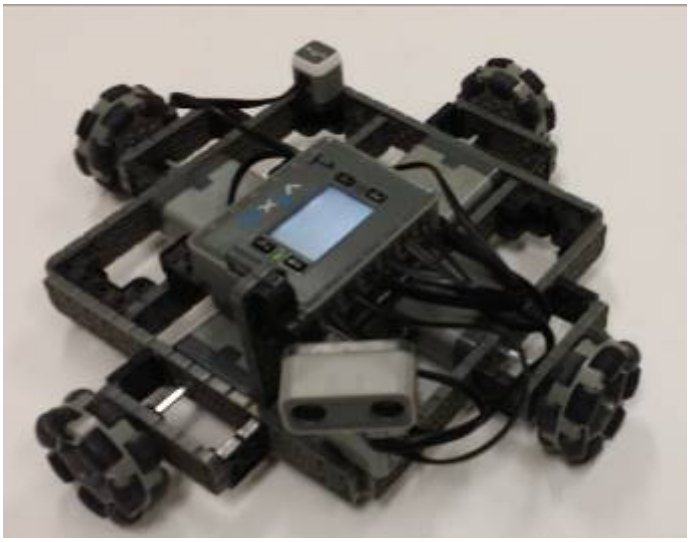
Bot of Avoiding Obstacles

Nikhil Sharma



Introduction

General Description And What It Does:

We are going to build a robot using VEX IQ that will be able to navigate through various environments autonomously. It will have one ultrasonic sensor that will allow it to detect objects and move around it. Eventually, we will put it in a maze and have the robot complete the maze by itself.

Why/Inspiration:

I chose to do obstacle avoidance because it offers multiple paths to get to the solution. Unlike holonomic drive or some of the other check offs, there is no one solution to getting obstacle avoidance. Obstacle avoidance is also a very important new and developing field. Google's cars are going to need highly sophisticated obstacle avoidance algorithm, and doing this project will give me a very basic look into that algorithm.

Primary Programming Goals: This bot must be able to:

1. Take 3 values using the ultrasonic sensor. One value for the front, one value for the left, and one value for the right.

2. Use programming to take those three inputs and make a decision to go the path with the greatest distance

3. Has to be able to navigate a "canyon". Three equidistant objects, one front, one right and left. The robot has to be able to detect all three objects, recognize that it's stuck in a "canyon" and be able to back up.

Primary Programming Challenges:

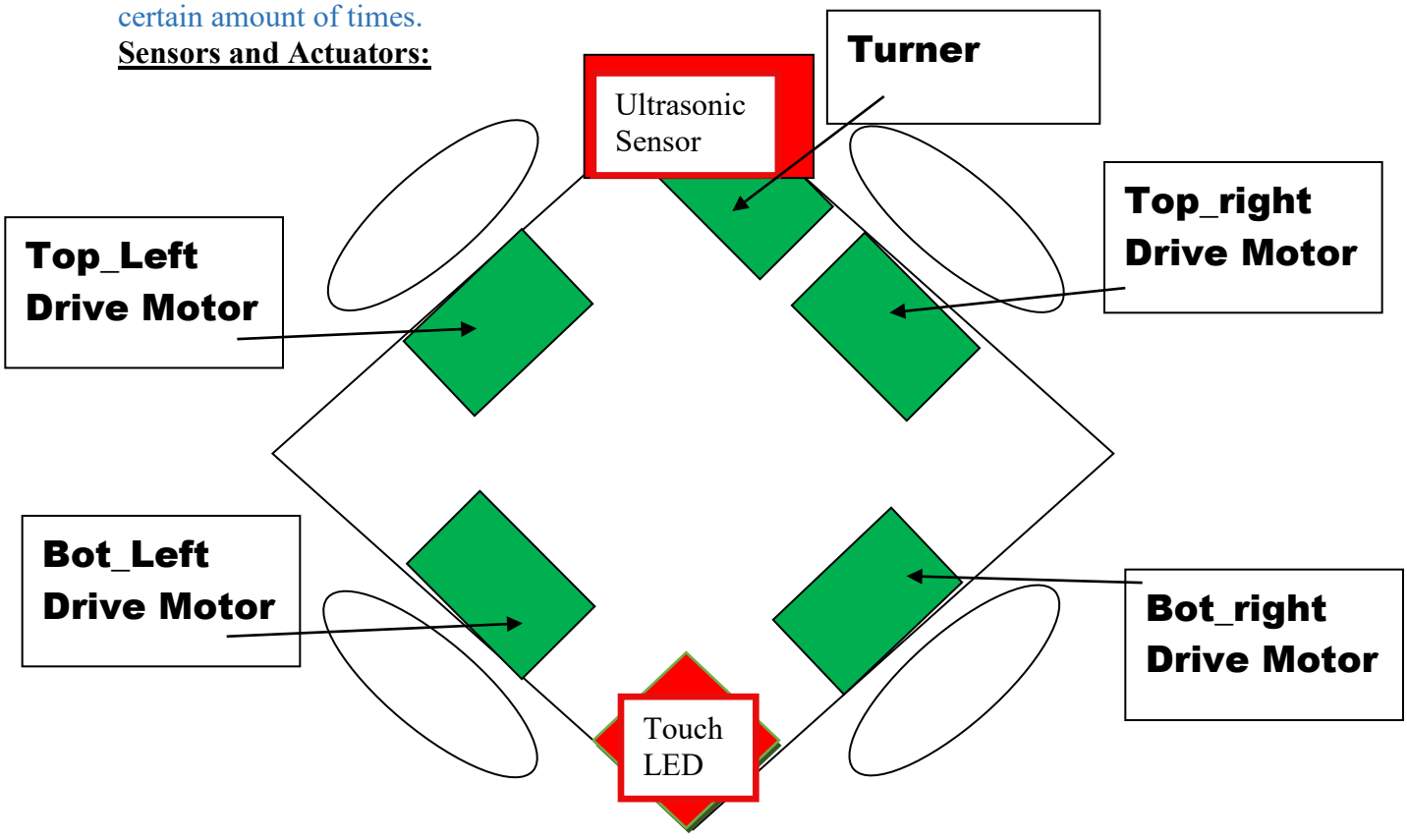
The hardest part with obstacle avoidance will be the logic involved in determining which way the robot will move. We have the ultrasonic sensor set up at the front of the bot, we have the correct orientations of the three positions. The difficulty comes in taking those values and turning them into decisions. This project is hard because there are so many different scenarios in which the code can mess up or become ambiguous. While in our project, this might just result in running into a box, in real life, it might mean a fatal car crash.

Secondary Programming Goals: It would be nice if this bot would:
1. Have the robot continuously checking all 3 sides. Meaning instead of having to jerk its head, it would have a continuously rotating head.

2. Have the robot be able to check distance backwards as well to see if backing up is a viable option.

Secondary Programming Challenges:
The continuously rotating motor shouldn't be a problem, we'll just have it constantly moving between the three positions. The harder part will be checking backwards. The reason this will be harder than the primary challenges is because we will have to put in place a threshold of distance that will determine if the robot will back up or not. The most easily made mistake with backing up will be getting caught in an infinite loop, in which the robot just backs up, then goes forward, then backs up. We will have to put a counter in place that will make the robot stop, and re-assess the situation after it backs up a certain amount of times.

Sensors and Actuators:



Programming Day 1: **10/23/15**

```
#pragma config(Motor,  motor1,top_left,tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor, motor2, bot_left, tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor, motor3, bot_right, tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,top_right, tmotorVexIQ, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard !!*//
task main()
{
	motor[motor1] = 31;
	motor[motor2] = 31;
	motor[motor3] = -31;
	motor[motor4] = -31;
	wait1Msec(1750);
}
// This code makes the robot go forward exactly 1 foot.
```

We had about 30 minutes today to start working on the robot. Since we are modifying a pre-made holonomic driving VEX IQ bot, we figured out what values we needed to set the motors to make the robot to go forward. The most important part was to find the timing to make it go forward exactly 1 foot. I wish I could say there was a complex formula I used to find the answer, but I just used trial and error.

Programming Day 2: **10/27/15**

```
int ahead=0;
int dleft=0;
int dright=0;
int dec= 0;//0=left 1=straight 2=right
void fore(){
 motor[motor1] = 31;
motor[motor2] = 31;
motor[motor3] = -31;
motor[motor4] = -31;
wait1Msec(1750);
// This code makes the robot go forward exactly 1 foot.
}
void left(){
 motor[motor1] = -31;
motor[motor2] = 31;
motor[motor3] = 31;
```

```
motor[motor4] = -31;
wait1Msec(1750);
// This code makes the robot go left exactly 1 foot.
}
void right(){
motor[motor1] = 31;
motor[motor2] = -31;
motor[motor3] = -31;
motor[motor4] = 31;
wait1Msec(1750);
// This code makes the robot go right exactly 1 foot.

}
void back(){
 motor[motor1] = -31;
motor[motor2] = -31;
motor[motor3] = 31;
motor[motor4] = 31;
wait1Msec(1750);
// This code makes the robot go backwards exactly 1 foot.
}
void stopbot(){
  motor[motor1] =0;
motor[motor2] = 0;
motor[motor3] = 0;
motor[motor4] = 0;
wait1Msec(1);
 //this code pauses for 1 second

}
void check(){
  while(getMotorEncoder(motor5)<45){
 motor[motor5]=10;
}
dleft=SensorValue[port6];
  while(getMotorEncoder(motor5)>-45){
 motor[motor5]=-10;
}
ahead=SensorValue[port6];
  while(getMotorEncoder(motor5)>-135){
 motor[motor5]=-10;
}
dright=SensorValue[port6];
  while(getMotorEncoder(motor5)<-45){
```

```
 motor[motor5]=10;
}
}
void init(){
 resetMotorEncoder(motor5);
 while(getMotorEncoder(motor5)>-45){
 motor[motor5]=-10;
}
}
```

Today, we took the code from yesterday which made it move forward one foot, and adapted it to move in all 4 directions. Since we are modifying our holonomic drive VEX IQ robot to use with obstacle avoidance, we had to figure out which motors to reverse to make the robot move in the specific direction. After we got the code to make the robot move 1 foot forwards, backwards, to the right, and to the left, we started working with our Motor 5, which controls the ultrasonic sensor. We first had to establish an initial reference point, where the ultrasonic would automatically position itself at the start of the code execution. After we created the Init method which did this task, we started working on a Check method that would position the ultrasonic sensor at three points to get the forward, left, and right readings. We did this by using the encoder values from the motors, since the VEX IQ motors are smart motors.

Programming Day 3: **10/29/15**

```
#pragma config(Sensor, port6,  sonar,        sensorVexIQ_Distance)
#pragma config(Sensor, port7,  color,        sensorVexIQ_LED)
#pragma config(Motor,  motor1,        top_left,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor2,        bot_left,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor3,        bot_right,    tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,        top_right,    tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor5,        turner,       tmotorVexIQ, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
int ahead=0;
int dleft=0;
int dright=0;
int dec= 0;
//0=left 1=straight 2=right
int diff =0;
void init(){
 resetMotorEncoder(motor5);
 while(getMotorEncoder(motor5)>-45){
  motor[motor5]=-10;
```

```
 }
}

void stopbot(){
 motor[motor1] =0;
 motor[motor2] = 0;
 motor[motor3] = 0;
 motor[motor4] = 0;
 wait1Msec(1);
 //this code pauses for 1 second
}
void check(){
 while(getMotorEncoder(motor5)<45){
  motor[motor5]=10;
 }
 dleft=SensorValue[port6];
 while(getMotorEncoder(motor5)>-45){
  motor[motor5]=-10;
 }
 ahead=SensorValue[port6];
 while(getMotorEncoder(motor5)>-135){
  motor[motor5]=-10;
 }
 dright=SensorValue[port6];
 while(getMotorEncoder(motor5)<-45){
  motor[motor5]=10;
 }
 displayTextLine(3, "ahead V is: %d", ahead);
  displayTextLine(3, "leftV is: %d",dleft);
   displayTextLine(3, "right Value is: %d", dright);
}

void fore(){
 motor[motor1] = 31;
 motor[motor2] = 31;
 motor[motor3] = -31;
 motor[motor4] = -31;
 wait1Msec(1750);
 dec=0;
 // This code makes the robot go forward exactly 1 foot.
}
void back(){

 motor[motor1] = -31;
```

```
 motor[motor2] = -31;
 motor[motor3] = 31;
 motor[motor4] = 31;
 wait1Msec(1750);
 stopbot();
 check();
 diff= dleft-dright;
 if(diff>-250&&diff<250)
 {
  setTouchLEDColor(color, colorRed);
  back();
 }
 else{
  dec=0
 }
 // This code makes the robot go backwards exactly 1 foot.
}
void left(){
 motor[motor1] = -31;
 motor[motor2] = 31;
 motor[motor3] = 31;
 motor[motor4] = -31;
  wait1Msec(1750);
 /*if(dec=2){
   setTouchLEDColor(color, colorBlue);
 back();
 }
 wait1Msec(1750);
  dec = 1;*/
 // This code makes the robot go left exactly 1 foot.
}
void right(){
 motor[motor1] = 31;
 motor[motor2] = -31;
 motor[motor3] = -31;
 motor[motor4] = 31;
 wait1Msec(1750);
 /*if(dec=1){
    setTouchLEDColor(color, colorBlue);
  back();
 }
  dec = 2;*/

 // This code makes the robot go right exactly 1 foot.
```

```
}

void move(){
 if (ahead>250){
  fore();
 }
 else if(dleft>ahead)
 {
  if (dleft>dright)
  {
   left();
  }
  else {
   right();
  }
 }
 else if(dright>ahead){
  right();
 }
 else{
  fore();
 }
}
void step(){
 check();
 move();
 stopbot();
 setTouchLEDColor(color, colorGreen);
}

task main(){
 init();
 while(true){
  step();
 }
 /* fore();
 left();
 back();
 right();
 */
}
```

The next step in our code was to try to successfully navigate a maze using our Check method. We created a Step method that checked the readings from the three directions,

chose one direction, moved a foot that direction, and then stopped moving. We did two things to make debugging easier for us. We use the print to screen method to show us what values the robot is receiving, except we used it wrong at first. We were having all three strings display on the same line of the screen, which confused the robot. We also incorporated a Touch LED that turned a certain color to indicate which direction it was going.

Programming Day 4: **11/2/15**

```
#pragma config(Sensor, port6,  sonar,        sensorVexIQ_Distance)
#pragma config(Sensor, port7,  color,        sensorVexIQ_LED)
#pragma config(Motor,  motor1,        top_left,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor2,        bot_left,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor3,        bot_right,    tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,        top_right,    tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor5,        turner,       tmotorVexIQ, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
int ahead=0;
int dleft=0;
int dright=0;
int dec= 0;//0=left 1=straight 2=right
int rcan=0;
int lcan=0;
void init(){
 resetMotorEncoder(motor5);
 while(getMotorEncoder(motor5)>-45){
  motor[motor5]=-10;
 }
}

void stopbot(){
 motor[motor1] =0;
 motor[motor2] = 0;
 motor[motor3] = 0;
 motor[motor4] = 0;
 wait1Msec(1);
 //this code pauses for 1 second
}
void check(){
 while(getMotorEncoder(motor5)<45){
  motor[motor5]=10;
 }
```

```
 dleft=SensorValue[port6];
 while(getMotorEncoder(motor5)>-45){
  motor[motor5]=-10;
 }
 ahead=SensorValue[port6];
 while(getMotorEncoder(motor5)>-135){
  motor[motor5]=-10;
 }
 dright=SensorValue[port6];
 while(getMotorEncoder(motor5)<-45){
  motor[motor5]=10;
 }
 displayTextLine(1, "ahead V is: %d", ahead);
 displayTextLine(2, "leftV is: %d",dleft);
 displayTextLine(3, "right Value is: %d", dright);
}

void fore(){
 motor[motor1] = 31;
 motor[motor2] = 31;
 motor[motor3] = -31;
 motor[motor4] = -31;
 wait1Msec(750);
 dec=0;
 // This code makes the robot go forward exactly 1 foot.
}
void back(){
 motor[motor1] = -31;
 motor[motor2] = -31;
 motor[motor3] = 31;
 motor[motor4] = 31;
 wait1Msec(750);
 stopbot();
 check();
 if(dright<(rcan +250)||dleft<(lcan+250))
 {
  back();
 }
 else{
  dec=0;
 }
 // This code makes the robot go backwards exactly 1 foot.
}
void left(){
```

```
 motor[motor1] = -31;
 motor[motor2] = 31;
 motor[motor3] = 31;
 motor[motor4] = -31;
 wait1Msec(750);
 if(dec==2){
  stopbot();
  rcan=dright;
  lcan=dleft;
  if(lcan==26214)
   lcan=0;
  if(rcan==26214)
   rcan=0;

  back();
 }
 wait1Msec(750);
 dec = 1;
 // This code makes the robot go left exactly 1 foot.
}
void right(){
 motor[motor1] = 31;
 motor[motor2] = -31;
 motor[motor3] = -31;
 motor[motor4] = 31;
 wait1Msec(750);
 if(dec==1){
  stopbot();
  rcan=dright;
  lcan=dleft;
  if(lcan==26214)
   lcan=0;
  if(rcan==26214)
   rcan=0;
  back();
 }
 dec = 2;
 // This code makes the robot go right exactly 1 foot.
}

void move(){
 if (ahead>500){
  fore();
 }
```

```
 else if(dleft>ahead)
 {
  if (dleft>dright)
  {
   left();
  }
  else {
   right();
  }
 }
 else if(dright>ahead){
  right();
 }
 else{
  fore();
 }
}
void step(){
 if(dec==0){
  setTouchLEDColor(color, colorGreen);
 }
 else if(dec==1)
 {  setTouchLEDColor(color, colorBlue);}
 else{
  setTouchLEDColor(color, colorRed);
 }
 check();
 move();
 stopbot();
}

task main(){
 init();
 while(true){
  step();
 }
 /* fore();
 left();
 back();
 right();
 */
}
```

The final step to getting the check off was to get the canyon code working. A canyon situation is where you have your robot stuck in a situation where the all three readings (front, and the two sides) are all equal. To do this, we decided to keep a counter of when the bot moved left and right. When the bot went left, and then right, it took both right and left measurements. The robot then started backing up until the new measurements were larger than the original ones.

Final Program:

```
#pragma config(Sensor, port6,  sonar,          sensorVexIQ_Distance)
#pragma config(Sensor, port7,  color,          sensorVexIQ_LED)
#pragma config(Motor,  motor1,          top_left,      tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor2,          bot_left,      tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor3,          bot_right,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor4,          top_right,     tmotorVexIQ, PIDControl, encoder)
#pragma config(Motor,  motor5,          turner,        tmotorVexIQ, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
int ahead=0;
int dleft=0;
int dright=0;
int dec= 0;//0=left 1=straight 2=right
int rcan=0;
int lcan=0;
void init(){
  resetMotorEncoder(motor5);
  while(getMotorEncoder(motor5)>-45){
    motor[motor5]=-10;
  }
}
void stopbot(){
  motor[motor1] =0;
  motor[motor2] = 0;
  motor[motor3] = 0;
  motor[motor4] = 0;
  wait1Msec(1);
  //this code pauses for 1 second
}
void check(){
  while(getMotorEncoder(motor5)<45){
    motor[motor5]=10;
  }
  dleft=SensorValue[port6];
  while(getMotorEncoder(motor5)>-45){
    motor[motor5]=-10;
  }
  ahead=SensorValue[port6];
  while(getMotorEncoder(motor5)>-135){
    motor[motor5]=-10;
  }
  dright=SensorValue[port6];
  while(getMotorEncoder(motor5)<-45){
    motor[motor5]=10;
  }
  displayTextLine(1, "ahead V is: %d", ahead);
  displayTextLine(2, "leftV is: %d",dleft);
  displayTextLine(3, "right Value is: %d", dright);
}
void fore(){
```

```
void fore(){
  motor[motor1] = 31;
  motor[motor2] = 31;
  motor[motor3] = -31;
  motor[motor4] = -31;
  wait1Msec(750);
  dec=0;
  // This code makes the robot go forward exactly 1 foot.
}
void back(){
  motor[motor1] = -31;
  motor[motor2] = -31;
  motor[motor3] = 31;
  motor[motor4] = 31;
  wait1Msec(750);
  stopbot();
  check();
  if(dright<(rcan +250)||dleft<(lcan+250))
  {
    back();
  }
  else{
    dec=0;
  }
  // This code makes the robot go backwards exactly 1 foot.
}
void left(){
  motor[motor1] = -31;
  motor[motor2] = 31;
  motor[motor3] = 31;
  motor[motor4] = -31;
  wait1Msec(750);
  if(dec==2){
    stopbot();
    rcan=dright;
    lcan=dleft;
    if(lcan==26214)
      lcan=0;
    if(rcan==26214)
      rcan=0;

    back();
  }
  wait1Msec(750);
  dec = 1;
  // This code makes the robot go left exactly 1 foot.
}
void right(){
  motor[motor1] = 31;
```

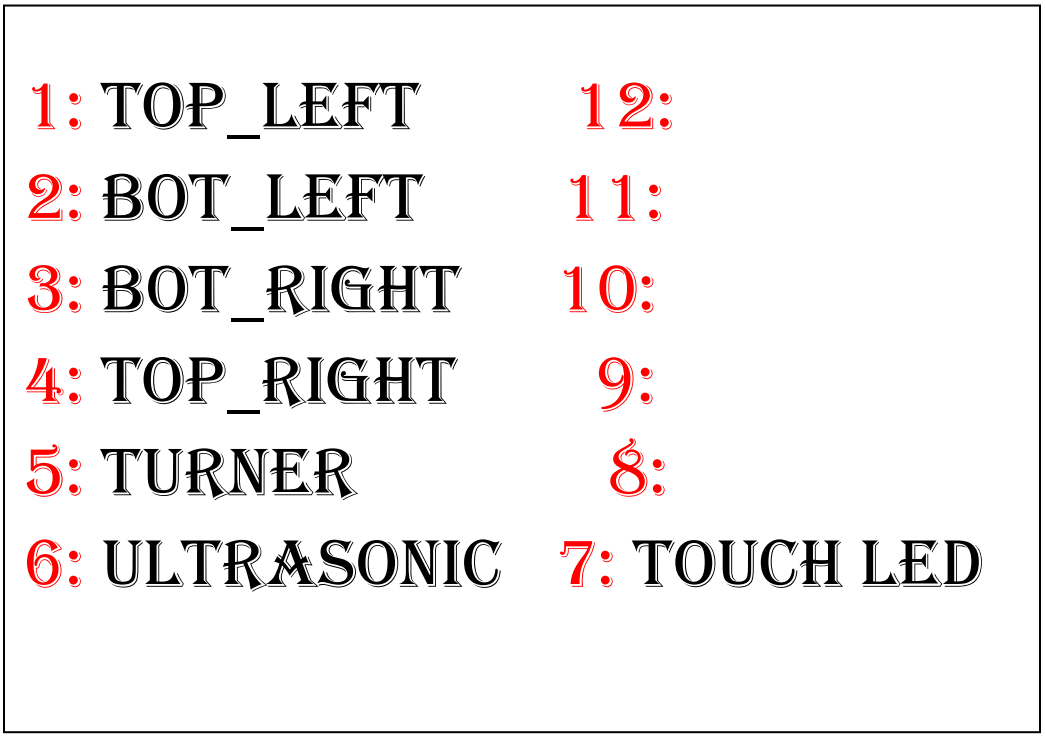
```
void right(){
  motor[motor1] = 31;
  motor[motor2] = -31;
  motor[motor3] = -31;
  motor[motor4] = 31;
  wait1Msec(750);
  if(dec==1){
    stopbot();
    rcan=dright;
    lcan=dleft;
    if(lcan==26214)
      lcan=0;
    if(rcan==26214)
      rcan=0;
    back();
  }
  dec = 2;
  // This code makes the robot go right exactly 1 foot.
}

void move(){
  if (ahead>500){
    fore();
  }
  else if(dleft>ahead)
  {
    if (dleft>dright)
    {
      left();
    }
    else {
      right();
    }
  }
  else if(dright>ahead){
    right();
  }
  else{
    fore();
  }
}
```

```
void step(){
  if(dec==0){
    setTouchLEDColor(color, colorGreen);
  }
  else if(dec==1)
  {  setTouchLEDColor(color, colorBlue);}
  else{
    setTouchLEDColor(color, colorRed);
  }
  check();
  move();
  stopbot();
}

task main(){
  init();
  while(true){
    step();
  }
  /* fore();
  left();
  back();
  right();
  */
}
```

PIC and Sensor Setup Page



Motor top_left Port #1
Use: Our motor located at the top left of the bot's base.

Motor bot_left Port #2
Use: Our motor located at the bottom left of the bot's base.

Motor bot_right Port #3
Use: Our motor located at the bottom right of the bot's base.

Motor top_right Port #4
Use: Our motor located at the top right of the bot's base.

Motor turner Port #5
Use: Our motor located at the front of the robot that turned the ultrasonic sensor.

Sensor sonar Ultrasonic Port #6
Use: This ultrasonic sensor gets the distance values from the front, left, and right of the robot.

Sensor color Touch LED Port #7
Use: This touch led turns a specific color to indicate which direction it is about to turn.

Project Results:

Primary Programming Goals:

1. Take 3 values using the ultrasonic sensor. One value for the front, one value for the left, and one value for the right.

Results: This primary goal was achieved. We used the ultrasonic sensor to get the distance values from the front, the left, and the right.

2. Use programming to take those three inputs and make a decision to go the path with the greatest distance

Results: This was also achieved. Once we got the three distance values, we set up a Check function that checked to see which one of the distances was the greatest one, and went down that path.

3. Has to be able to navigate a "canyon". Three equidistant objects, one front, one right and left. The robot has to be able to detect all three objects, recognize that it's stuck in a "canyon" and be able to back up.

Results: We also got this goal achieved. When the bot went left, and then right, it took both right and left measurements. The robot then started backing up until the new measurements were larger than the original ones.

Secondary Programming Goals:

1. Have the robot continuously checking all 3 sides. Meaning instead of having to jerk its head, it would have a continuously rotating head.

Results: We got this goal as well. Because we created a while loop to get the turner motor to rotate the ultrasonic sensor, the motor was constantly moving, which means the sensor was in continuous movement as well.

2. Have the robot be able to check distance backwards as well to see if backing up is a viable option.

Results: We were not able to get this check off. In the end, we decided that to implement this function, we would have to drastically change all of our logic loops and didn't want to spend that time.

Programming Notes:

There were multiple programming challenges we had to overcome. The first one was to figure out how to rotate the sensor so it could get accurate readings from all three directions. We solved this challenge by using the encoder values to initialize it to face forward, then using the encoder values, turn direct left and right. Another one was getting

the canyon code working. We tried using the threshold method, but that was giving us errors. In the end, when the bot went left, and then right, it took both right and left measurements. The robot then started backing up until the new measurements were larger than the original ones. We could not get a method to check if moving backwards was a viable choice, although we did implement an instance of this in our canyon code.

Project Conclusions

1. In retrospect, what was the biggest programming problem you had to overcome?

The biggest problem challenge to overcome was how to successfully implement the canyon code.

2. How did you finally solve this problem?

We tried using the threshold method, but that was giving us errors. Finally, we decided to keep a counter of when the bot moved left and right. When the bot went left, and then right, it took both right and left measurements. The robot then started backing up until the new measurements were larger than the original ones.

3. In retrospect, what was the biggest hardware integration problem you had to overcome?

The biggest hardware integration problem was where to position the motor to turn the ultrasonic sensor.

4. How did you finally solve this problem?

We finally solved it by putting it at an angle at the front of the base. We then had to initialize the head at an angle in the code, but using the encoder values, we were able to get it to face straight forward.

5. What one aspect of your code are you most pleased with?

The modularity of the code. We have everything arranged in clean and precise functions like Fore () and Check (). Our program makes very logical sense and is very reader-friendly. By keeping our code clean, we're able to quickly find where the errors were occurring.

6. What one aspect of your code are you least pleased with?

While we did get the canyon code working, we did in a shortcut way. If we had more time, I would have liked to have gone back and used the threshold method to detect if the robot was in a canyon, and then back up until the robot gets out of the canyon.

7. If you had to start again right now, what would you do differently?

I would have started the code with addressing the canyon code. We addressed the canyon code last in the program, which meant that we couldn't call certain methods from other methods because of the order of them in the program.

8. How did you deal with time management and cooperation during the project?

We had about one and a half weeks to get this program running due to FBLA. We started delegating the assignments, first to get the values to make the robot move a foot, then the different methods used to move the robot in the different directions. Towards the end, we would have one person waiting at the computer ready to debug the program, and one person on the field testing the program.

9. What concrete things did you learn about programming during this project?

In RobotC, the order of the functions matter. Whereas in Java or C#, the order of the functions doesn't matter, in RobotC you can't call a function that's below another function with a function that's above the other two.

10. What concrete skills did you acquire during this project?

From this project, I've learned how to successfully implement obstacle avoidance logic. Even though we only checked 3 directions in this program, using the obstacle avoidance skills I've learned from this projects, I could modify the logic to check 5 or more points (although that would be a pain).

11. If you had to give words of wisdom to future classes, what would you say was the key to making this project work?

When working with obstacle avoidance, make sure to plan out all the scenarios. Before you start the actual code, start a flowchart that will display all the different scenarios that your bot could be in, and what the code would make the robot do.