To: Dr. Berry

From: Team Moravec, Devon Adair and Hunter LaMantia

Date: 1/20/2019

Re: Wall-Following: PD Control Lab 3

**Introduction**

The purpose of this lab is to use proportional (P) or proportional-derivative (PD) control to make our robot follow walls. The robot’s IR sensors detect the robot’s distance from nearby walls and the robot attempts to follow them within 4 to 6 inches. The robot is also capable of handling hallways, inside corners, and outside corners. If the robot loses a wall and fails to find it again, it will randomly wander until it finds a new wall to follow.

**Methods**

We use state machines to let the robot determine how it should move. The states tell the robot if it is following a right wall, following a left wall, following a hallway, has lost a left wall, has lost a right wall, is randomly wandering, or has found a new wall. The robot also uses a flag within some states to determine if an obstacle is in front of it. The states and flag are set by the IR sensors.

When the robot is in the state of following a left or right wall, it first checks if there is a wall in front of it. If there is, it will back up a few inches to avoid future collisions and then turns 90 degrees and follows the new wall. If there is no wall in front of it, it determines its distance from the wall. If it is within 4 to 6 inches from the wall, it moves forward. If it is too close or too far away from the wall, it uses P control to get back on track. Physically, the robot does a series of pivots to get back to its correct position. Our P control was smooth enough that we did not need to set up PD control.

We also set up an emergency run-away feature for if the robot gets close enough to the wall that P control would swing its backside into the wall. The robot pivots 90 degrees towards the wall so that it faces the wall without colliding with it. It then moves backward a few inches and pivots 90 degrees back to be parallel with the wall again.

When the robot is in the state of having detected two walls, it calculates how close it is to both of them. If it is an equal distance between them (within about an inch) it will move forward. If it is closer to one wall than the other, it alternates spinning and pivoting to get back on track.

If the robot is following one wall, but it loses that wall, it moves forward a distance roughly the length of its body and spins 90 degrees towards the wall it was previously following. It then moves forward the length of its body and checks whether or not it has found the wall again. If it finds the wall, it follows it. If it does not find the wall, it tries to find it two more times before giving up.

If the robot has either not found a wall yet or has given up on finding a lost wall, it will enter the state of randomly wandering. In this state, the robot randomly generates a distance for its left wheel to travel, a distance for its right wheel to travel, and directions for both wheels to move. It does this until it finds a new wall to follow.

When the robot finds a new wall, it checks the front wall flag. If the wall is in front of it, it will turn 90 degrees to the right and follow it. If the wall is not in front of it, the robot will just follow it like normal.

*1. What does diagram for the 3 layer subsumption architecture look like?*

N/A

*2. What did the robot do when it encountered a corner while wall following?*

When encountering inside corners, the robot reverses a few inches to ensure that it won’t bump into the wall in front of it. It then turns 90 degrees away from the wall it is currently following and starts to follow the new wall.

*3. What did the robot do when it encountered doorways and/or corners?*

When encountering outside corners or doorways, the robot turns 90 degrees towards the wall it was following and move forward until it detects a wall again. If it does not detect another wall, it goes into random wander behavior.

*4. When tuning the proportional controller and/or derivative controller, did the robot exhibit any oscillating, damping, overshoot or offset error? If so, how much?*

The robot had oscillations that we never completely got rid of, but this is inherent to only using pivots to straighten out. Our final values left us with very small oscillations, but while tuning the it we had to deal with very drastic swinging when we used a too high P gain value. On the other hand, lowering the P gain value damped our course correction enough to make the robot incapable of returning to the wall when it got too far away.

*5. What were the results of the different P and D controller gains? How did you decide which one to use?*

We found that P gains of 0.5 or lower typically did not allow the robot to adjust its course enough to stay on course, and values of 1.0 or higher cause the robot to swing around too much and hit walls. We settled on a value of 0.75, as that seemed to keep the robot on course without hitting walls.

We did not use the PD controller, so we have no values for that.

*6. How accurate was the robot at maintaining a distance between 4 and 6 inches?*

When the robot is between 4 and 6 inches it does a great job of staying there for a reasonable distance. However, when it gets too close or too far from the wall it typically takes a long time for it to finally get back between 4 and 6 inches.

*7. Did the robot ever lose the wall?*

After going around an outside corner, the robot typically needs to move forward a short distance before it is able to straighten itself out. This is normally not a problem, but when two outside corners are encountered within a short period of time, the robot loses the wall about half the time.

*8. Compare and contrast the performance of the Wander and Avoid behaviors compared to last week’s lab.*

The Wander behavior was copied straight from our previous lab, but modified to fit the new runToStop() function. We also simplified the function, removing unnecessary speed randomization. However, the Avoid behavior is completely different. Previously our robot would either stop in front of obstacles or move away from them without any target. Since we now want it to stay on the wall, we wrote obstacles in front of it to be treated as corners to go around, meaning we needed to special code other than our corner case handler. Obstacles between the robot and the wall (or just getting too close to the wall) is handled by pivoting towards the obstacle and backing away from it before straightening back out.

*9. What was the general plan to implement the feedback control and subsumption architecture on the robot?*

Feedback control worked by first detecting the distance from the wall and determining whether the robot is too close or too far from the wall. This distance for the wheels to pivot is then calculated by taking a constant distance (1/8 of a rotation) and multiplying it by the sensor error and our chosen gain value.

*10. How could you improve the control architecture and/or wall following/follow center behaviors?*

The method of course correction while following one wall was sloppier than it needed to be. Using a series of pivots resulted in some swinging that took a lot of work to minimize. We could have improved it by instead using the follow center version of course correction, where the series of pivots is replaced by a spin followed by a pivot. This is much smoother by default and would have saved us a lot of time.

*11. What does the overall subsumption architecture diagram with all 4 layers look like?*

N/A

*12. What was the pseudocode and flow chart for the program design?*

See Appendix

*13. Did you use any suppression and inhibition with the integration of Layers 2 and 3?*

In general, no state had to override any other state. Each state was distinct enough that there were never any overlapping cases. However, some states (namely, random wander) were only active when no other states were active, and therefore seemed to be suppressed.

*14. How did you implement the finite state machine to integrate the various behaviors? Did you use any inhibition and suppression to create layers in this behavior?*

(answered in 13 & 15)

*15. How did you keep track of the robot’s state and as it switched between behaviors?*

Each state was active when a specific combination of sensors detected walls. For example, follow right wall is active when only the right sensor detects a wall, and follow hallway is active when both the left and right sensors detect walls. Within the main movement method, there is a top level if-else statement that checks which state is active and calls the appropriate behavior.

**Results**

Our robot worked fine in the end, but there were some improvements that we could have made in retrospect. Our method of correcting the robot’s course when following one wall is different from that of following a hallway. With one wall, the robot does a series of pivots to correct its course, but in hallways the robot does a series of spins and pivots. The hallway method was much more smooth and effective, and had we thought of it at the start we believe our single wall following would be much smoother.

One problem that we spent a lot of time fixing was making our input distances work. For example, inputting forward(100) behaved the same as forward(10000). We eventually realized that the timer interrupt was preventing the movement behavior from finishing, preventing us from fine tuning our movements. This was fixed by turning the timer off when runToStop() is active.

Another problem we had was the robot overshooting its targets. This was an easier fix to figure out. We just cut the speed in half and allowed the robot to take its time. This had the secondary effect of making the robot move less distance between updating its sensors, making everything more accurate.

Despite the potential improvements, the robot did everything we needed it to do, and it did them well. The things we learned should make troubleshooting the next lab much easier.

**Appendix**

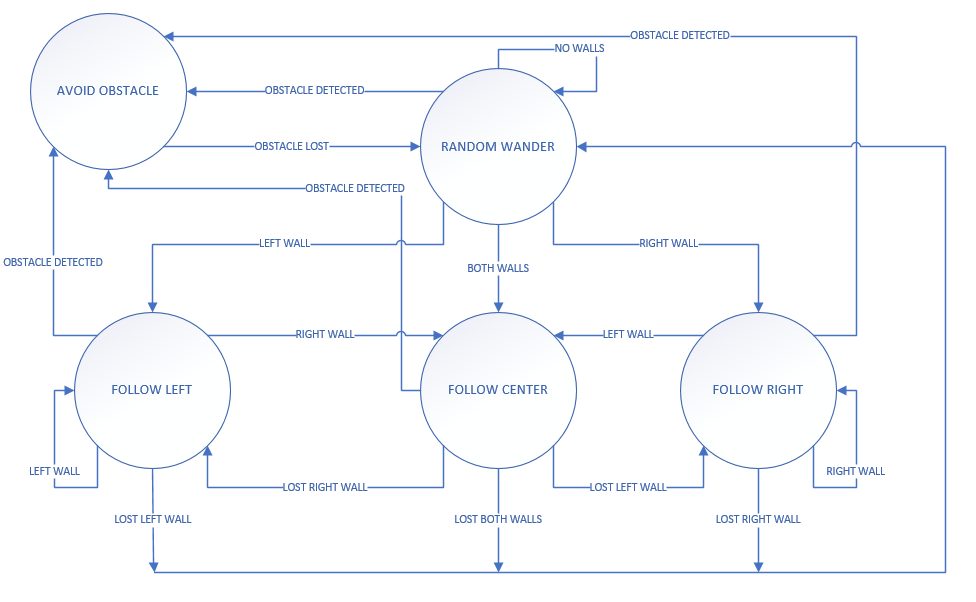


Figure 1: State Diagram

Table 1: State Transition Table

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Input** | **Output** | **Next State** |
| Random Wander | No Walls | No Walls | Random Wander |
| Random Wander | Obstacle Lost | Obstacle Detected | Avoid Obstacle |
| Random Wander | Lost Right Wall | Right Wall | Follow Right |
| Random Wander | Lost Left Wall | Left Wall | Follow Left |
| Random Wander | Lost Both Walls | Both Walls | Follow Center |
| Avoid Obstacle | Obstacle Detected | Obstacle Lost | Random Wander |
| Follow Left | Left Wall | Left Wall | Follow Left |
| Follow Left | Left Wall | Lost Left Wall | Random Wander |
| Follow Left | Lost Right Wall | Right Wall | Follow Center |
| Follow Left |  | Obstacle Detected | Avoid Obstacle |
| Follow Right | Right Wall | Right Wall | Follow Right |
| Follow Right | Right Wall | Lost Right Wall | Random Wander |
| Follow Right | Lost Left Wall | Left Wall | Follow Center |
| Follow Right |  | Obstacle Detected | Avoid Obstacle |
| Follow Center | Both Walls | Lost Both Walls | Random Wander |
| Follow Center | Right Wall | Lost Right Wall | Follow Left |
| Follow Center | Left Wall | Lost Left Wall | Follow Right |
| Follow Center |  | Obstacle Detected | Avoid Obstacle |