the matrix corresponding to a grid cell. The steps of the algorithm are listed below:

1. Initialization. Distribute the 80 particles uniformly across all 16 grid cells.
2. Measurements. Detect if there are obstacles to the west, north, east, or south of the current robot location. Match those measurements with the probabilities given in slide 27.
3. Measurement estimation. For each cell, form an initial estimate that the robot is in that cell. The initial estimate is equal to the sum of the three probabilities found in step two. Also, add each probability to a running sum as each estimate is made (this is needed for the next step). Skip cells that have no particles in them for this step.
4. Normalize probabilities. Divide each probability by the sum found in the previous step. Skip cells that have no particles in them.
5. Importance factor. Multiply each probability by the number of particles currently in that cell (FIXME are you even supposed to do this? b/c they didn’t. DEFINITELY DO IT THE SECOND ITERATION) .
6. Resampling. Multiply each probability by the total number of particles. Sort the probabilities into decreasing order. Assign particles to the cells with the highest probabilities first. Take the ceiling of the probabilities when assigning particles. (FIXME RREMOVE THIS Round the probabilities to integers when assigning particles). If multiple cells have the same probability: pick one at random and give it particles.
7. Motion update. Move the robot in some direction (up, left, right, or down) and update the particles. If there is a wall blocking the motion, the particles in that cell are unchanged. Else, move a percentage of the particles to the new cell. The percentage is obtained from the probabilities in slide 27. For example, if the robot is moving up a cell, 70% of the particles will move up, while the remaining 30% remain at the same cell.
8. Repeat from step 2 until a high enough probability has been obtained. (90%?)

**Conclusions**

This lab was a step up from the first in terms of the breadth and depth of the concepts involved. Many important lessons from the first lab, such as straight-line motion and rotating the robot in-place, returned in this lab. The two aspects of the lab that proved to be most troublesome were implementing the algorithms in practice and being too hasty in starting the coding process. On the other hand, unit conversions were much easier to deal with than they were previously.

As I studied the material presented in class, I did not find it very difficult to understand conceptually. The air-conditioning analogy was very illuminating, and it helped me grasp the purpose of PID at a high level. Likewise, the Bug algorithms seemed simple enough at first. The Bug 0 algorithm in particular has only four steps, and none of these steps is very complicated. When it came time to program in Webots, however, these concepts were not simple to implement. Using PID correctly is a delicate task, with several intermediate values to manage while juggling data from multiple sensors and possibly the camera. And, implementing the Bug 0 algorithm took quite some time, as there are quite a few cases to consider. Overall, I learned that I should expect the concepts discussed in class to be challenging to implement in Webots, even if they seem straightforward in theory.

My progress in Lab 2 was hampered by my tendency to begin writing code before fully understanding the solution I intended to implement. A good example of this bad habit is evident in my controller for the Bug 0 algorithm, which is rather messy in multiple places. Drawing out a state diagram or making notes by hand would have assisted me greatly before beginning the programming process. Although my code gets the job done, it is not an elegant solution, and it may prove problematic if I need to review it in the future. This issue could have been avoided by spending more time understanding the problem beforehand and by simply allotting myself more time to complete the assignment.

Converting between units was one of the three problem points I encountered in Lab 1, but I did not find it to be nearly as challenging for this project. Of course, there were still instances in which I forgot to make a conversion, but when this occurred it did not take long to recognize what had gone wrong and correct the mistake. I was aided in this project by the conversion functions I had created for Lab 1, along with the documentation I included in my Lab 1 report. This has made it clear to me that writing good documentation is crucial, even if it may seem wasteful at the time.

All in all, most of these problems involved a lack of preparation and an overly optimistic outlook. I have learned that I should temper my expectations regarding the difficultly of these labs, and I intend to spend more time planning out my approach before diving into the code itself. On a positive note, the lessons I learned regarding documentation will surely aid me in the future. With two labs remaining in the course, I will no doubt have plenty of opportunities to make use what I have learned.