Name- Elgene Menon Leo Anthony

Username- leoanelge

Student ID-300492604

Automated Vehicle Challenge Final Report

Abstract

This report details the creation and testing of a small automated robot that is required to navigate a series of obstacles in a testing course without a human operator. To do this a two wheeled vehicle powered by a battery and operated by a Raspberry PI was created. The robot did all that it was required to do and passed the testing course with a perfect mark. The importance of this result is that it shows how well automated vehicles can run even with low cost and low powered components.

Introduction

This report details the creation of a small robot designed to independently manoeuvre through a series of obstacles on a testing course. The report will cover the background knowledge needed to understand this material. It will detail the methods used to build the robot. It will describe the results of the testing of the robot. It will discuss the implications of these results. The report is meant to investigate the best ways to create software and hardware for a robot that can function independently without a human operator.

In brief the robot is meant to open a small gate by sending it an opening command via Wi-Fi. Then the robot must follow a black line that changes from straight to wavy and is interrupted by a red square. Then the robot must choose the right direction to turn when the black line splits in different directions. Finally, the robot must stop following the black line and approach three coloured pillars in a specific sequence. The benefits of this project are that it helps team members to develop software and hardware design, building and testing skills.

Background

There are three concepts to be aware of before reading this report. These are PID, C++ and Raspberry PI.

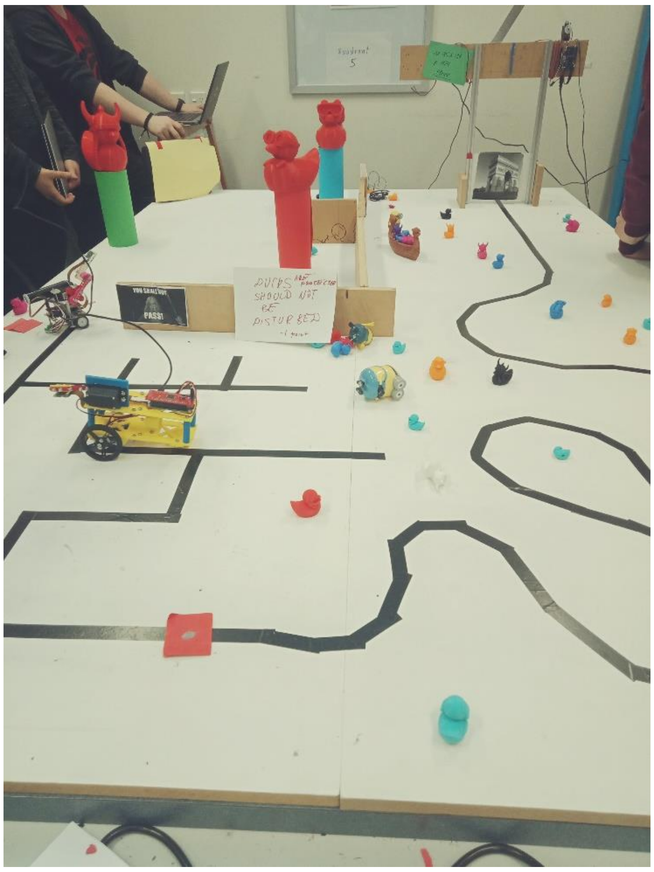
PID stands for proportional integral derivative and it is a control system used in a variety of engineering and manufacturing fields. A brief definition is "A PID controller is an instrument used in industrial control applications to regulate temperature, flow, pressure, speed and other process variables."[1] In relation to the

[1] (2019, Apr). What is a PID Controller? *Omega*. [Online]. Available: https://www.omega.com/en-us/resources/pid-controllers

robot our team used a PID formula in the computer code to keep the robot moving at a steady pace.

C++ is defined by the Encyclopaedia Britannica as a "high-level computer programming language … it is based on the traditional C language but with added object-oriented programming and other capabilities. C++, along with Java, has become popular for developing commercial software packages that incorporate multiple interrelated applications."[2] In relation to this robot all of the software was written in the C++ programming language.

The Raspberry PI is "a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT)."[3] For this robot the team used a Raspberry PI as the computer that operated the robot.



A photo of the full testing course

[2] E. Gregersen. (2012, Sept). C++. Encyclopaedia Britannica. [Online]. Available: https://www.britannica.com/technology/C-computer-language

[3] (2019, June). What is a Raspberry PI? Opensource. [Online]. Available: https://opensource.com/resources/raspberry-pi

Method

Software

Quadrant 1

In the first quadrant the robot begins by going out of a gate. Therefore, the 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.

To open the gate, the team did the following. First, the team created a method called" OpenGate ()". In that method the team created a new array containing the IP address of the gate. Then the team used the "connect_to_server ()" function which contains an array with the server address and the port number of the gate server.

Then the team created another array called "message[]". The message reads "Please". This message is sent from the robot to the gate server. The gate then sends back a 24-character password to the robot. The robot then sends this password back to the gate server which opens the gate [i]

To call the method the team used the code – "robot.OpenGate()" in the main() method.[ii] The robot can now follow the black line through the gate and into Quadrant 2.

Quadrant 2

In the second quadrant the robot must continue to follow the line as it becomes curvy. Therefore, the second obstacle is how to program the robot to move in a curvy fashion following the line. To do this the team 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 off 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.

For the method "initHardware()" the team did the following. The team created a new variable called "int err" and the team assigned it to "init(0)". Then the team called the left wheel "v_left" and the right wheel "v_right." Both had a value of "48". "48" stops the wheel.

The camera rises from the robot. To do this the team assigned the value of "35."

Next the team called the method "setMotors()" and then set the camera by using the following code "set_motors(3, cam_tilt)". Then the team coded "return err".[iii]

The method "setMotors()"sets the speed values for the left, the right wheel and the camera height.[iv]

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. [v]

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 calculating the Kp value. [vi]

Quadrant 3

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, the team 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. [vii]

To handle the sharp corners in Quadrant 3 the team 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 the team instead made one wheel stop completely while the other remained running in order to force the robot to make a sharp turn.[viii]

The other problem in Quadrant 3 is that there are three intersections where the line makes a sharp corner and splits into two. Therefore, the team had to figure out how to make the robot only follow the line in the direction that the team wanted it to follow. The first two turns the team wanted the robot to turn left and on the last turn the team wanted the robot to turn right.

The team made the robot turn in the direction it wanted by creating a count of the turns instructing the robot which way to turn at each sharp turn.
The team did this by writing code that told the robot if the black pixels were more than 280 and less than “(NumleftTurn < 2)” then it should turn left. Otherwise the robot was instructed to turn right. [ix]

Quadrant 4

Quadrant 4 begins when the black line ends. When this occurs, the camera must tilt up. There are three pillars in Quadrant 4. The first pillar is red, the second is green and the third is blue. A piece of yellow paper suspended vertically above the testing ground signifies the end of the course.

To complete Quadrant 4 the robot must drive up to each colour in the following order – red, green and blue. It must stop before it knocks the pillar over and then reverse slightly and continue to the next colour. To complete the course the robot must drive up and stop in front of the yellow paper.

Up until this point the camera has been facing down so that it can track the black line on the ground. However now the camera must tilt up so that it can recognize the colours of the pillars. To get the camera to tilt up Motor 3 is used to adjust the camera angle from 35 degrees (the downward facing position) to 62 degrees (the upward facing position).[x]

Red pillar

Now the robot must advance to the red pillar, approach it and stop without hitting it. To do this the robot must turn right, look for red pixels and then move forward in a straight line toward the pillar.

To turn right motor 5 which operates the left wheel must keep spinning and motor 1 which operates the left wheel must stop.[xi]

Now the robot must find the red pillar, advance to it and stop without hitting it. To find the red pillar the robot must narrow the width of the camera's perception and measure the number of red pixels. [xii] Then the "setMotors()" method is used again to move the robot forward to the red pillar and the "Sleep1(700)" function is used to stop the robot before it hits the red pillar.

The robot must now reverse, turn left, find the green pillar, advance towards the green pillar and stop before hitting it.

If the robot is going forward the range for the left wheel is 49 to 65 and the range for the right wheel is 46 to 30. Therefore, to make the robot reverse instead of going forward the ranges of the two wheels should be switched. This means the range of the left wheel is 46 to 30 and the range of the 49 to 65. Doing this causes the robot to reverse. To stop the robot reversing the "sleep1(2000)" function is used. [xiii]

The robot must look now left – to do this the "lookLeft ()" method is used. The "lookLeft()" method works by stopping the left wheel and keeping the right wheel spinning. [xiv]

Green pillar

To find the green pillar the robot must narrow the width of the camera's perception and measure the number of green pixels.[xv] Then the "setMotors()" method is used again to move the robot forward to the green pillar and the "Sleep1(700)" function is used to stop the robot before it hits the green pillar.

The robot must now reverse away from the green pillar. The process for doing this is the same as that which was used for the previous pillar.[xvi]

The robot must now look right. The "lookRight ()" method is used. The "lookRight ()" method works by stopping the right wheel and keeping the left wheel spinning.[xvii]

Blue Pillar

To find the blue pillar the robot must narrow the width of the camera's perception and measure the number of blue pixels.[xviii] Then the "setMotors ()" method is used again to move the robot forward to the blue pillar and the "Sleep1(700)" function is used to stop the robot before it hits the blue pillar.

The robot must now reverse away from the blue pillar. The process for doing this is the same as that which was used for the red and green pillars.[xix]

The robot must now look left. The "lookLeft ()" method is used again. The "lookLeft ()" method works by stopping the left wheel and keeping the right wheel spinning.

Yellow Paper

The robot must now advance to the yellow paper suspended vertically above the testing course and stop in front of it. To do this the robot must narrow the width of the camera's perception and search for yellow pixels.[xx] To move toward the yellow paper the robot uses the "setMotors ()" method again. When it reaches the yellow paper, the robot reverses slightly to align itself squarely towards the yellow paper and then stops. To reverse the robot uses the same method as it used for the red, green and blue pillars. To stop the robot uses the “sleep1 (2000)” function.[xxi] We added a victory dance function to our robot which was not a required part of the assessment.[xxii]

Hardware

Building Materials

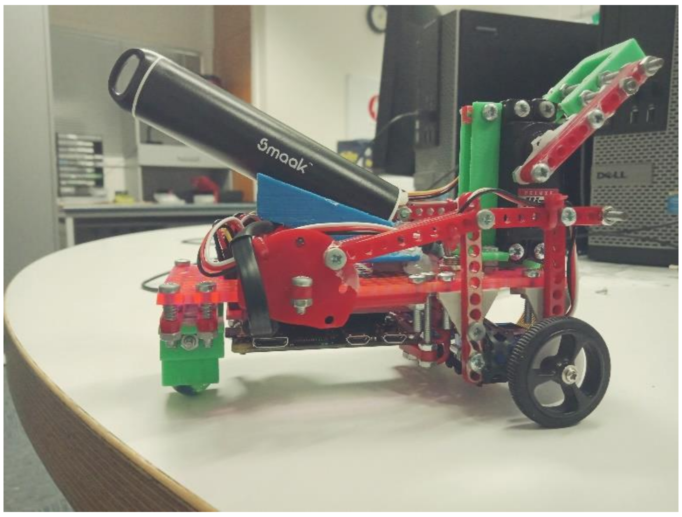
The robot was assembled with spare parts, three dimensional (“3D”) printed parts and specific ready-made components. The most important specific ready-made components were the Raspberry PI, the camera, the camera servo and the battery pack. The most important 3D printed component was the chassis as this was the component that all the other components were attached to.

Design

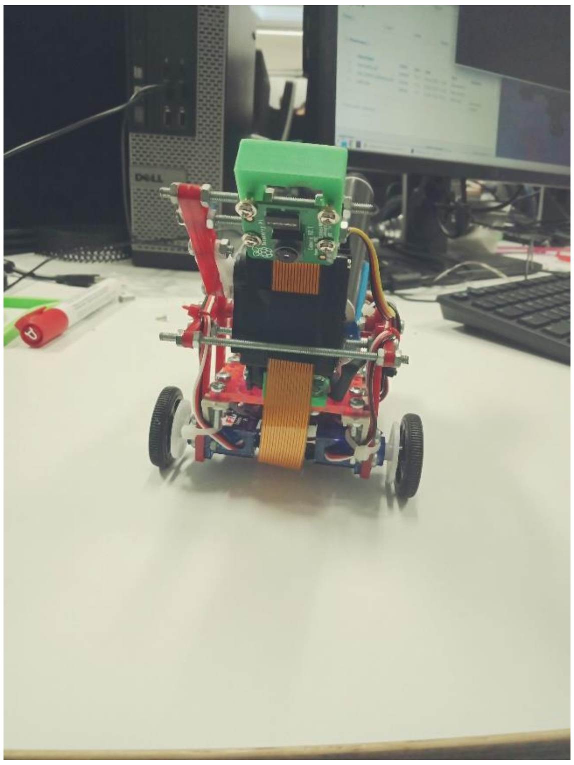
The chassis is a rectangle of 3D printed plastic. On the bottom side of the chassis, in the centre of one end of the rectangle is plastic housing that contains a marble. The marble does not move within the housing. Instead it is dragged along behind the front two wheels

Also, on the bottom side of the chassis is a Raspberry PI. It sits in the approximate centre of the rectangular shaped chassis. At the opposite end of the rectangular chassis there are two wheels. These are also attached to the bottom end of the chassis. These two wheels are placed on opposite sides of the same end of the chassis. The two large wheels now sit in the “front” of the chassis and the marble sits in the centre of the “back” end of the chassis.

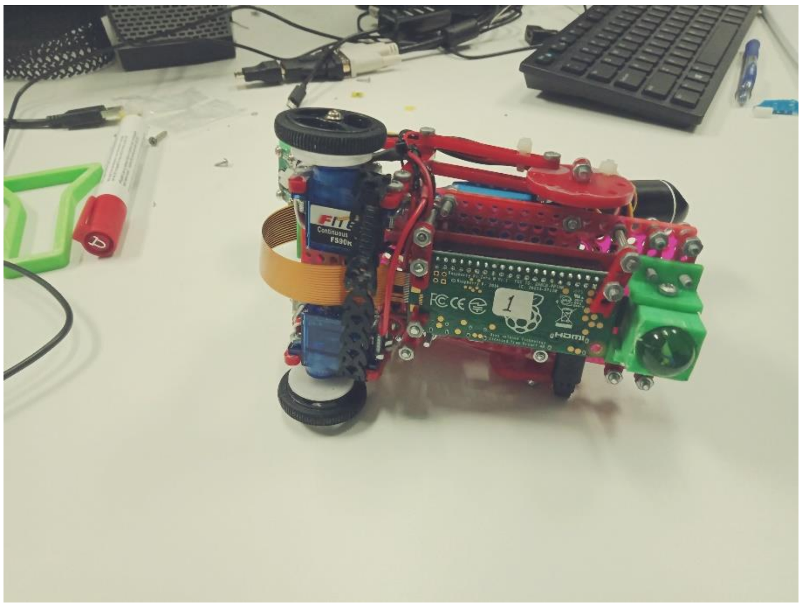
On the top side of the chassis directly above the two front wheels is the camera servo and camera. The camera servo has a turning mechanism allowing the position of the camera to be changed. Behind this sits the battery cradle. It is a long plastic trough that extends almost the entire rest of the length of the top side of the chassis. However, it sits on one side of the chassis not in the centre. A cylindrical shaped battery in plastic rests in this cradle.



Side view of the robot



Front view of the robot. The ribbon which caused difficulties is bronze coloured.



A view of the bottom side of the chassis

Key Design Choices

There were several key design choices which helped increase the efficacy of our robot. First, our team used smaller wheels. Therefore, our robot was easier to manoeuvre than many of our classmate's robots. Second, placing the camera and camera servo as far forward as possible was important for two reasons. It helped with weight distribution because the battery and battery cradle weighted down the back of the chassis significantly. Also putting more weight on top of the front wheels reduced the friction on the marble. This is important because the marble already generates a lot of friction as it is stationary. Thirdly, the team attached an on/off switch with an indicator light to our robot. Many other teams did not do this, and it allowed us to switch our robot on and off without having to unplug the cord from the battery. Finally, our team attached the raspberry PI to the bottom side of the chassis not the top side. This was important for two reasons. First there was not enough room on top of the chassis for both the Raspberry PI and the battery. Also, the battery was not attached to the battery cradle with any fastening mechanism. This meant that Raspberry PI was protected in case the battery fell out of the battery cradle.

Teamwork

Our team consisted of four members. The project tasks were divided amongst the four team members. Bronte Sheehan was the Project Lead and hardware support with the responsibilities of organising team meetings, reporting on progress and designing the 3D printed components on Computer Assisted Design (CAD) software. Elgene Leo Anthony was responsible for Software Development. This meant writing the code for Quadrants 1 to 3 and extending functionality. Jason Lim was responsible for software documentation. This meant debugging software and committing to git, writing test cases and documenting performance against milestones. Easton Miller Jose was responsible for Hardware. This meant building the chassis, testing components, connecting sensors and debugging hardware. Our team coordinated our efforts using GitHub.[4]

Results

Final Assessment Results

The results of the final assessment of the robot were excellent. Our team received 100% on the final assessment of our robot. The robot completed all four quadrants of the testing ground and did everything it was required to do.

Problems in Testing

The creation of the code for Quadrants 1 and Quadrants 2 went well. The code worked as it was supposed to do for both Quadrants. In addition, there were no hardware problems at this stage. The robot was able to open the gate and follow the black straight and curvy lines. No significant debugging of the hardware or software was required for Quadrants 1 and 2.

Significant problems only occurred with both the robot hardware and software when it came time to test the robot in Quadrant 3. The software initially could not get the robot to turn at the intersections. After significant debugging of the code this problem was solved.

It was also in Quadrant 3 that the most major hardware problem eventuated. A ribbon runs from the camera on top of the chassis over the front of the chassis, between the front two wheels and connects to the Raspberry PI on the bottom of the

[4] B. Sheehan. (2019, June). AVC Project. *GitHub*. [Online]. Available: https://github.com/brontesheeh/AVCproject

chassis. This is a key wire as it connects the Raspberry PI to the camera. However, it kept dragging on the ground and becoming disconnected. Therefore, the robot would stop working. This problem was fixed by binding the ribbon closer to the chassis with cable ties.

The testing of Quadrant 4 required no improvements to the hardware. The code also worked well without any significant debugging on its first attempt.

Discussion

Discussing the Results

The final assessment result demonstrates that the team's software and hardware choices were adequate to complete the task assigned.

In terms of the software the only significant debugging that was required during the testing phase was in Quadrant 3. This was because up until this point the robot was simply following the black line. When the black line split into more than one direction our robot did not know which way to turn. Our team solved this problem by embedding a count of the turns into the code which instructed the robot which way it should turn at each of the intersections.

In Quadrants 1, 2 and 4 the code performed well the first time it was tested, and significant debugging was not required. Therefore, overall the software results were good.

The results of the hardware testing were also good. The only significant problem that occurred was with the ribbon wire connecting the camera and the Raspberry PI becoming disconnected. However, this was fixed very easily with the addition of cable ties. Therefore, the results show the design and build of the hardware were fundamentally adequate for the task our team was given.

Discussing Improvements

Although the robot was adequate, and it completed all the tasks found in the final assessment there are several minor improvements that could improve the robot.

Obviously as discussed above it is important to ensure that the ribbon wire is closely connected to the body of the chassis so that it cannot drag along the ground and become disconnected.

The placement of the battery and battery cradle were not included in the initial design and build of the robot hardware. A large space was reserved for them on the top side of the chassis behind the camera and camera servo. Therefore, when the battery and the battery cradle were attached their position was not ideal. The battery and the battery cradle were attached on the left-hand side of the chassis not in the centre of the chassis. As the battery and battery cradle were quite heavy this meant that the entire robot was unbalanced and could have potentially fallen over on to its left-hand side. Therefore, the robot could be significantly improved by moving the battery cradle and battery to the centre of the top side of the chassis.

The robot could probably be improved by being a four-wheel instead of a two-wheel vehicle. The placement of the stationary marble in the centre of the back of the robot means the robot is unbalanced on the back end. This is especially a problem when the robot itself is already unbalanced on the left side due to the placement of the battery and battery cradle.

The reason the robot only has two wheels is because the amount of power generated by the front two wheels is only enough to drag along the marble not two additional wheels. Therefore, turning the robot into a four-wheel vehicle would involve either increasing the power of the front two wheels or adding two back wheels that have their own motors. However, this last option would require significant rewriting of the code as the robot turns left and right operating its two front wheels independently at different speeds. New code would therefore have to be written to run both back wheel motors as well.

Conclusion

In conclusion, our robot performed very well and was well suited to the tasks it was assigned to accomplish. Our team built a two-wheeled robot that was automated to run through a testing course and overcome specific obstacles designed to cause failure. Our robot was able to do this and received a mark of 100% on the final testing assessment. This technology could be applied to any task that requires a vehicle to move without direct control from a human user. A real-world example of this is the driverless car technology being developed by companies such as Tesla and Uber.[5] Therefore the AVC robot built for this project has developed important skills for our team in our future careers in the ICT industry.

[5] P. Leskin. [2019, June]. Uber just revealed a new self-driving car that it will use to take on Tesla and Waymo in the robotaxi wars. Business Insider. Online. Available: https://www.businessinsider.com.au/uber-self-driving-volvo-car-competes-tesla-waymo-robotaxi-wars-2019-6?r=US&IR=T

APPENDIX

i

```
int ROBOT::OpenGate()
{   char server_addr[] = "130.195.6.196";
        connect_to_server(server_addr, 1024);
        char message[]= "Please";
        send_to_server(message);d
        char password[24];
        receive_from_server(password);
        send_to_server(password);
        v_left = 63;                        // fowards speed range from 49 -> 65 | left
wheel motor 5
   v_right = 30;                            // fowards speed rande from 46 -> 30 | right
wheel motor 1
   setMotors();
   sleep1(2000);                            //go straight at max speed for 2 seconds,
then find and follow line
        return 1;
}
```

ii

```
void setMotors();
int initHardware();
int measureLine();
int followLine();
```

iii

```
int ROBOT::initHardware()
{       int err;
   cout<<"Start"<<endl;
   err = init(0);
   cout<<"After init() error: "<<err<<endl;
   v_left = 48;                                 // fowards speed range from 49 -> 65
| left wheel motor 5
   v_right = 47;                                // fowards speed range from 46 -> 30
| right wheel motor 1
        cam_tilt = 62;                          // 35->faces up......62->faces down
        dv = 0;
        setMotors();
        return err;
}
```

iv

```
void ROBOT::setMotors()
{   //cout<<"v_left: "<<v_left<<"     v_right: "<<v_right<<endl;
        set_motors(5,v_left);                   // fowards speed range from 49 -> 65
| left wheel motor 5
```

```
        set_motors(1,v_right);                  // fowards speed rande from 46 -> 30
| right wheel motor 1
        set_motors(3, cam_tilt);
        hardware_exchange();
}
```

v

```
int ROBOT::measureLine()
{       int row = 120;
        int col = 0;
        double threshold = 70;
        take_picture();
        update_screen();
        linePresent= true;
        lineError = 0.0;
        n_black = 0;
        for( col = 0; col < cam_width; col++)
        {       if( get_pixel(row, col, 3) <= threshold)
                {       lineError += (col - cam_width/2.0);
        //calculate line error
                        n_black++;
                }
        }
        //cout <<"Number of Black Pixels: "<<n_black<<endl;
        if(n_black>300)
                //if intersection ahead
        {       if(numOfTurns < 2)
        //go left if number of left turns is less than 2
                {       turnLeft();
                }
                else                            //go right if number of left turns has
been reached
                {       turnRight();
                }
                numOfTurns++;
        }
        //cout<<"Number Of Turns: "<<numOfTurns<<endl;
        if(n_black < 10)
        {       linePresent = false;
                return -1;
        }
        linePresent = true;
        //cout<<"Line Error: "<<lineError<<endl;
        lineError = lineError/n_black;
        return 0;
}
```

vi

```
int ROBOT::followLine()
{       measureLine();
```

```
        if (hardTurn==true)
        {       hardTurn = false;
                return 0;
        }
        //cout<<"Line Present: "<<linePresent<<endl;
        if(linePresent==true)
        //follow line
        {       dv = (int)(lineError*kp);
                v_left = v_left_go + dv;
                v_right = v_right_go + dv;
                //cout<<"Line Error: "<<lineError<<"   dv: "<<dv<<endl;
                setMotors();
        }
        else                                    //reverse
        {       //cout<<"Line is Missing"<<endl;
                v_left = 43;
                v_right = 52;
                setMotors();
                sleep1(100);
        }
        return 0;
}
```

vii

```
int ROBOT::redFlag()
{       int n_red=0;
        int row = 120;
        int col = 0;
        for(col = 0; col < cam_width; col+=2)
        {       int red = get_pixel(row, col, 0);                       //get amount of red in
pixel
                int green = get_pixel(row, col, 1);                     //get amount of green in
pixel
                int blue = get_pixel(row, col, 2);                      //get amount of blue in
pixel
                //cout<<"Red: "<<red<<"     Green: "<<green<<"     Blue:
"<<blue<<endl;
                if (green+blue<red-20)                                          //count number
of red pixels mid screen
                {       n_red++;
                }
        }
        //cout<<"Number of Flags: "<<paperFlag<<endl;
        if (n_red > 20)
        //flag count +1
        {       paperFlag++;
                v_left = 52;            // fowards speed range from 49 -> 65 | left wheel
motor 5
                v_right = 42;           // fowards speed rande from 46 -> 30 | right wheel
motor 1
```

```
                setMotors();
                sleep1(1500);                    //go straight for 1.5 seconds, then
find line again
        }
        return 0;
}
```

viii

```
int ROBOT::followLine()
{       measureLine();
        if (hardTurn==true)
        {       hardTurn = false;
                return 0;
        }
        //cout<<"Line Present: "<<linePresent<<endl;
        if(linePresent==true)
        //follow line
        {       dv = (int)(lineError*kp);
                v_left = v_left_go + dv;
                v_right = v_right_go + dv;
                //cout<<"Line Error: "<<lineError<<"   dv: "<<dv<<endl;
                setMotors();
        }
        else                                    //reverse
        {       //cout<<"Line is Missing"<<endl;
                v_left = 43;
                v_right = 52;
                setMotors();
                sleep1(100);
        }
        return 0;
}
```

ix

```
int ROBOT::measureLine()
{       int row = 120;
        int col = 0;
        double threshold = 70;
        take_picture();
        update_screen();
        linePresent= true;
        lineError = 0.0;
        n_black = 0;
        for( col = 0; col < cam_width; col++)
        {       if( get_pixel(row, col, 3) <= threshold)
                {       lineError += (col - cam_width/2.0);
        //calculate line error
                        n_black++;
                }
        }
        //cout <<"Number of Black Pixels: "<<n_black<<endl;
        if(n_black>300)
                //if intersection ahead
        {       if(numOfTurns < 2)
        //go left if number of left turns is less than 2
                {       turnLeft();
                }
                else                                    //go right if number of left turns has
been reached
                {       turnRight();
                }
                numOfTurns++;
        }
        //cout<<"Number Of Turns: "<<numOfTurns<<endl;
        if(n_black < 10)
        {       linePresent = false;
                return -1;
        }
        linePresent = true;
        //cout<<"Line Error: "<<lineError<<endl;
        lineError = lineError/n_black;
        return 0;
}
```

x

```
int ROBOT::initHardware()
cam_tilt = 62;  // 35->faces up......62->faces down
```

xi

```
int ROBOT::lookRight()  //turn a small amount to the right
{   v_left = 52;                // fowards speed range from 49 -> 65 | left wheel motor 5
    v_right = 47;               // fowards speed rande from 46 -> 30 | right wheel motor
1
    setMotors();
    sleep1(700);
    return 0;
}
```

xii

```
int ROBOT::measureShape()
{       take_picture();
        update_screen();
        int row = 120;
        int col = 0;
        shapePresent = 1;
        shapeError = 0.0;
        n_red = 0;
        n_green = 0;
        n_blue = 0;
        n_yellow = 0;
```

```
if(shapePresent==true)                                                    //coloured
shape in sight, go to it
     {     dv = (int)(shapeError*kp);
           v_left = v_left_go + dv;
           v_right = v_right_go + dv;
           //cout<<"Shape Error: "<<shapeError<<"dv: "<<dv<<endl;



     for( col = 0; col < cam_width; col++)
     {     int red = get_pixel(row, col, 0);               //get amount of red in
pixel
           int green = get_pixel(row, col, 1);             //get amount of green in
pixel
           int blue = get_pixel(row, col, 2);              //get amount of blue in
pixel
           //cout<<"Red: "<<red<<"    Green: "<<green<<"    Blue:
"<<blue<<endl;
           if(reachedRed==false && reachedGreen==false &&
reachedBlue==false)
           //looking for red
           {     if(green+blue<red-20)           //is it a red pixel?
                 {     n_red++;

                       //add to the number of red pixels
                       shapeError += (col - cam_width /2.0);

     //calculate shape_error
                 }
           }
```

xiii

```
if(reachedRed==false && reachedGreen==false && reachedBlue==false)
                 //looking for red
     {     if(n_red>300)
                                                   //if red shape is reached
reverse for 1 second then look left
           {     reachedRed=true;
                 v_left = 43;          // fowards speed range from 49 -> 65 | left
wheel motor 5
                 v_right = 52;         // fowards speed rande from 46 -> 30 | right
wheel motor 1
                 setMotors();
                 sleep1(2000);
                 lookLeft();
                 return 0;
           }
           shapePresent = true;
           //cout<<"Shape Error: "<<shapeError<<endl;
```

```
                shapeError = shapeError/n_red;
                return 0;
        }
```

xiv

```
int ROBOT::lookLeft()        //turn a small amount to the left
{   v_left = 48;        // fowards speed range from 49 -> 65 | left wheel motor 5
    v_right = 43;                // fowards speed rande from 46 -> 30 | right wheel motor
1
    setMotors();
    sleep1(700);
    return 0;
}
```

xv

```
else if(reachedRed==true && reachedGreen==false && reachedBlue==false)
                                                        //looking for green

                {       if(blue<red && blue*2<green)//is it a green pixel?              ////
check values
                        {       n_green++;

                                //add to the number of green pixels
                                shapeError += (col - cam_width /2.0);

        //calculate shape_error
                        }
                }
```

xvi

```
else if(reachedRed==true && reachedGreen==false && reachedBlue==false)
                        //looking for green
        {       if(n_green>300)
                                                        //if green shape is
reached reverse for 1 second then look right
                {       reachedGreen=true;
                        v_left = 43;
                        v_right = 52;
                        setMotors();
                        sleep1(2000);
                        lookRight();
                        return 0;
                }
                shapePresent = true;
                //cout<<"Shape Error: "<<shapeError<<endl;
                shapeError = shapeError/n_green;
                return 0;
        }
```

xvii

```
int ROBOT::lookRight()  //turn a small amount to the right
{   v_left = 52;                    // fowards speed range from 49 -> 65 | left wheel motor 5
    v_right = 47;                   // fowards speed rande from 46 -> 30 | right wheel motor
1
    setMotors();
    sleep1(700);
    return 0;
}
```

xviii

```
else if(reachedRed==true && reachedGreen==true && reachedBlue==false)
                                //looking for blue
                    {       if(red<green && red<blue && red<100 && green>100 &&
blue>120 && green<150 && blue<170)          //is it a blue pixel?     //// check values
                            {       n_blue++;
                                                                        //add to the
number of blue pixels
                                    shapeError += (col - cam_width /2.0);
                                                    //calculate shape_error
                            }
                    }
```

xix

```
else if(reachedRed==true && reachedGreen==true && reachedBlue==false)
                                //looking for blue
        {       if(n_blue>280)
                                                                //if blue shape is
reached reverse for 1 second then look left
                {       reachedBlue=true;
                        v_left = 43;
                        v_right = 52;
                        setMotors();
                        sleep1(2000);
                        lookLeft();
                        return 0;
                }
                shapePresent = true;
                //cout<<"Shape Error: "<<shapeError<<endl;
                shapeError = shapeError/n_blue;
                return 0;
        }
```

xx

```
else if(reachedRed==true && reachedGreen==true && reachedBlue==true)
                                //looking for yellow
                    {       if(red>blue && green>blue && red>110 && green>110 &&
blue>40 && red<190 && green<190 && blue<100)                    //is it a yellow
pixel?                  //// check values
```

```
                              {         n_yellow++;
                                                                                                                        //add to the
number of yellow pixels
                                        shapeError += (col - cam_width /2.0);
                                                                                          //calculate shape_error
                              }
                    }
          }
```

xxi

```
else if(reachedRed==true && reachedGreen==true && reachedBlue==true)
                              //looking for yellow
          {         if(n_yellow>300)
                                                                                                              //if yellow shape is
reached reverse for 1 second then DANCE!!!!!!
                    {         v_left = 43;
                              v_right = 52;
                              setMotors();
                              sleep1(2000);
                              victoryDance();
                              count=10000001;
                              return 0;
                    }
                    shapePresent = true;
                    //cout<<"Shape Error: "<<shapeError<<endl;
                    shapeError = shapeError/n_yellow;
                    return 0;
          }
          return 0;
}
```

xxii

```
int ROBOT::victoryDance()  //hopefully this gets used...
{   v_left = 65;                              // fowards speed range from 49 -> 65 | left wheel
motor 5
    v_right = 65;                             // fowards speed rande from 46 -> 30 | right wheel
motor 1
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    cam_tilt = 62;
    setMotors();
    sleep1(600);
    cam_tilt = 35;
    setMotors();
    sleep1(600);
    return 0;
}

int ROBOT::run()
{       initHardware();
        OpenGate();
        open_screen_stream();
        while(count < 10000000)
        {       if (paperFlag < 2)                      //Q1, Q2, Q3
                {       //take_picture();
                        //update_screen();
                        followLine();
                        redFlag();
                        //cout<<"count: "<<count<<endl;
                }
                else if  (paperFlag > 1)                //Q4
                {       if(cam_tilt==62)                //if just getting to Q4 look up
                        {       cam_tilt = 35;
                                lookRight();
                                //sleep1(1000);
                        }
                        //take_picture();
                        //update_screen();
                        goToShape();
                }
                count++;
        }
        close_screen_stream();
        stoph();
        return 0;
}

int main()
{
        ROBOT robot;
        robot.run();
}
```