Name- Elgene Menon Leo Anthony

Username- leoanelge

Student ID-300492604

Progress Report

Introduction

The purpose of the Automated Vehicle Challenge (AVC) is to create a robot that can follow a line through four quadrants. In each quadrant a series of obstacles awaits the robot. The robot itself has two wheels, a camera and is controlled via Wi-Fi. The purpose of the AVC project is to for students to learn how to work on a complex project together. The skills needed for this project are software programming, hardware creation, trouble shooting and documentation of results. My part in this team is to create the code for the core and the completion tasks which are found in quadrants 1 to 3. I have now completed this work. In addition, I carried out extensive testing and debugging to make sure that the code operated the robot correctly.

Background

In the first quadrant our robot begins by going out of a gate. Therefore, our first obstacle was how to get the robot to open the gate. The robot physically carries a server. This server connects to the IP address of the gate via Wi-Fi. The gate then opens.

In the second quadrant our robot must continue to follow the line as it becomes curvy. Therefore, our second obstacle is how to program the robot to move in a curvy fashion following the line. To do this we designed our robot to operate in the following manner. When the camera records that the robot is in the centre of the black line then it commands both wheels to move at the same speed. If the camera records that it is moving towards the left side, then it will increase the speed of the left wheel and decrease the speed of the right wheel. This will bring the robot back to the centre of the line. The exact opposite process would apply if the robot was moving of towards the right side instead of going straight. By ensuring the robot always follows the centre of the line this means that when the line curves the robot curves as well as it seeks to maintain itself in the centre of the line.

In between quadrant two and quadrant three there is a piece of red paper in the shape of a circle that is placed on top of the black line. As it momentarily interrupts the flow of the black this obstacle is meant to stop the robot moving forward. I overcame this obstacle by teaching the camera that when it sees red pixels it should ignore them and keep moving forward in a straight line.

In the third quadrant the line turns in a series of sharp corners. At some of these corners the line splits and continues in two separate directions. Therefore, we had two obstacles to overcome. First how to get the robot to turn sharply and secondly how to make the robot only follow the line we wanted it to follow. To get the robot to follow the line and turn a sharp corner it had to stop moving one wheel completely while the wheel on the side we wanted it to turn to continued moving. For example, if we want the robot to follow the line through a sharp left hand turn it must stop moving its right-hand wheel completely while the left-hand wheel continues to move. This keeps the robot in the centre of the line as it turns the sharp corner. Secondly to ensure that the robot only turned the direction we wanted it to turn we sent it an instruction in code via Wi-Fi.

Methods

To open the gate, I used the following code. First, I created a method called OpenGate (). In that method I created a new array of server address which have the IP address of the gate. Then I connected it to the server using the array of the server address and the port number. I created another array called message[]. The message says “Please”. Then it is sent to the server. The next line of code shows that the password has 24 characters in the array. Then to open the gate the password must be received from the server to the gate and then the gate sends the password back to the server. This is what the code for the whole process looks like:

```
int ROBOT::OpenGate()
{   char server_addr[] = "130.195.6.196";
        connect_to_server(server_addr, 1024);
        char message[]= "Please";
        send_to_server(message);
        char password[24];
        receive_from_server(password);
        send_to_server(password);
        return 1;}
```

To call the method I use the code - robot.OpenGate() in the main() method.

In Quadrant Two I created four methods –
initHardware();
void setMotors();
int MeasureLine();
int FollowLine

For the method initHardware()I did as follows. I created a new variable called “int err” and I assigned it to “init(0)”. Then I called my left wheel “v_left” and my right wheel “v_right.” Both had a value of “48”. “48” stops the wheel.
The camera rises up from the robot. To do this I assign the value of “35.”
Next I call the method “setMotors()” and then set the camera by using the following code “set_motors(3, cam_tilt)”. Then I code “return err”.

The entire code is as follows:

```
int ROBOT::initHardware()
{   int err;
    cout<<" Hello"<<endl;
    err = init(0);
    cout<<"After init() error="<<err<<endl;
v_left = 48;
v_right = 48;
cam_tilt = 35;
dv=0;
setMotors();
set_motors(3, 35);
return err;
```

The method “setMotors()”sets the speed values for the left, the right wheel and the camera height. The code is as follows:

```
set_motors(5, v_left);
set_motors(1, v_right);
set_motors(3, 35);
```

The method “MeasureLine()” uses the camera to measure the pixels of the rows and columns of the black line. By doing this it detects whether the robot is following the centre of the line or whether it has gone off to either side. It measures the threshold of line to determine this. The code is as follows:

```
int ROBOT::MeasureLine()
{
        take_picture();
        update_screen();

        int row = 120;
        int col = 0;
        int max = 0;
        int min = 256;

        for( col = 0; col < cam_width; col++)
        {
                int wh = get_pixel(row, col, 3);
                if (wh > max){ max = wh;}
                if (wh < min){ min = wh;}
        }
        line_present= 1;

        if(min > 80)
        { line_present= 0;
                return -1;
        }

        double throuse_hold = (max + min) / 2.0;
        cout<<"Threshold: " << throuse_hold  <<endl;
```

```
        line_error = 0.0;
        n_black = 0;

        for( col = 0; col < cam_width; col++)
        {
                if( get_pixel(row, col, 3) > throuse_hold)
                {
 }
if( get_pixel(row, col, 3) <= throuse_hold)
{
//cout << "col: "<<col<<" ";
   line_error += (col - cam_width /2.0);
    n_black++;
  }
        }

           if(n_black>280){
                        if(NumleftTurn < 3){
                                turnLeft();
                        }
                        else{

                        turnRight();
                        }NumleftTurn++;
             }

        if(n_black == 0)
        {
                line_present= 0;
                return -1;
        }

        line_present= 1;
        cout << "line error: " << line_error<<endl;
        line_error = line_error/n_black;
        return 0;
}
```

The method “FollowLine()” makes the robot move along the black line. If the robot deviates from the black line it makes the robot go back to the black line. It does this by calculating the Kp value. The code is as follows:

```
int ROBOT::FollowLine()
 {  MeasureLine();
          if (hardTurn==true){
          hardTurn= false;
          return 0;
  }
          cout <<"Line Present: " << line_present << endl;
          if(line_present)
```

```
        {  dv =(int)(line_error*kp);
                v_left = v_left_go + dv;
                v_right =v_right_go + dv;
                cout<<"line_error= "<<line_error<<"dv= "<<dv;
                setMotors();
        }

   else
        {  //go back
                cout<<"line is missing"<<endl;
                v_left = 43;
                v_right = 52;
                setMotors();
                sleep1(100);
        }
        return 0;
}
```

At the entrance to Quadrant 3 the black line temporarily is replaced by a piece of red paper. This is meant to confuse the robot. To overcome this obstacle, I made coded the camera so that whenever it recorded red pixels it should ignore them and keep moving in a straight line until it encounters the black line again.
The code for this is as follows:

```
int ROBOT::redFlag()
{
        // count number of red
        int n_red=0;
        int row = 120;
        int col = 0;
        for( col = 0; col < cam_width; col++)
        {
                if( get_pixel(row, col, 0) <= 50)
                {
                        //cout << "n_red col: "<<col<<" ";

                        n_red++;
                }
        }
        if (n_red < 20 && paperFlag == false) //
        {
                paperFlag = true;
                set_motors(5,53);
                set_motors(1,42);
                setMotors();
                sleep1(1000);
        }
        return 0;
```

To handle the sharp corners in Quadrant 3 I slightly adjusted the code from Quadrant 2 which was used for the curvy line. The principle is the same. Instead of slowing down one wheel to cause the robot to follow a curve I instead made one wheel stop completely while the other remained running in order to force the robot to make a sharp turn.

```
    v_left = 48;
    v_right = 48;
void ROBOT::setMotors()
 {

          cout<<"v_left= "<<v_left<<"\t v_right= "<<v_right;
          set_motors(5, v_left);
          set_motors(1, v_right);

    set_motors(3, cam_tilt);
          hardware_exchange();
 }
```

The other problem in Quadrant 3 is that in three places where the line makes a sharp corner it splits into two. Therefore, I had to figure out how to make the robot only follow the line that I wanted it to follow. The first two turns I wanted the robot to turn left and on the last turn I wanted the robot to turn right.
I made the robot turn in the direction I wanted by creating an account of the turns instructing the robot which way to turn at each sharp turn.
I did this by writing code that told the robot if the black pixels were more than 280 and less than “(NumleftTurn < 3)” then it should turn left.
Otherwise the robot was instructed to turn right. When the robot turns left it uses this code:

```
if(n_black>280){
if(NumleftTurn < 3){
turnLeft();
}
else{
turnRight();}
NumleftTurn++;
}

int ROBOT::turnLeft()
{
          v_left = 52;
    v_right = 37;
    setMotors();
    sleep1(1000);
    return 0;
}
```

To turn the robot right I used this code:

```
int ROBOT::turnRight()
{
          v_left = 37;
```

```
    v_right = 52;
    setMotors();
     sleep1(1000);
    return 0;
}
```

The main method that is used for all the code is as follows:

```
int main()
{
        ROBOT robot;

        robot.initHardware();
        //robot.OpenGate(); Open Gate 2/2
        int count = 0;
        open_screen_stream();

        while( count < 5000)
        {
                take_picture();
                update_screen();
                robot.FollowLine();
                robot.redFlag();
                }
                count++;
        close_screen_stream();
        stoph();
        return 0;
}
```

Results

The Results of our testing of the robot for quadrants 1 and 2 were excellent. The robot did exactly what I wanted it to do. It opened the gate in quadrant 1 and in quadrant 2 the robot followed the curvy line with no problems. I have not yet been able to complete a test on quadrant 3. This is because I have only discovered how to turn the robot to the right today. In addition, we have had a hardware problem with the robot that caused our first camera to break during initial testing of quadrant 3.

Discussion

I have now completed all my assigned work for the group. I have written the code for quadrants 1 to 3. As discussed, I have not yet been able to test quadrant 3. However once the camera is replaced on the robot, I will be able to test quadrant 3 and I believe it will work.

There are two things that could have been improved in the work thus far. First, I ended up doing all the testing and debugging of the code even though this was not

my assigned task in our original plan. Secondly, the unfortunate breakage of the camera meant that testing quadrant three must be delayed.