Peter Atsaves
Assignment 4 Maze Traversal Report

Introduction

The task we had to accomplish was to create a self-driving robot that can start at one end of a maze and travel to the end where there is a light source. The robot can't be moved or touched by us at any point during the run. Our approach was to use the walls to guide our robot. Since every maze is bounded by walls, that means you can follow the wall from the start to the end. We came up with a couple ideas such as scanning for open space or searching for light, but we thought that following the wall would be the simplest method. We wanted our robot to travel along the wall with the wall to the left. If there was a corner, we wanted our robot to turn right and continue. If our robot moved forward and lost track of the wall, it would do a 180. Outside of these two things, our robot would just continue straight along the wall.

Mechanical Design

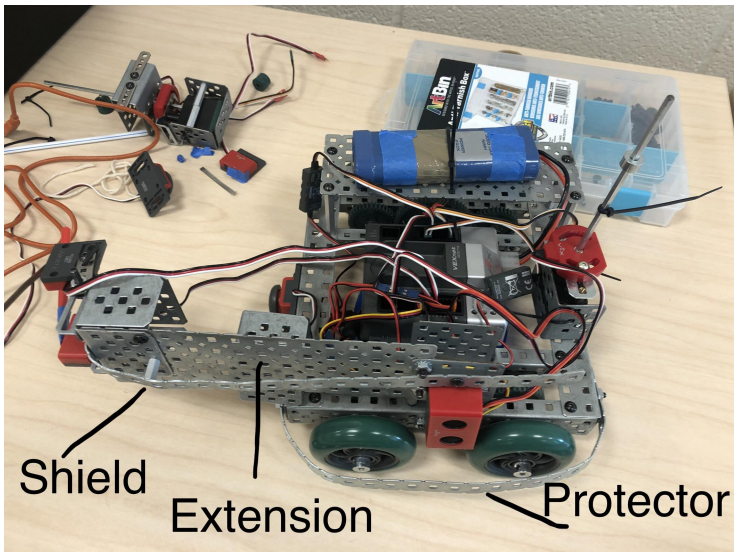


Figure 1

Figure 1 is our finished design of our robot. We planned to leave our robot the same besides just adding a bump sensor in front and an ultrasonic sensor to the side, but we had to add a lot more. Since our right motor is significantly stronger than the left motor, we could never get the

robot to go close to straight consistently, even if we adjusted the speeds. To compensate for this we added the extension. This is a strong metal piece that goes along the wall. Since our robot is traveling along the wall with the wall to the left, this extension helps stop the robot from turning into the wall. We noticed that just having this was not enough because our robot could get caught in the small openings of the walls. This leads to us adding the shield which is a metal piece that is curved. The shape is held by a half inch pillar that is screwed into the extension. The purpose of this curve is so that the robot doesn't have any sharp parts to get stuck in the wall. We originally added the ultrasonic sensor in front and had it scan, but we decided to just mount it to the left side. The purpose of this is so that the ultrasonic sensor can scan if there is a wall to the left. If the distance is too big, then the robot knows it has to turn back to keep following the wall. We also added the protector to the outside. This is made to help smoothen the turns when the robot has to turn 180 degrees. The protector helps us because there is no vulnerable spot for the robot to get stuck by a wall. When the ultrasonic sensor no longer senses a wall to the left, it will make a 180. Since the ultrasonic sensor is between the two tires, it will only sense this once only one tire has passed the wall. Due to our robot always going slightly left, without the protector the robot would turn left and get caught between the extension and the wheels if there was no protector.

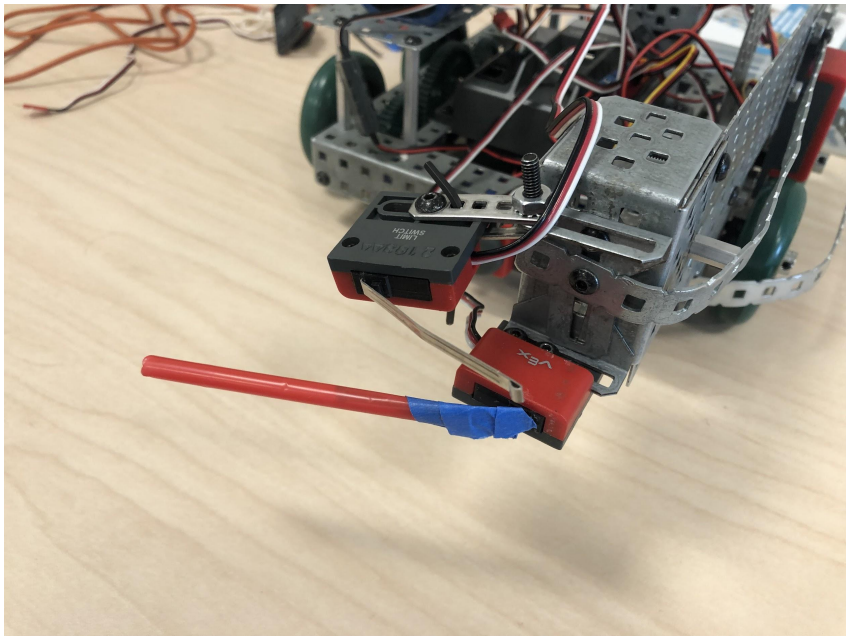


Figure 2

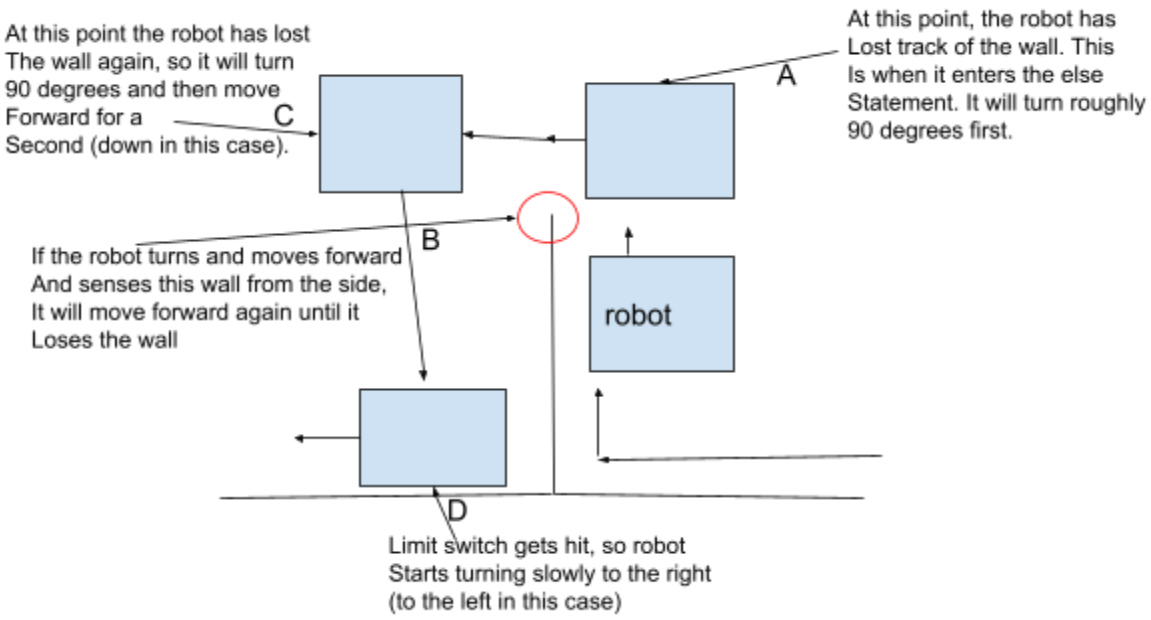
Figure 2 shows a closer view of the end of the extension. The front of the extension basically acts as the front of the robot, so we had to add 2 limit switches. We originally had a bump sensor, but we found that this was unreliable in sensing the bumps. We added the first limit switch (with the blue tape) by screwing it in horizontally attached to a metal piece. This metal piece was screwed into the cube shaped metal piece which was attached to the extension. We also found that sometimes the limit switch wasn't reliable, so we added a second one that goes the opposite way. We found this to be completely effective. These are all of the physical changes we made to our robot. Had our motors not been so different in power we probably would've only added a bump sensor in front and the ultrasonic sensor to the side, but our changes helped us a lot more than our original thoughts.

Software Design

```
WHILE(True)
        IF(limit switches are hit):
                Reverse
                Turn slightly right
        IF(ultrasonic sensor < 150mm):
                IF(limit switches are hit):
                        Reverse
```

```
                Turn slightly right
ELSE:
        Turn left 1 second
        WHILE(ultrasonic sensor > 150mm)
                IF(limit switches are hit):
                        Reverse
                        Turn slightly right
                        BREAK
                Move Forward 1 second
Move Forward
```

Our software was designed so that we are in an infinite while loop so the robot is always doing something. It starts by getting the limit switch values and checking if they are equal to 1. If they are, the robot will turn 90 degrees (since it ran into a corner). It then checks if the robot is close to a wall. If it is, it checks the limit switches again. This is since the robot needs to turn 90 degrees. Due to our physical design, the robot often gets stuck when turning so each turn is a different amount. So instead of always making it turn 90 degrees, it turns a little each time. If the robot at this point is not close to the wall, it enters the else statement. (This happens when a robot needs to turn 90 or 180 degrees to keep following the wall. This is because it's not next to the wall anymore.) When it enters the while loop, it immediately turns for a second. It then checks if the robot is away from the wall. If it is, we check first if the limit switches are hit. We then move the robot forward 1 second. We keep doing this until the robot is close to the wall. Based on how our code is, the robot will exit the while loop when it sees the side of a wall. It will then need to run the main while loop again and go back to the else statement once it loses the side of the wall and keep doing this until it has found the wall again and is lined up.



Performance Evaluation

Our robot went through many revisions in order to be able to go through the maze. We started out by just adding a bump sensor to the front of the robot. We also put a limit switch on both sides of the robot so that it can feel if the wall is to the side of it. We did some thinking and figured that the limit switches on the sides might not be the best solution, so we mounted an ultrasonic sensor on a servo so that it can rotate similar to the light sensor in the previous assignment. Our approach was to have the ultrasonic sensor turn to whatever direction the wall is at. So if the wall was to the left, our robot would move forward and keep sensing the wall. It would only stop when the bump sensor in front got hit or the ultrasonic sensor no longer sensed the wall to the left. We spent a lot of time trying to make the turns exactly 90 and 180 degrees. We then ran into the problem of our motors always being inconsistent as far as speed. So we couldn't even get to the turns because our robot would always turn into a wall when it should be

going straight. At this point, we actually decided to do a different approach where we just scan for open space and travel to the most open space, while using the light sensor to scan for light. We worked on this approach for a couple of classes, but we didn't find any consistency. We also ran into problems with the sensing such as when the wall was on an angle from the robot. It would give a really big value so the robot would sometimes think the wall was the most open space. We decided that we needed to go back to our original approach. We knew we wouldn't have this problem if we could always keep the robot parallel to the wall when moving forward. We decided at this point that our software logic and code was pretty good, but our robot needed physical changes to work. We needed our robot to travel straight consistently along the wall. The best solution for this was to add the extension, which would help guide the robot (as shown in figure 1). This alone was not enough as some walls had small openings in them. That made us add the shield which is a curved metal piece that prevented the robot from getting stuck in these openings. Now because we added these, if the robot ever traveled straight and the wall stopped (as shown in figure 3 below), it would already be leaning slightly left and get stuck in the wall on the turn.

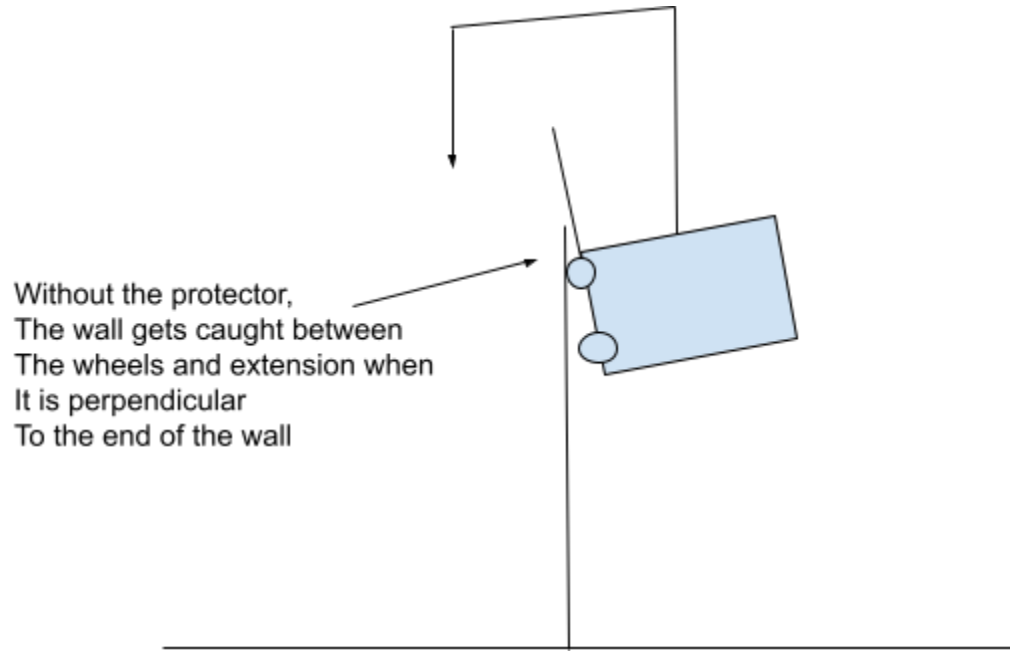
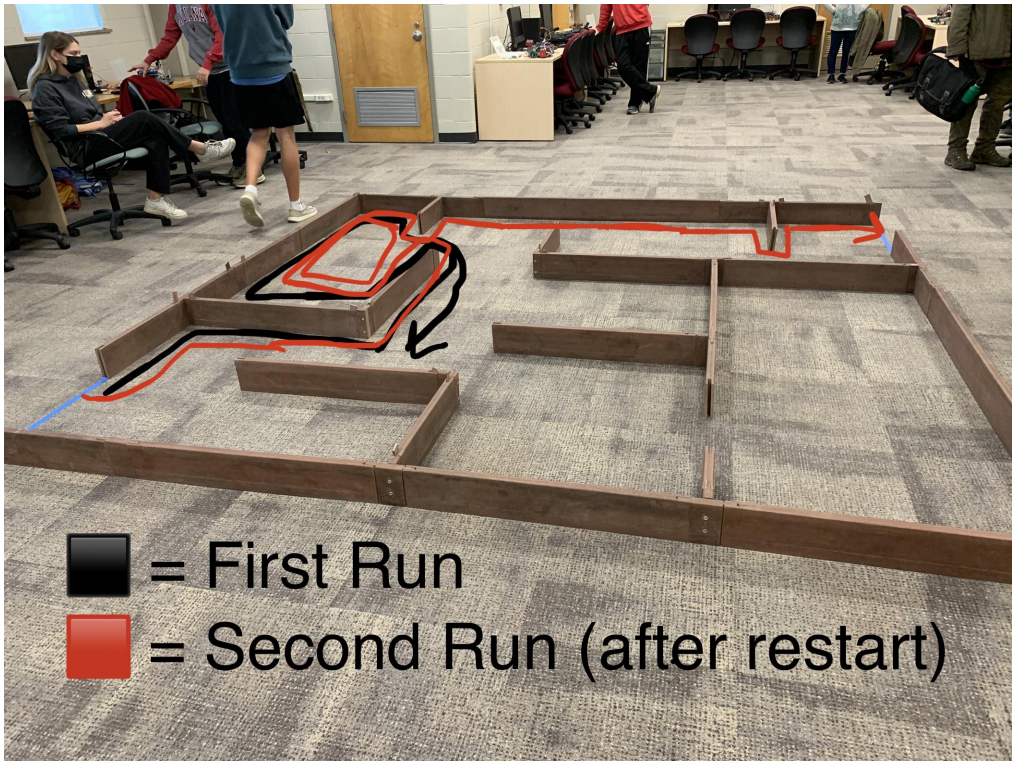


Figure 3

The protector was just made to have a buffer. Since we added these main three pieces, we removed the bump sensor because it would no longer get hit. We added one limit switch to the end of the extension. We found that sometimes the limit switch wouldn't get hit even if the robot was in a corner because sometimes the robot was at a slight angle. We then added another limit switch to the same position but had it go the opposite way. This solved our problem. From this point on, we made some adjustments to the code so that the robot turns slowly and correctly. One problem that came up was that we couldn't easily turn the robot the same amount each time. This was because the robot would either enter the wall at a different angle each time, or when it turned the protector would rub across the wall to the left and prevent it from turning the amount we want. We didn't want to make any more physical changes, so we just changed our software so that the robot turns in small increments each time until it is parallel with the wall it needs to follow.



The performance of our robot in the competition actually went worse than expected. We finished our final product the Friday before the competition, and tested it multiple times going both ways and each time was successful. On our practice trial, we had the robot go along a path that was much longer than the one on test day, and this took only 3 minutes with only 30 bumps. When it came to competition day, our robot started out well. It took the path in black in the picture above. The robot went to the small wall and the top and tried going around it. It ended up completely ignoring the side of the wall for some reason and turned right instead of going forward and then left. It ended up going the wrong way and started going back towards the front. We didn't expect this to happen, and we picked the robot up and stopped the run because the robot was heading away from the light. We placed the robot at the front and started our second run. It moved smoothly along the track and got to the same point as last time. Our robot turned when it saw the small wall at the top left. Instead of making a 180, it went forward and got caught on the wall at the middle left. The reason this happened is because our robot turns inconsistently, so sometimes it turns too far from the wall and ends up losing it. We let the robot keep running because we knew it would make its way back. It traveled along the walls and made it back to the same point. This time, it turned nearly 90 degrees when it got to the small wall in the top left. It went down, then made a 180 like we expected. After this, the robot traveled along the red path at the top of the maze. It ran into the wall at the top right, turned 90 degrees, then made a 180 and made it to the light. Our robot made it through the maze in 4 minutes 39 seconds. It took 64 bumps, but it made it all the way through which was the goal. We definitely could've made improvements to our robot such as making the turns more precise, but our main goal was just to get through the maze. We didn't care if it was pretty or not. One thing that could've helped us on our design was adjusting the protector on the side. Whenever our robot runs into a wall, we have it back up first before turning. We found that a lot of times the protector would be caught on the wall to the left when trying to turn. So instead of being able to make a clean turn each

time, the robot would have to reverse and sometimes turn 3-4 times because it would keep getting stuck.

Conclusion

Overall, the performance of our robot went very well. On the day of the competition, we were one of the three groups that got our robot all the way to the end of the maze. We came in 2nd place as far as timing, but we had 64 bumps which was a lot compared to most groups. We were happy with the performance of our robot since we only restarted it one time during the competition. One thing we would do to improve this is putting more time into our software design. We were just focused on getting our robot to go all the way through the maze and not that worried about time. So, we would change the robot to turn more each time it runs into a wall instead of needing to turn multiple times and hitting the wall multiple times. We would also put more time into trying to figure out how to make the robot go straight consistently. If we were able to compensate for the different motor speeds, then we wouldn't need the extension to help the robot from crashing into the wall. This would make our robot lighter so it would also be able to travel quicker and move through the maze in less time. We could also incorporate the light sensor into our design. If the robot were to ever get stuck or go the wrong way, we could use it to periodically scan for light and go in that direction.

Appendix

```
#pragma config(Sensor, dgtl3,  sonarSensor,    sensorSONAR_mm)
#pragma config(Sensor, dgtl6,  bumpsensor,     sensorTouch)
#pragma config(Sensor, dgtl7,  bumpsensor2,    sensorTouch)
#pragma config(Sensor, dgtl8,  test,           sensorTouch)
#pragma config(Sensor, dgtl9,  one,            sensorTouch)
#pragma config(Sensor, dgtl10, two,            sensorTouch)
#pragma config(Sensor, dgtl11, three,          sensorTouch)
#pragma config(Motor,  port2,           leftMotor,     tmotorServoContinuousRotation, openLoop)
#pragma config(Motor,  port3,           rightMotor,    tmotorServoContinuousRotation, openLoop)
#pragma config(Motor,  port4,           ultServo,      tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//


//motor values are negative for forward
//and positive for backwards because
//we reversed our robot
task main(){
    motor[rightMotor] = -50; //move forward (different speeds to compensate
    motor[leftMotor] = -45;  //for motors with different powers)
    wait1Msec(200);
    bool b = true;

    while(b){ //infinite while loop
        int bumpSensor = SensorValue[bumpsensor];  //gets bump sensor values (limit switches)
        int bumpSensor2 = SensorValue[bumpsensor2];
        if (bumpSensor == 1 || bumpSensor2==1){ //if either bump sensor is hit
            motor[rightMotor] = 0; //stop
            motor[leftMotor] = 0;
            wait1Msec(250);
            motor[rightMotor] = 30; //reverse
            motor[leftMotor] = 30;
            wait1Msec(500);
            motor[rightMotor] = 0; //rotate to the right
            motor[leftMotor] = -40;
            wait1Msec(1400);
            motor[rightMotor] = 0; //pause for a moment
            motor[leftMotor] = 0;
            wait1Msec(250);
            }

        int ultSensor = SensorValue(sonarSensor); //get ultrasonic sensor value


        if (ultSensor > 150 || ultSensor == -1){ //check if robot is far away from wall
            motor[rightMotor] = 0; //pause for a moment
            motor[leftMotor] = 0;
            wait1Msec(500);
            if (bumpSensor == 1 || bumpSensor2==1){ //if bump sensor is hit
                motor[rightMotor] = 0;  //pause for a moment
                    motor[leftMotor] = 0;
                    wait1Msec(250);
                    motor[rightMotor] = 30; //reverse
                    motor[leftMotor] = 30;
                    wait1Msec(500);
                    motor[rightMotor] = 0; //rotate to the right
                    motor[leftMotor] = -40;
                    wait1Msec(1400);
                    motor[rightMotor] = 0; //pause for a moment
                    motor[leftMotor] = 0;
                wait1Msec(250);
              }
```

```
            else{ //if bump sensor is not hit
                motor[rightMotor] = -40; //turn left
                motor[leftMotor] = 30;
                wait1Msec(1200);
                while(ultSensor > 150 || ultSensor == -1){ //when robot is far away from wall
                    ultSensor = SensorValue(sonarSensor);
                    bumpSensor = SensorValue[bumpsensor];
                    bumpSensor2 = SensorValue[bumpsensor2];
                    if (bumpSensor == 1 || bumpSensor2==1){ //if bump sensor is hit, stop
                        motor[rightMotor] = 0;
                        motor[leftMotor] = 0;
                        wait1Msec(250);
                        motor[rightMotor] = 30; //reverse
                        motor[leftMotor] = 30;
                        wait1Msec(500);
                        motor[rightMotor] = 0; //rotate to the right
                        motor[leftMotor] = -40;
                        wait1Msec(1400);
                        motor[rightMotor] = 0;
                        motor[leftMotor] = 0;
                        wait1Msec(250);
                        break;
                        }
                    motor[rightMotor] = -40; //move forward for a second
                    motor[leftMotor] = -35;
                    wait1Msec(1);
                }
            }
            motor[rightMotor] = 0; //pause for a moment
            motor[leftMotor] = 0;
            wait1Msec(500);
        }
        motor[rightMotor] = -50; //move forward for .2 seconds
        motor[leftMotor] = -45;
        wait1Msec(200);

    }
}
```