Maze Traversal

I. Intro

This assignment was four weeks long, and contained our most challenging proposition yet. We had to build a robot that was capable of traversing a random maze with only the tools in our Vex kit. We were allowed to use SquareBot or build a new robot. The robot had to complete the maze within seven minutes and once started at the beginning of the maze, it could not be touched until it either finished the maze or got stuck. In the event that it got stuck or messed up, it could be removed from the maze and replaced at the beginning of the maze while the time continued to run.

II. Mechanical Design

Our final mechanical design was rather simple. The main robot maintained the original design of SquareBot. The chassis was made up of six rails. Two for the inside of the robot, two for the outside of the robot, and two for the front and back to hold the inner and outer rails together. Upon the outer rails, we mounted a raised metal bracket into which we attached the 3000 mAh battery. Four screws and washers were used to secure the battery into the middle of the metal bracket along with a rubber band and plastic tie. Below and to the front of the battery, we installed the Vex microcontroller, into which we attached all of the sensors, motors, and battery of the robot. Beneath the robot we kept the original motors of SquareBot attached to three gears each, which power the four wheels. On the left side of the robot, we attached a raised sonar sensor to the outer rail. Beneath this sensor I attached a metal disc, which I will discuss below. On the front of the robot, two bump sensors cover the front of the robot.

The bump sensors on the front of the robot were imperative for this assignment. We started with one bump sensor in the middle of the front railing, but this design was flawed for one main reason, it was unable to deal with any impacts made at an angle to the front railing. Thus, we decided to move this first bump sensor to the left and add a second bump sensor. This was a step in the right direction, and allowed the robot to register almost all contact to its front rail. However, even this design had a flaw. When confronted with a corner, the two bumpers could remain untriggered. The corner would simply penetrate between the robot’s bumpers.

We found an elegant solution to this problem. Instead of using the button-based bump sensors, we switched to angled bump sensors. These sensors were capable of not only covering the entire front of the robot, but also adding addition length to the sensing capabilities of the front of the robot. In order to cover the entirety of the robots front rail, we attached straws to the bumpers.

We started tackling any issues we had with the robot hitting the walls to its sides by adding angled bump sensors to each side, that when triggered, would cause the robot to backup and turn slightly to correct its path. These got bent, however, and I was constantly afraid for their safety. In addition, they needed to cover the sides of the robots without impeding the capabilities of the front bumpers. Thus, they needed to be installed at an angle to the side bumpers. This meant that even with lots of tape, the side bumpers were loose, and prone to getting caught behind the wheels.

Our solution to the side bumper issue was to remove the offending bumpers, and add metal discs to those corners. These discs were able to take impact from walls to the side of the robot, and yet maintain the robots forward direction. These discs were the perfect solution apart from the fact that they could fall back onto the wheels and stop to robot if hit too hard from the front. This led to our undoing in the final demonstration.

After the final demonstration, we added brackets to hold the metal discs that could be installed with two screws rather the one. This method proved to be full proof.

III. Algorithm

Though our code was very messy throughout the four weeks we were assigned to the project, the final code was very simple. After all of the motor and sensor imports and the initialization of our main task, we defined two integer values labelled left and right. These values corresponded to the speed at which each motor should run when moving forward. The left value was higher than the right value so as to offset the slight tilt to the left that we experienced while moving forward. We then created two while loops. The first ran indefinitely as a while(true) loop. This allowed our robot to continually solve the maze. The second while loop only ran if neither of the sensors was triggered and the value of the left-mounted sonar sensor was less than 500 units. Inside this loop, we let the motors run continually at their predefined ‘left’ and ‘right’ values.

Outside this inner while loop, we created an if-else statement that checked if the bump sensors had been triggered. In the event that they had been triggered, the robot backed up and then turned right. Else, the robot turned left and moved forward.

Our reasoning behind these statements were as follows:

1. Our robot will have to run continuously through the maze.
2. If the robot is adjacent to a left wall of the maze and it has not hit the next wall in front of it, it should move forward.
3. The robot should move forward as fast as possible, but at a safe speed so that the bumper collision did not break the robot.
4. If the bump sensors were triggered, then the robot must have been following the left wall and run into another wall, and must then turn right to avoid running into the left wall.
5. If the bumpers did not hit anything, then the robot must be at the end of the left wall, and must turn left to continue following the wall

We used carefully tested wait times on all the movements to make sure that the robot turned exactly 90 degrees in each direction, and had enough room after backing up or moving forward to complete its next motion.

IV. Performance Evaluation

We faced more issues during this lab than in any other lab so far. This was to be expected, as the lab was far more challenging, and allotted four weeks to complete the given task. Our original robot design was very straightforward. It had a bump sensor on the front, a servo motor, and a sonar sensor attached to said servo. The robot would scan its front and left, and report the distances in each direction. If the left wall was close, as was the front wall, then the robot made a right turn. If only the front wall was close, the robot made a left turn. This design had numerous problems. The first of which was its speed. The robot had to stop in order to scan its surroundings. This meant that it leapt forward at a very slow pace. In addition, the robot required additional code to have it check the most recent scan values and decide which direction to turn in the event of a bump during forward motion. The robot was also unable to handle running into walls at an angle and running into corners of the maze. One of the smaller problems was that the robot’s sensor could not see over the top of the maze, and thus required that we remove a lot of the securing washers in order to reduce its distance the base of the servo motor. This meant that the sonar sensor would fall off if the robot was ever turned upside down while being removed from the maze. This also made the sonar sensor wobbly, and occasionally affected our readings. We replaced this model on the last week with our final wall crawling robot. I worked on the robot alone for 12 hours on the last two days in order to properly implement the wall crawler.

The first step was removing the servo motor and installing the sonar sensor into a dedicated left-facing position. This way, the sonar could continuously scan the robots distance from the wall, and respond in real-time. As mentioned above, I also installed side bumpers and front-facing angled bumpers to deal with edge and corner issues.

Before the final run, my robot performed admirably well. After tightening the metal discs that would occasionally fall back onto the wheels, the robot completed the maze five times! The average run time was two minutes and five seconds with twenty-one bumps. I was very confident that our robot would be able to easily traverse the maze. However, I was mistaken

The robot only made its way through approximately 85-90% of the maze. This was thanks to the metal discs coming loose and hitting the wheels repeatedly. Our robot was not affected by the small change to the end of the maze. Thus, I know that our algorithm was not at fault. I had to restart the robot multiple times during the final run, and make a quick pit stop to fix the metal discs. Though the result was disappointing, I was able to fix my last issues by adding a bracket with more screws, as mentioned above. The robot was then able to complete the maze.

V. Conclusion

This lab was very stressful and probably far more indicative of the challenges that professional computer scientists in the world of robotics face. I can confidently say that I learned a great deal from this assignment. Though my robot was unable to complete the maze during its final run, its software design was very efficient and worked properly. I believe that in the future, I will be far more driven to designing software than hardware. There were simply too many tiny issues with the construction of the robot, and I was not given tools that I believed would allow me to build the perfect robot for my hardware vision. In addition, if I had been able to solder the metal discs onto the ends of the robot, I would have never had any issues with their motion.

My only request for future labs will be that my partners pull more weight than they did in this lab. I had to design all of the hardware and software in the last two days in order to complete the maze. This was in addition to my other coursework, which is as strenuous as any computer science student taking multiple computer science courses.

Overall, I give this lab the best of reviews. I learned more about robotics than in any other lab, and while it was stressful, I had a lot of fun. I can’t wait to work on more robotics after the break.

// imports

#pragma config(Sensor, dgtl1, bump, sensorTouch)

#pragma config(Sensor, dgtl4, bump2, sensorTouch)

#pragma config(Sensor, dgtl7, sonarSensor, sensorSONAR\_mm)

#pragma config(Motor, port1, rightMotor, tmotorVex393, openLoop)

#pragma config(Motor, port10, leftMotor, tmotorVex393, openLoop)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

task main() // main task

{

int left = 65; // slightly higher value for the left motors to counteract the tilting issue

int right = 60; // right motor value

while (true) { // while loop to keep the robot continually running

wait1Msec(300); // waits are always important to give the robot time to think

while (SensorValue[bump] == 0 && SensorValue[bump2] == 0 && SensorValue[sonarSensor] < 500) { // while following the left wall, go straight with the left and right vars

motor[leftMotor] = left;

motor[rightMotor] = right;

}

if (SensorValue[bump] == 1 || SensorValue[bump2] == 1) { // if the bump sensors are

motor[leftMotor] = 0; // activated

motor[rightMotor] = 0;

wait1Msec(300); // stop

motor[leftMotor] = -127;

motor[rightMotor] = -127;

wait1Msec(195); // backup

motor[leftMotor] = 0;

motor[rightMotor] = 0;

wait1Msec(300); // stop again

motor[leftMotor] = 127;

motor[rightMotor] = 0;

wait1Msec(730); // turn right. 730 milliseconds is the exact time for a right turn

motor[leftMotor] = 0; // on our configuration

motor[rightMotor] = 0;

wait1Msec(300); // stop before lopping the while

}

else { // lost the left wall

motor[leftMotor] = 0;

motor[rightMotor] = 0;

wait1Msec(300); // stop

motor[leftMotor] = 0;

motor[rightMotor] = 127;

wait1Msec(715); // turn left. 715 is the exact value for our configuration

motor[leftMotor] = 0;

motor[rightMotor] = 0;

wait1Msec(300); // stop once more

motor[leftMotor] = left;

motor[rightMotor] = right;

wait1Msec(250); // drive forward so as to avoid any corners

motor[leftMotor] = 0;

motor[rightMotor] = 0;

wait1Msec(300); // stop before looping

}

}

}