Gabriel Laverghetta

CDA 4621 Control of Mobile Robots

Lab 3

**Mathematical Computations**

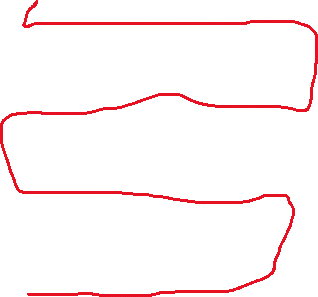
Task 1

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.

Reposition in cell 13

Long motion forward

Short motion forward

This task required the robot to utilize PID control to move towards the goal while keeping its distance from the side walls. The robot starts at a distance of 50 inches from the end wall and must move forward 40 inches, stopping exactly 10 inches from the end wall. This motion must be completed in 30 seconds. To complete the motion within this time, the robot measures its distance from the goal (using the front distance sensor), and it keeps track of how much time has elapsed since the simulation began. Dividing the remaining distance by the remaining time yields the desired linear velocity V.

PID control is used to keep the robot centered in its path. Let y(t) be the distance from each side wall as measured by the distance sensors. Let r(t) = 3.624 in., the optimal distance from each wall. When both side distance sensors report this value, the robot is directly in between the walls. The error e(t) is given by the equation below:

(1)

The error is positive when the robot is too close to the wall, and it is negative when the robot is too far from the wall.

The motor velocity control function is as follows:

(2)

The constant Kp was determined through experimentation. After testing different starting orientations (in the range of -30 to 30 degrees, inclusive), using 2.5 for the Kp value yielded the best results. This value caused the robot to stabilize its motion and center itself in the corridor fastest, even when starting at an orientation of 30 degrees.

After determining the error for each motor, the two errors are compared to determine which is larger. If the absolute value of the left sensor’s error is larger, the robot is closer to the wall to its left. In this case, the robot’s left motor should be sped up to move it away from the left wall. The new velocity will be the sum of V and the left motor velocity control value. The right motor’s velocity is determined by solving the following equation for vr:

(3)

A similar procedure is used when the right sensor’s error is larger.

Task 2

This task can be divided into two parts. First, the robot must rotate in place until it detects the yellow cylinder with its camera. As was determined in the previous lab, setting one motor to a speed of 2.92 and another motor to -2.92 results in the fastest possible rotation. For this task, the robot is rotated at a speed equal to 1/3 of this. The goal is to turn the robot so that the cylinder is near the center of its field of view, and turning too quickly could lead to an inaccurate rotation. The robot will turn counterclockwise until the cylinder is within five pixels of the center of the camera view.

Once the robot has turned to face the cylinder, the next step begins. As with the previous task, the robot’s linear velocity is equal to the distance remaining (measured using the front distance sensor) divided by the remaining time. The robot moves towards the cylinder, using PID control to keep itself aligned with the cylinder. For this task, rather than using the left and right distance sensors to calculate the error, a better approach involves tracking the position of the cylinder in the image taken by the camera. This error will be zero when the cylinder is exactly in the center of the image, which is the case when its X position value is 40. Positive error indicates that the robot is facing too far to the right, while negative error indicates that the robot is facing too far to the left. The error is given by the equation below:

error = camera\_center\_x - position / (camera\_center\_x / 8) (4)

When the error is positive, the left motor should be sped up to realign the robot with the cylinder. The new velocity of the left motor will be as follows:

left\_motor\_velocity = (linear\_velocity + error) \* Kp (5)

A similar procedure if the error is negative. The right motor will be sped up to compensate.

Task 3 - Corridor

This task required the robot to implement wall-following behavior. The robot maintains a distance of 7 inches from the wall it is following as best it can. The PID techniques from previous tasks were used to keep the robot away from the walls. Whenever the robot detects a wall in front of itself, it will attempt to turn left or right, depending on which controller is currently active. If the left turn controller is active, and the robot detects a wall to its left and in front, it will perform a 180 degree turn. If the right turn controller is active, and the robot to detects a wall to its right and in front, it also turns 180 degrees.

The task is finished after the robot traverses the entire path, which is about 220 inches. Whenever the robot performs a 180 degree turn, it adds twice the number of inches it has travelled up to that point to this value. For example, if the robot has travelled 30 inches in total before it turns 180 degrees, the total length of the path increases to 280 inches.

Task 3 - Maze

The maze component of Task 3 requires some changes to be made to the robot’s controller. In particular, if the robot only makes left turns or right turns, it will be impossible to complete the task from some starting positions. To compensate for this, the robot will switch from the kinds of turns it is performing (left turns become right turns and right turns become left turns) after performing two 180 degree turns. This prevents the robot from getting stuck and allows it to make progress in the maze. The task ends at the top left corner of the maze, which is reached when all three distance sensors report values less than or equal to the stopping distance (8.4 inches).

Task 4

The final mandatory task involves the implementation of the Bug 0 algorithm. This algorithm assumes that the robot knows the direction to the goal from any point, including its start position. In addition, the robot should be able to sense obstacles locally. Either left turns or right turns, but not both, may be performed. The four steps of the algorithm are as follows:

1. Head towards the goal, if not blocked by an obstacle.
2. Turn upon reaching an obstacle.
3. Wall-follow the obstacle until the robot has another straight path to the goal.
4. Repeat until the goal is reached.

The robot begins by rotating in-place until the goal is within three pixels of the center of its camera view. From there, the robot moves directly to the goal, maintaining a constant motor speed of 3.5 rad/s for each of the motors. When an obstacle is encountered, the robot begins turning until two conditions are met: the front sensor no longer reports any obstacle close by and the IMU reading is equal to 0 ± 0.05, pi/2 ± 0.05, or pi ± 0.05. This stipulation ensures that the robot stays close to the wall while wall-following. The task ends when the goal is directly in front of the robot.

Left turns are used in this implementation of the algorithm, but right turns may also be used with only small changes to the code. Using right turns may improve the speed at which the robot finds the goal in some circumstances, but it could also decrease performance in others. Both solutions will have the same overall success rate if all possible starting positions are taken into account.

**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.