Arthur Hernandez
Tuesday October 19th
Autonomous Robotics

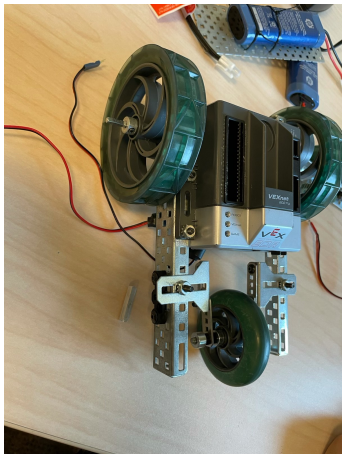
Assignment 4: Maze Traversal Report

Introduction

All autonomous robots rely on sensors to interpret their environment. With the multitude of possible sensors at our disposal, it is important to use the correct type, the correct amount, and to use sensors at the correct time in order to create powerful and useful autonomous robots. When all mechanical issues involved are mitigated and are paired with effective software that utilizes the mechanical design in the most advantageous way, robots can coordinate multiple sensors and effectors to accomplish nontrivial tasks. The goal of this assignment was to create a robot that would be able to traverse a maze from beginning to end within 6 minutes with the least amount of collisions against the maze walls. The maze would be constructed randomly with a light placed at the end. On the last day of the assignment a competition took place where each robot ran through the same maze and was scored by their total time to complete the maze plus a 1 second penalty for each collision with a maze wall. In order to build our robot, my team utilized the Vex CORTEX microcontroller and the RobotC programming environment. Using light, sonar, and bump sensors, our teams main focus was to complete the maze first and to worry about the amount of collisions during the debugging trial and error process. Our mechanical and software designs alternated throughout the weeks we had to create this robot in order to best fit our timeline and will be discussed in the following sections.

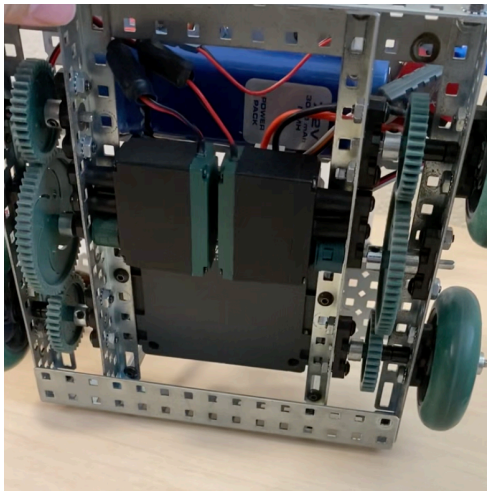
Mechanical design

First Mechanical Design:



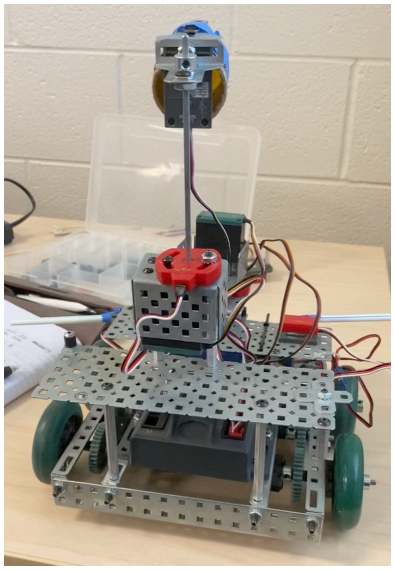
The first step that our team had in this assignment was to decided whether or not to change our robot chassis. I thought it would be best to keep the original Squarbot design and slightly modify it in order to save time and focus on the software design. Between our indecisions we ultimately decided to choose to design a smaller chassis thinking that it would be better at navigating the narrow maze paths. Furthermore, we

also resolved to have a tripod wheel design. This design only used 2 motors, one for each of the front wheels, and one unpowered wheel at the back for stability and to make rotating easier. When initial testing this design our bot would not turn or rotate properly. The bot would instead hop around and be very inconsistent in its movement. One indecision we made in order to fix this issue was to change the wheel sizes to the larger and grippier wheels seen in figure 1. After testing these wheels with multiple different motor speeds, wider chassises, and back wheel placements, we were still not able to fix this issue.

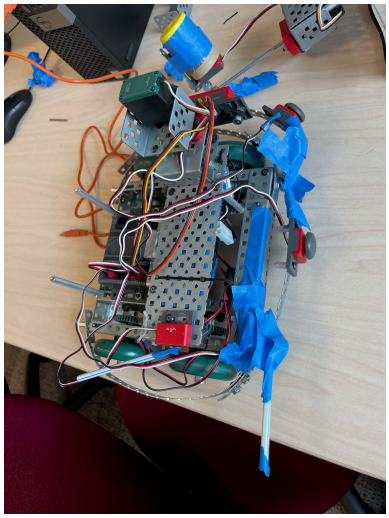


Second Mechanical Design:

Because of these reasons, we abandoned the idea of creating a unique robot design and decided to rebuild Squarebot with the use of 2 motors, one for the left 2 wheels and the other one for the right 2 wheels, each pair connected to their corresponding axle by gears (see figure 2). This design had worked great beforehand and in hindsight should never have been changed. These issues around the chassis resulted in almost a week of lost time that would later degrade our software design. After the chassis was finalized we had to choose how our sensors would be set up based on how we wanted our software to interact with its environment. We decided that our Sonar Sensor would be the most useful when traversing through the maze. Therefore, using our knowledge from previous assignments, we placed our sonar sensor on a servo motor on the front of the robot. This would allow us to sense

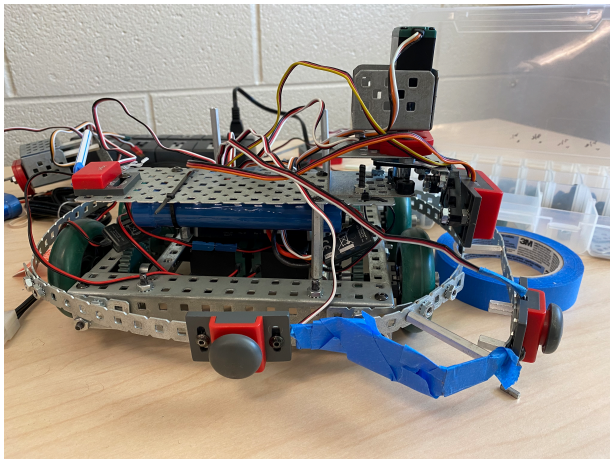


distances from the front of the bot before it was close enough to collide into anything when going forward. We also decided to use a light sensor placed on a servo at the back center of our bot. This decision was made so that it would not interfere with our sonar readings. It was placed in the center so that any light readings could be translated to directional movement accurately in the software design. Two bump switch sensors were placed on the left and right of the bot directly on the chassis so that if the bot were ever to be veering toward a wall this movement could be corrected. This all of these design decisions can bee seen in figure 4. This was not our final design as it proved to be too complex.



Third Mechanical Design:

Because of the poor quality of our software design after the competition day, our team decided it would be best to change our design to be a very simple concept which actually actually be able to pass through a maze; something that could be implemented very quickly, this being the "follow the left wall strategy". Our final design was highly based on trial and error, making changes based on places that the bot would get stuck at when going through the maze. We positioned our sonar sensor on the left side of the bot and used a bumper / shield to protect it from hitting or getting stuck on a wall. A button button sensor was placed on the right corner of the shield helping the robot turn if it ever got stuck on that corner after under-turning around an edge. Another button sensor was placed on the front of the

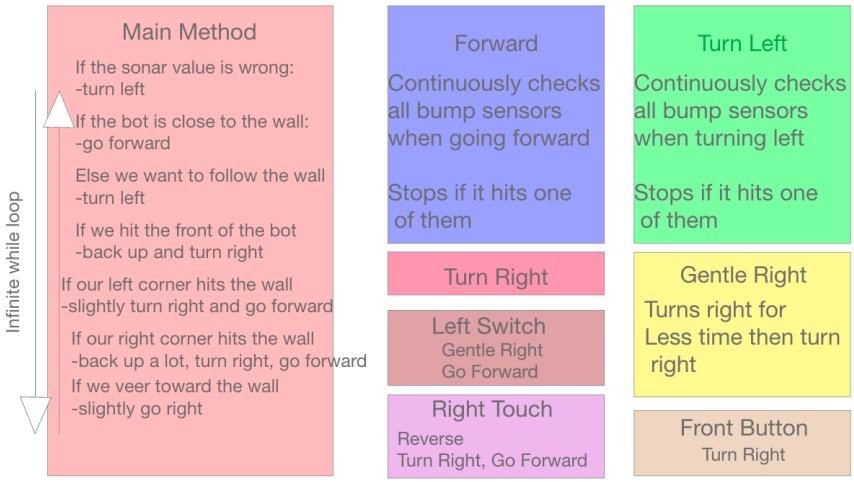


bot so that it would turn away from any wall it hit head on. The last switch sensor was placed at the right corner in case the bot overturned after gong around an edge. Tape was also placed on the edges of the shield and on the edges of buttons to prevent the bot from getting stuck on any maze walls (see figure 5 and 6).

Software design

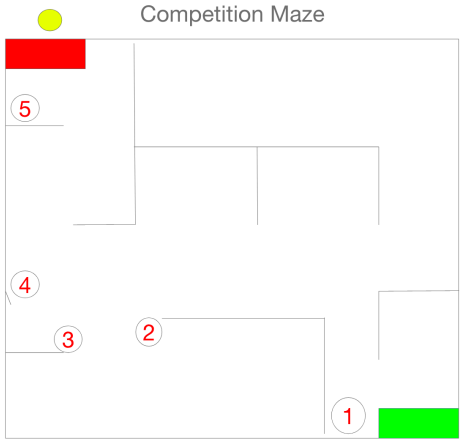
Our software design changed for both the second and third mechanical design iterations, but for both, a while loop was used in the main function to check through different conditions. If those conditions are met, then a separate function would be called. This function would not be iterative but would only be called and ran through once. For the second mechanical design, we tried to create a very accurate bot that would be able to calculate which direction to go forward based around the longest distance path the sonar sensor would detect at either a left, right, or center position (See Appendix DeepLaneScan). This deep lane scan function would set the servo that the sonar sensor was connected to using motor[SonarServo] = 2*I; inside a for loop and saving the largest value it could find in its range to an integer variable instantiated outside the for loop. The servo will then run through the same loop but this time it will compare the largest value found to the current value. If the values are close to similar then the program will save the servo value and evaluate whether it is larger or smaller than 0. If it is significantly larger the bot will decide to move forward and right. If it is smaller it will decided to go forward and left. If it is close enough to 0 it will only move forward. While the bot isn't moving in the DeepLaneScan function it checks for any types of collisions detected from the left and right bump sensors as seen in Appendix MazeBotCompetitionCode. Although we had our light sensor mounted onto the bot and had code written for it, we never had time to implement it into our actual bot. The second software design had a much more straight forward approach. For this design, we again created multiple functions to move the bot. A turn right function that would turn the bot right by reversing the right wheels and going

forward with the left; A turn right gentle function that would turn right as well but for a shorter amount of time; And a reverse function that would reverse both motors of the robot for a very short period. The turn left function was more complex as it kept running until a switch was hit, either the left, front or right bumper sensors were pressed. The go forward function implemented these same switch hits but also measured whether the bot was still close enough to the wall or not. The functionality for the bump sensors can be seen in figure 6. These functions were then iterated through the main function depending on the sonar and bump sensor values.

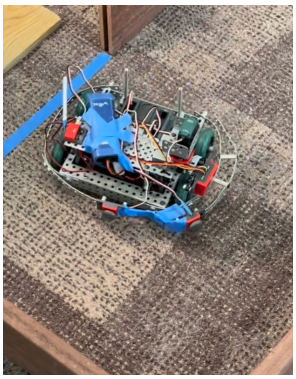


Performance Evaluation

First Design Evaluation: There were many problems in our first software design that we could not figure out in time before the competition day. The main issue that we had was reading values from the sonar sensor in time before it moved again. The other main issue came from the way that our conditional if statements decided to move based on the servo values. This was evident from the way that if our robot decided to turn right it would not find a value larger than that of the left wall and would turn left, getting stuck in an infinite loop of left and right turns. It would also not sense that it was too close to a wall and would run straight into them. This assignment brought



out many weaknesses that my team had when having to take the right decisions. Over-complication set us back in all parts of this assignment. I believe all of our software design issues could have been solved if we kept things simple. Seeing as though our competition robot did not make it past point 1 in figure 7, and had 79 collisions, I would count our completion attempt to be a failure, but this does not mean we did not have successes. We learned a lot of valuable lessons. My team has vowed to work better together and to all be on the same page in order to finish our objectives. Although we did not do well for our competition, we showed perseverance in coming back to class the next day and working together until we were able to finally complete the maze. We continued to test through parts of the maze where our bot would have errors, parts 2,3,4, and 5. In error 2, we added right touch that helped us turn through the corner when over-turning left. In error 3 we added the right switch that helped when under-turning left. And In error 4 we added tape to the left switch so that it would not get jammed against the gap in the wall. In error 5 we ran the bot through the maze around 5 times before being able to get our straw on our switch to cross the finish line (see figure 8).



Appendix 1: DeepLaneScan

```
#pragma config(Sensor, in1,    light,          sensorReflection)
#pragma config(Sensor, dgtl10, sonar,          sensorSONAR_cm)
#pragma config(Motor,  port2,           leftMotor,     tmotorServoContinuousRotation, openLoop,
driveLeft)
#pragma config(Motor,  port3,           rightMotor,    tmotorServoContinuousRotation, openLoop,
driveRight)
#pragma config(Motor,  port4,           lightSensor,   tmotorServoStandard, openLoop)
#pragma config(Motor,  port5,           SonarServo,    tmotorServoStandard, openLoop)

//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
// scan the wall
void servoScan(){

   int i;
   int c = 0;
   // array to hold the sonar data from each point in the sweep, each index says which way that
the sonar was pointing
   // going to need to aim for the highest distance away

   float arr[60];
   for(i = -60; i < 60; i = i+2){
      motor[SonarServo] = 2*i;
      wait1Msec(50);
      arr[c] = SensorValue(sonar);
      c = c + 1;
   }
   writeDebugStream("%d",arr[0]);
}

// find and follow the longest  path found after a few circles
void deepLaneScan(){
   //scan around in a circle 4 times
   int largestValSoFar = 25; //test value
   int i;
   int newValue;

   for(i = 0; i < 36; i++){
         writeDebugStream("HI");
      // circle right storing highest sonar values
      motor[leftMotor] = -55;
      motor[rightMotor] = 53;
      // run for 1 second
      wait10Msec(100);
      motor[leftMotor] = 0;
      motor[rightMotor] = 0;
```

```
        wait1Msec(10);
        newValue = SensorValue(sonar);
        if(newValue > largestValSoFar){
            largestValSoFar = newValue;
        }
    }
    // same thing as the prior loop but react when a high value is found
    for(i = 0; i < 36; i++){
        motor[leftMotor] = -55;
        motor[rightMotor] = 53;
        // run for 1 second
        wait10Msec(1);
        motor[leftMotor] = 0;
        motor[rightMotor] = 0;

        //wait1Msec(10);
        newValue = SensorValue(sonar);
        if(newValue >= largestValSoFar - 5){
            scanLight();
            moveForward();
            scanLight();
            break;
        }
    }
}
task main(){
    deepLaneScan();
}
```

Appendix 2: MazeBotCompetitionCode

```
#pragma config(Sensor, in1,    light,         sensorReflection)
#pragma config(Sensor, dgtl8,  bSwitch1,      sensorTouch)
#pragma config(Sensor, dgtl9,  bSwitch2,      sensorTouch)
#pragma config(Sensor, dgtl10, sonar,         sensorSONAR_cm)
#pragma config(Motor,  port2,           leftMotor,     tmotorServoContinuousRotation, openLoop,
driveLeft)
#pragma config(Motor,  port3,           rightMotor,    tmotorServoContinuousRotation, openLoop,
driveRight)
#pragma config(Motor,  port4,           lightSensor,   tmotorServoStandard, openLoop)
#pragma config(Motor,  port5,           SonarServo,    tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

// initializations
// global distance cutoff for the robot is too close
int cutoff = 25;
int farFront = 55;
// temp Value
int val;
// set the speed for the robot for all code
int lSpeed = 25;
int rSpeed = 27;
int waitTime = 100;
```

```
// function to scan for the light, returns the servo value where the light was the highest
int scanLight(){
    int countVal;
    int endVal = -127;
    int lightVal;
    int highestLight = 4096;
    int highCount;

    motor[lightSensor] = countVal;
    // loop through whole range of motion starting from the right
    for(countVal = 127; countVal > endVal; countVal = countVal - 5){
        // Check current light value
        lightVal = SensorValue(light);
        //Check if new light value is brighter than old brightest light value
        if(lightVal + 5 < highestLight){
            highestLight = lightVal;
            highCount = countVal;
            // update highest light position
            //lightPos = countVal;
        }
        // wait at each position for 300 miliseconds
        wait1Msec(50);
        // move to the new position
        motor[lightSensor] = countVal;
    }
    if(highestLight < 5000){
        return(highCount);
    }
    else{
        return(0);
    }


}
// turn right
void turnRight(){
    motor[leftMotor] = -lSpeed;
    motor[rightMotor] = rSpeed;
    wait1Msec(850);
    motor[leftMotor] = 0;
    motor[rightMotor] = 0;

}
// turn left
void turnLeft(){
    motor[leftMotor] = lSpeed;
    motor[rightMotor] = -rSpeed;
    wait1Msec(850);
    motor[leftMotor] = 0;
    motor[rightMotor] = 0;
```

```
}
// move forward
void goForward(){
    motor[leftMotor] = lSpeed;
    motor[rightMotor] = rSpeed;
    wait1Msec(850);
    motor[leftMotor] = 0;
    motor[rightMotor] = 0;
}

// move backward
void reverse(){
    motor[leftMotor] = -lSpeed;
    motor[rightMotor] = -rSpeed;
    wait1Msec(3*waitTime);
    motor[leftMotor] = 0;
    motor[rightMotor] = 0;
}


// returns if the servo is close to the wall on the left side,0 if its too close to the wall
// ad something to keep it from decreasing the distance to the left wall in processs
// Add something to check more than one position
int servoCheckLeft(){
    motor[SonarServo] = -105;

    val = SensorValue(sonar);
    // check for special case
    if(val == -1){
        reverse();
        servoCheckLeft();
    }

    return(val);
}
// check to the right
int servoCheckRight(){
    wait1Msec(waitTime/2);
    motor[SonarServo] = 110;
    wait1Msec(waitTime/2);
    val = SensorValue(sonar);
    if(val == -1){
        reverse();
        //reccursion should be removed but should not happen
        servoCheckRight();
    }



    return(val);
}

// check front distances
int servoCheckFront(){
```

```
    motor[SonarServo] = 0;
    val = SensorValue(sonar);
    // catch if the servo gives a -1 value
    if(val == -1){
        reverse();
        //reccursion should be removed, new file used instead
        servoCheckFront();
        val = 0;
    }

    return(val);
}

// method to check the front whisker sensors
int senseBumps(){
    if(SensorValue(bSwitch1) == 1){
        turnRight();

        return(1);
    }
    if(SensorValue(bSwitch2) == 1){
        turnLeft();
        //turnLeft();
        //turnLeft();
        return(2);
    }
    return(0);
}

// initialize variables
int lv,rv,fv,lightV;

task main()
{
    while(true){
        // check for bumps
        if(senseBumps() != 0){
            continue;
        }

        wait1Msec(50);
        // check servo either side
        rv = servoCheckRight();

        wait1Msec(50);
        rv = servoCheckRight(); // needs twice for some reason
        wait1Msec(50);
        fv = servoCheckFront();
        wait1Msec(75);

        lv = servoCheckLeft();
        wait1Msec(50);
        if(senseBumps() != 0){
            continue;
```

```
    }

    if(fv > farFront){
      goForward();

      continue;
    }

    else if(lv <= cutoff -3 ){ //turn right and forward if there is a lot of room to the right

      turnRight();
      goForward();
    }
    // turn left and forward if there is a lot of room to the right
    else if(rv <= cutoff){
      turnLeft();
      goForward();
      //continue;

    }
    // go forward if possible
    else if(fv >= cutoff){
      goForward();
      continue;
    }

    // wall all around and turn around
    else if(lv < 30 && rv < 30 && fv < 30){
      turnLeft();
      turnLeft();
      turnLeft();

    }
    else{
      reverse();
    }

  } // end while loop
}// end program
```

Appendix 3: MazeBotTurnLeft

```
#pragma config(Sensor, in1,    light,          sensorReflection)
#pragma config(Sensor, dgtl6,  rightTouch,     sensorTouch)
#pragma config(Sensor, dgtl8,  ,               sensorTouch)
#pragma config(Sensor, dgtl9,  leftSwitch,     sensorTouch)
#pragma config(Sensor, dgtl10, sonar,          sensorSONAR_cm)
#pragma config(Sensor, dgtl12, frontButton,    sensorTouch)
```

```
#pragma config(Motor,  port2,           rightMotor,    tmotorServoContinuousRotation, openLoop,
driveLeft)
#pragma config(Motor,  port3,           leftMotor,     tmotorServoContinuousRotation, openLoop,
driveRight)
#pragma config(Motor,  port5,           servoMotor,    tmotorServoStandard, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

// speed for the motors
int speed = 30;
// distance to stay from the walls
int sonarCutoff = 16;

//reverse backward
void reverse(){
	motor[leftMotor] = -speed;
	motor[rightMotor] = -speed;
	wait1Msec(370);
	motor[leftMotor] = 0;
	motor[rightMotor] = 0;
}


void turnRight(){
	motor[leftMotor] = speed;
	motor[rightMotor] = -speed;
	wait1Msec(1750); // arbitrary to equate to prior: 2750
	motor[leftMotor] = 0;
	motor[rightMotor] = 0;
}

//same as turn right but for much less time
void turnRightGentle(){
	motor[leftMotor] = speed;
	motor[rightMotor] = -speed;
	wait1Msec(235); // arbitrary to equate to prior: 2750
	motor[leftMotor] = 0;
	motor[rightMotor] = 0;
}

void goForward(){
	motor[leftMotor] = speed + 2; // 2 to account for the speed disparity and keep it away
from the wall
	motor[rightMotor] = speed;
	wait1Msec(850);
	if(SensorValue(frontButton) == 1){
		motor[leftMotor] = 0;
		motor[rightMotor] = 0;
		turnRight();
	}
	// leftButton check
	if(SensorValue(leftSwitch) == 1){
		reverse();
			turnRightGentle();
			goForward();
```

```
	}
	if(SensorValue(rightTouch)){
			reverse();
			reverse();
			turnRight();
			goForward();
		}
	motor[leftMotor] = 0;
	motor[rightMotor] = 0;
	if(SensorValue(sonar) <= sonarCutoff){
		motor[leftMotor] = 0;
		motor[rightMotor] = 0;
	}

}

void turnLeft(){
	motor[leftMotor] = 13;
	motor[rightMotor] = speed;

	if(SensorValue(frontButton) == 1){
		motor[leftMotor] = 0;
		motor[rightMotor] = 0;
		turnRight();
	}
	if(SensorValue(leftSwitch) == 1){
		motor[leftMotor] = 0;
		motor[rightMotor] = 0;
		turnRightGentle();
		goForward();
	}
	if(SensorValue(rightTouch)){
			reverse();
			reverse();
			turnRight();
			goForward();
		}
}
task main(){
	// sonar value
	int sv;
	while(true){
		sv = SensorValue(sonar);

		if(sv < 0){
			turnLeft();
		}
		if(sv < sonarCutoff){
			goForward();
			//count = 0;
			//else turn left
		}
		else{
			turnLeft();
```

```
		}
		//we hit the bump sensor turn right
		if(SensorValue(frontButton) == 1){
			// changable
			reverse();
			turnRight();
		}
		if(SensorValue(leftSwitch) == 1){
			turnRightGentle();
			goForward();
		}
		if(SensorValue(rightTouch)){
			reverse();
			reverse();
			turnRight();
			goForward();
		}

		// try to reallign if possible
		sv = SensorValue(sonar);

		if(sv < 6){
			turnRightGentle();
		}
	}// end while loop

}
```