Table of Contents

Table of Pictures

Introduction

The purpose of this project is to familiarize students with image processing using C++. Deeper insight and knowledge is gained by refraining from using readily available, external, image processing libraries. Only code provided in the scope of this class and written by us students was used. The image processing theory is acquired in a learning by doing manner; via a project where we are to develop a robot, capable of firing a ping pong ball at an adversary's robot. The only input allowed to the robot is the commands of our C++ image processing program which is continuously running and sending the adequate instructions wirelessly. The image processing algorithm sends our robot the solutions it believes will best achieve its preprogramed objective of world domination and striking the opponent's robot with a ping pong ball. The project culminates with all teams participating in a project showcase and competition; similar to a tournament.

Parts & Equipment

Specifications

Arduino Uno

The Arduino Uno microboard was used to send and receive commands and controls the DC motors attached to the wheels and the firing system's servo motor. This breadboard is connected to all the components making up our robot. Power is provided by a battery pack located on the lower surface of our robot. An XBEE communication system acts as a messenger between our C++ image processing program and the ping pong firing robot controlled by the Arduino.

Servomotor

Figure 1-HS-422 Servo Motor

A servo motor was used for the firing mechanism, it loaded the trigger and fires the ball once the proper command is received from the Arduino.

Side Wheels

For the sidewheels, we have opted to use a pair of DC motors to control movement, front and back; as well as heading, by operating each of the two motors at different or opposite velocities.

When considering wheel design, we conceded that wheels of larger diameter would facilitate and speed up linear movement. We have chosen a wheel diameter of 65mm. Pivoting and rotation is obtained thanks to a ball caster with 1" plastic ball located at the front of the robot, underneath the firing barrel. Moreover, the wheel's thickness plays a part on the grip and agility of unit.

Camera

A PS3 eye camera with the following specifications was used. Giving us a steady stream of 30 FPS with a resolution of 640*480 pixels. Moreover, we had tuned the parameters of the camera such as gain, exposure and white balance manually to best fit the operating environment. As we know, in the realm of automation and image processing, it is crucial that the hardware be tuned to the most adequate settings before starting to tweak with the software aspect of things.

Figure 2-PS3 Eye Camera

Battery Pack

Figure 3 - ROTOR RC Extreme Power 25-50C 2500mAh 11.1V 3S1P

DC Motor Controller

Figure 4-L298 Dual H-Bridge DC MOTOR Controller

Design Process and Building

Vehicle's Frame

Our vehicle has a plywood chassis to which all the components forming our robot are attached. We had selected plywood as we believe it had all the characteristics necessary for an optimal design, being a lightweight and rigid material. Lightweight is a key characteristic when it comes to speed; lighter components would mean the ability to achieve a higher velocity using the same power source

The Firing Mechanism

Our firing mechanism had gone through several iterations before settling on the final design. We have opted for a system in which the firing barrel is stationary and aim is achieved by changing the orientation of the robot's chassis. The manufacturing technique chosen was to produce the part using an ABS plastic, 3D printer, programmed to a 60% density.

In designing the frame, we went under the assumption that a lower center of gravity would help with maneuvering characteristics. However, for all the components to fit properly a widebody design was selected.

Building Techniques

We had resorted to using double sided tape to make our prototypes and final version of the robot. This ensured flexibility and reusability allowing us to tinker and tweak our designs with greater ease.

Wiring

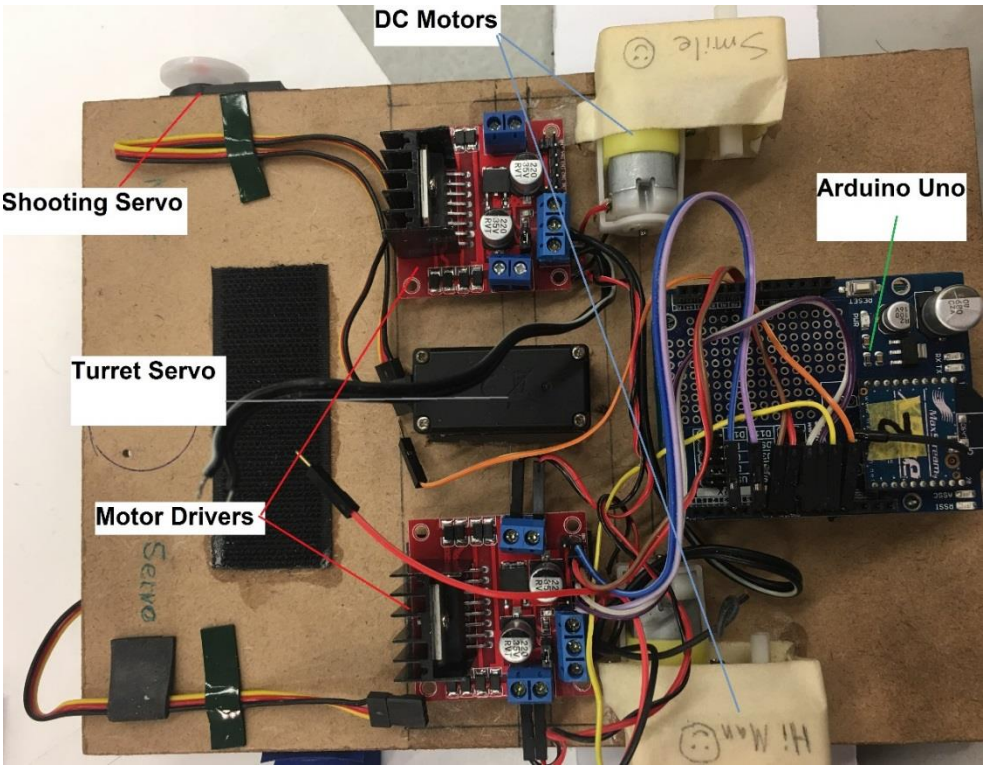

Figure 5 - The Wiring of the various components making up the Ping Pong firing robot

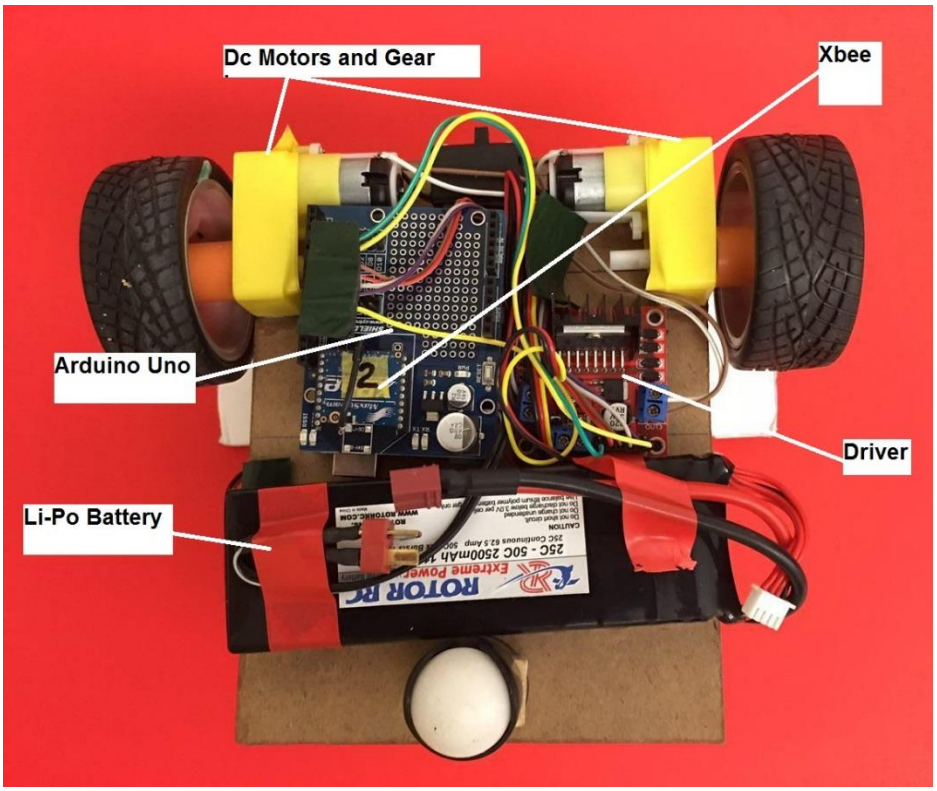

Figure 6- Image showing the robot's components from an additional perspective

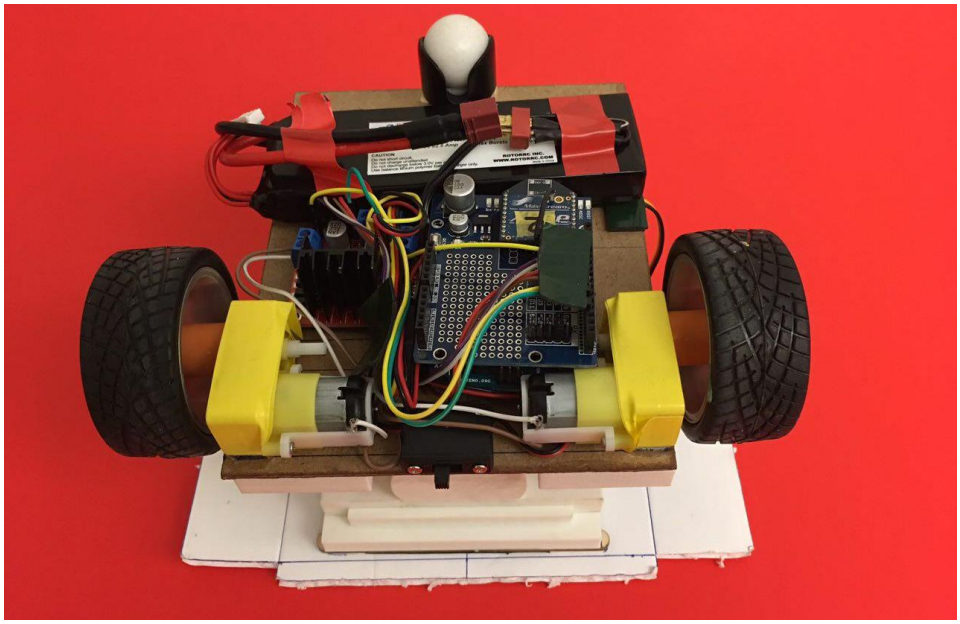

Figure 7- Image showing the underbelly of our Robot

Coding and Image Processing

Planning the Software and Writing the Code

As with all software development endeavors, we thought it would be best practice to have a robust pseudo code before ever writing a single line of C++ code. We wanted to share consensus on what the robot would be capable of doing and how it should go about the achieve those goals. The golden rule of image processing being garbage in, garbage out. Along this same guiding principle in automation, we have been though to attempt and manually complete a task before proceeding with its automation and algorithm development. This aids in better understanding the details as well as the insights of the process we are trying to automate.

Moreover, we thought it was crucial to first establish communication capabilities and collaborative tools between the various members of our group. We believed that using a cross platform, cloud based IDE, such as the one named "CodeShare.io" would enable us to contribute to the main source code individually, at our own pace, yet still doing so in collective and organized manner, without having to constantly worry about version control and overwriting.

Method for Color Identification

```
for (int n = 1; n <= nlabels; n++)
	{
		sum_r = 0;
		sum_g = 0;
		sum_b = 0;
		count = 0;
		r_ave = 0;
		g_ave = 0;
		b_ave = 0;

		// calculat the average value
		for (int i = 0; i < size; i++)
		{
			if (pl[i] == n)
			{
				count++;
				sum_b += p0[3 * i];
				sum_g += p0[3 * i + 1];
				sum_r += p0[3 * i + 2];
			}
		}
		b_ave = sum_b / count;
		g_ave = sum_g / count;
```

```
        r_ave = sum_r / count;
        // create the rgb_ave
        for (int i = 0; i < size; i++)
        {
                if (pl[i] == n)
                {
                        prgb_ave[3 * i] = b_ave;
                        prgb_ave[3 * i + 1] = g_ave;
                        prgb_ave[3 * i + 2] = r_ave;
                }
                else if (pl[i] == 0) // background, set to all white
                {
                        prgb_ave[3 * i] = 255;
                        prgb_ave[3 * i + 1] = 255;
                        prgb_ave[3 * i + 2] = 255;
                }
        }
}

for (int n = 1; n <= nlabels; n++)
{
        position = 0;
        double ic = 200.0, jc = 300.0;
        int ip, jp;

        centroid2(a, label, n, ic, jc);

        ip = ic;
        jp = jc;

        position = a.width*jp + ip;
        B = prgb_ave[position * 3];
        G = prgb_ave[position * 3 + 1];
        R = prgb_ave[position * 3 + 2];

        int G_B = abs(G - B);
        int R_B = abs(R - B);
        int R_G = abs(R - G);

        bool green = (R<110 && G>105 && B<120 );
        bool darkblue = (R<85 && G<120 && B>120);
        bool lightblue = (G>100 && R<110 && B>100 && G_B<25);
        bool red = (R>130 && G_B<25 && G<120);
        bool black = (R<100 && R_B<10 && R_G<10 );
        bool yellow = (R>150 && G>150 && B<180);
        bool purple = (R_B <20 && G<100 && R>80 && R_G>10);
```

This method makes use of the creation of an averaged image made up of averaged values of R, G and B to identify colors in an image. We had placed the colors to be used during the competition across the workspace and had taken image with the camera. We then manually inputted this image into the

IrfanView Software and using the mouse and pointer we determined the RGB characteristics of the colors of interest, by clicking on them on the image. Once we had those values they were hard coded into our source code to be used in the image processing algorithm. Our program compares pixels of interest to the RGB values hard coded into our source code for color identification purposes. If they match to what has been stored as a green, then the color is identified as being green. The second possible method for four colors is as follow. For other colors a similar procedure can be used:

```
int get_objects_color(int avg_blue, int avg_green, int avg_red)

{
	//Find object color based on Euclidean Distance method
	int red_RValue, red_GValue, red_BValue;
	int blue_RValue, blue_GValue, blue_BValue;
	int green_RValue, green_GValue, green_BValue;
	int color = 0;
	long R_diff, G_diff, B_diff;
	double red_dist, blue_dist, green_dist;

	// red color characteristics vector(255,0,0)

	red_BValue = 0;
	red_GValue = 0;
	red_RValue = 255;


	R_diff = red_RValue - avg_red;
	G_diff = red_GValue - avg_green;
	B_diff = red_BValue - avg_blue;
	red_dist = sqrt(R_diff*R_diff + G_diff*G_diff + B_diff*B_diff);

	// blue color characteristics vector(0,0,255)

	blue_BValue = 255;
	blue_GValue = 0;
	blue_RValue = 0;


	R_diff = blue_RValue - avg_red;
	G_diff = blue_GValue - avg_green;
	B_diff = blue_BValue - avg_blue;
	blue_dist = sqrt(R_diff*R_diff + G_diff*G_diff + B_diff*B_diff);

	// green color characteristics vector(0,255,0)

	green_BValue = 0;
	green_GValue = 255;
	green_RValue = 0;


	R_diff = green_RValue - avg_red;
	G_diff = green_GValue - avg_green;
```

```
        B_diff = green_BValue - avg_blue;
        green_dist = sqrt(R_diff*R_diff + G_diff*G_diff + B_diff*B_diff);



        // get minimum value
        double min = 500;
        if (red_dist<min) min = red_dist;
        if (blue_dist<min) min = blue_dist;
        if (green_dist<min) min = green_dist;



        if (min == red_dist && red_dist <= 210) color = 1;//red
        else if (min == green_dist && green_dist <= 210)  color = 2;//green
        else if (min == blue_dist && blue_dist <= 210) color = 3;//blue
        else if (red_dist>210 && blue_dist>210 && green_dist>210) color = 4;//black object


        else color = 0;

        return color;
        }
```

Escaping the Opponent

```
double getHidingAngle(double angle_robot, double angle_target, double ex)
{
        if (angle_target > angle_robot)//
        {
                if (abs(angle_target - angle_robot) > 180) return ex;

                else if (abs(angle_target - angle_robot) < 180) return -ex;
        }
        if (angle_target < angle_robot)//
        {
                if (abs(angle_target - angle_robot) > 180) return -ex;

                else if (abs(angle_target - angle_robot) < 180) return ex;
        }
}
```

This function is used to escape the opponent, when an opponent is detected, based on the angle between our robot and opponent's robot, the trajectory and direction of escape are identified and sent to the Arduino for execution.

```
double getBestAngle(double robot_x, double robot_y, double robot_r, double target_x,
double target_y, double obstacle_x, double obstacle_y, double angle)
{
 if (obstacle_y >= 480 - robot_r && robot_r < obstacle_x && obstacle_x < 640 - robot_r)
 if ((robot_x < obstacle_x && obstacle_x < target_x) ||
(robot_x<obstacle_x&&target_x<obstacle_x&&robot_y>obstacle_y&& obstacle_y>target_y) ||
 (robot_y<obstacle_y&&obstacle_y<target_y&&robot_x>obstacle_x&&target_x>obstacle_x))
 return angle;
 else
 return -angle;
 else if (obstacle_y <= robot_r&&robot_r < obstacle_x&& obstacle_x < 640 - robot_r)
 if ((robot_x < obstacle_x&&obstacle_x < target_x) ||
(robot_x<obstacle_x&&target_x<obstacle_x&&target_y>obstacle_y&&obstacle_y>robot_y) ||
 (robot_x > obstacle_x&&target_x > obstacle_x&&robot_y > obstacle_y&&obstacle_y >
target_y))
 return -angle;
 else
 return angle;
 else if (obstacle_x > 640 - robot_r)
 if ((target_y > obstacle_y&&obstacle_y > robot_y) ||
(robot_y<obstacle_y&&target_y<obstacle_y&&robot_x>obstacle_x&&obstacle_x>target_x) ||
 (robot_y > target_y&& target_y > obstacle_y&&robot_x < obstacle_x&&obstacle_x <
target_x))
 return -angle;
 else
 return angle;
 else if (obstacle_x <= robot_r&&robot_r < obstacle_y&& obstacle_y < 480 - robot_r)
 if ((target_y > obstacle_y&&obstacle_y > robot_y) || (robot_y < obstacle_y&& target_y <
obstacle_y&&robot_x < obstacle_x&&obstacle_x < target_x) ||
 (robot_y > obstacle_y && target_y > obstacle_y&&robot_x > obstacle_x&&obstacle_x >
target_x))
 return angle;
 else
 return -angle;
 else
 return -10000;

       if (abs(lastangel - angle) < 25)
              return angle;
       else
              return -angle;
}
```

When the obstacle is close to any of the workspace's edges, our robot will have to maneuver around the obstacle in a particular setting to reach the target and avoid going out of bounds. It will have to counter the obstacle from the side of the obstacle looking inside the workspace. Therefore, should the distance between the centroid of the obstacle and the boundaries of the workspace be less than that of

the radius of the robot, our robot has only one navigable course to reach its target. This function mitigates the issues arising from this circumstance, and sends the appropriate commands to our robot to enable it to reach the target still.

Robot, Target and Obstacle Identification

```
if (darkblue)
	{


		// on target label, the only green object# -- 1
		reset_label_number(label, target_label, n, 1);
	}

	else if (green)
	{

		reset_label_number(label, our_robot_label, n, 1);
	}
	else if (red)
	{

		reset_label_number(label, our_robot_label, n, 2);
	}
	else if (black)
	{
		reset_label_number(label, obstacle_label, n, 1);
	}

	else
	{

		for (int i = 0; i < size; i++)
		{
			if (pl[i] == n)
			{
				prgb_ave[3 * i] = 255;
				prgb_ave[3 * i + 1] = 255;
				prgb_ave[3 * i + 2] = 255;
			}
		}
	}
}
```

The identification of our robot, that of the opponent and the obstacle is done based on their colors. The above function shows how the labels of these items of interest are specified.

CAD Drawings

We started out with a CAD model of our robot. This enables us to visualize the design and bring about potential enhancements without a physical prototype being made. We then assembled the components and tested the viability and performance of the design. After an initial round of testing, changes were decided and acted upon.

Figure 8- General Components of the Robot

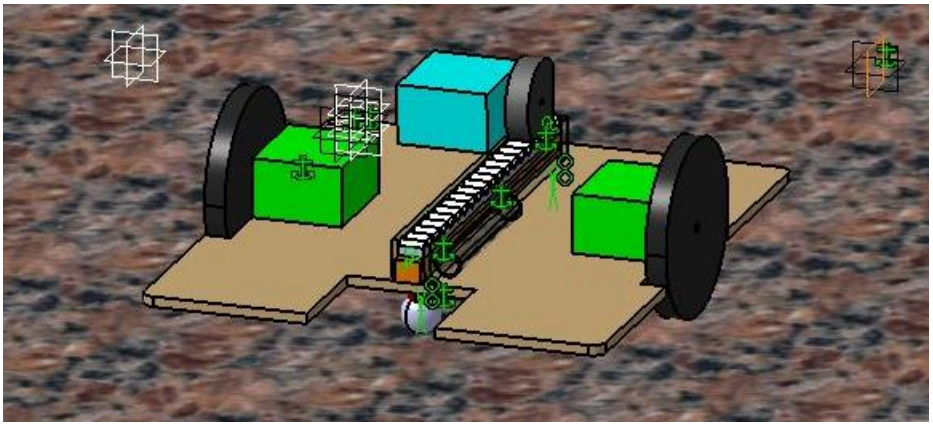

Figure 9- Abstraction of the Firing Mechanism

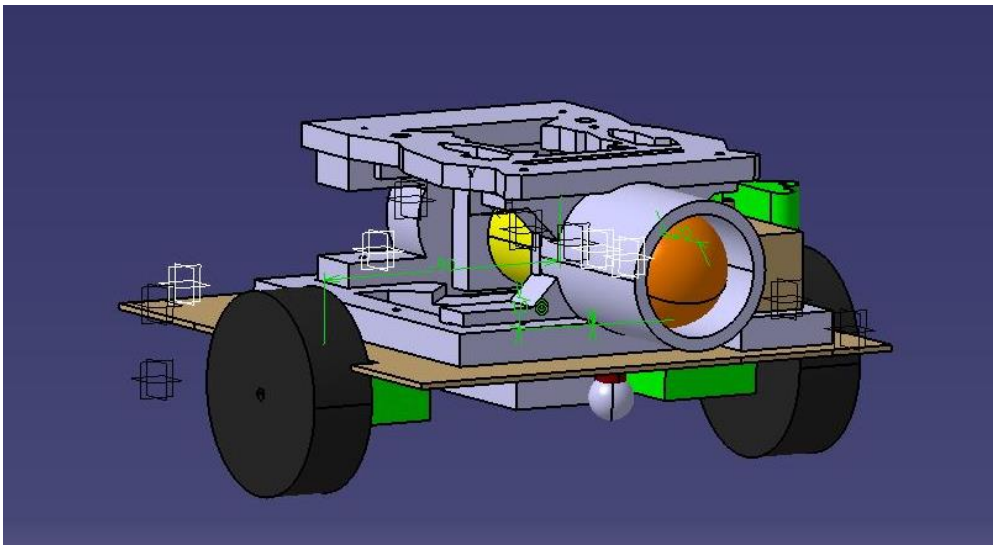

Figure 10- CAD Drawing Showing Redesigned Firing Mechanism

Assembled Firing Mechanism

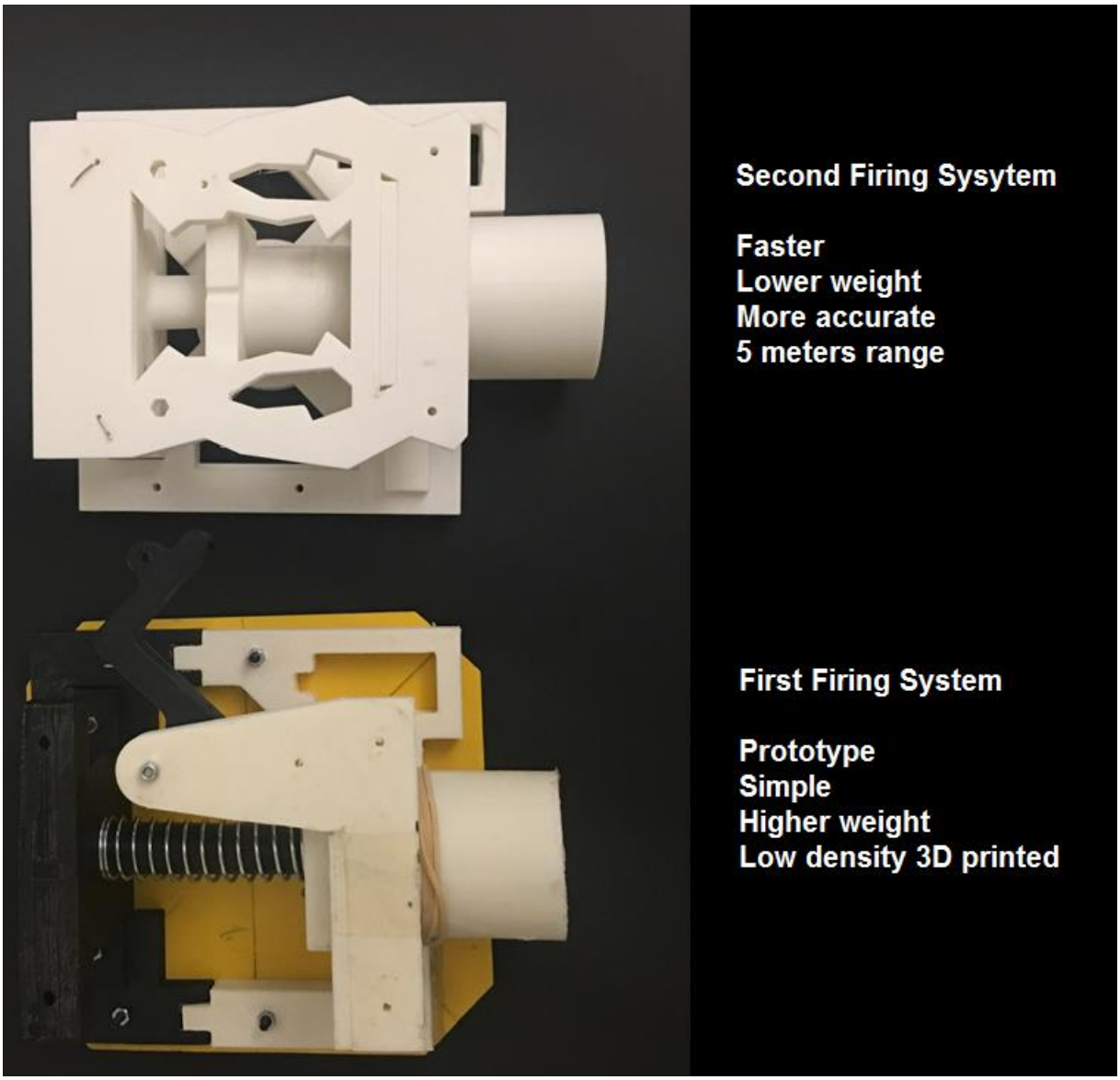

Figure 11- Redesigned and 3D Printed Firing Mechanism

Pictures of Assembled Robot

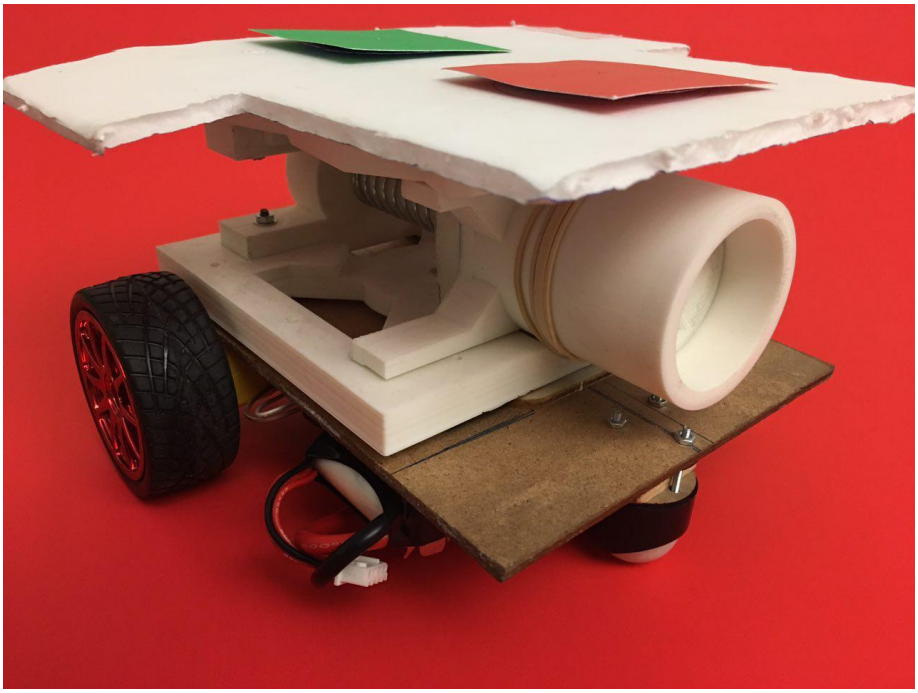

Figure 12- Side View of the Assembled Robot

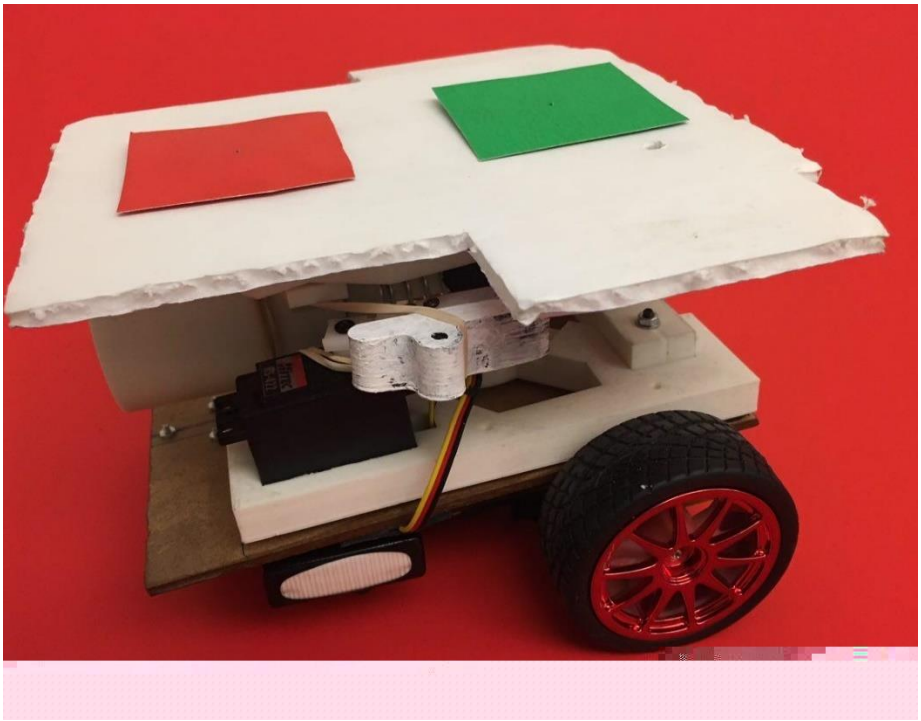

Figure 13 - Image showing the identification labels on top of the Robot

Our Testing Area

We had built a test area with conditions mimicking the actual operating environment of our robot. This space was similar in size to the real operating conditions.

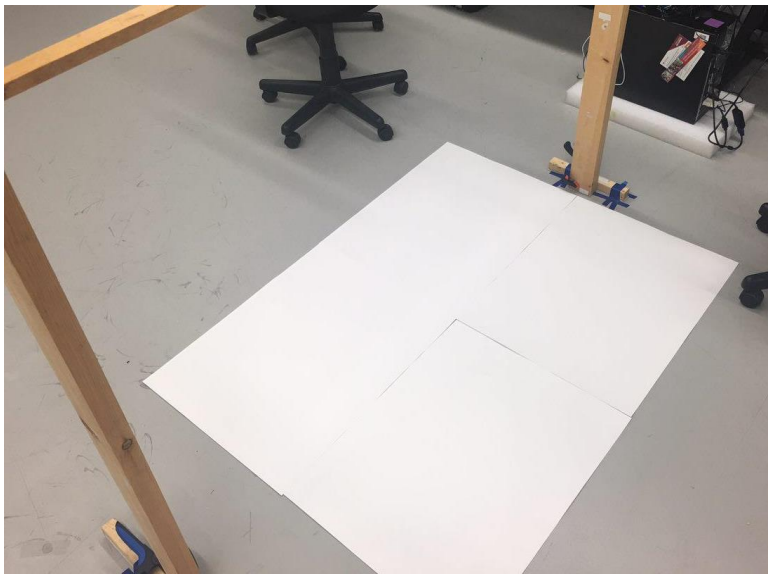

Figure 14- Our Testing and Troubleshooting Area

Geometry of Image Processing

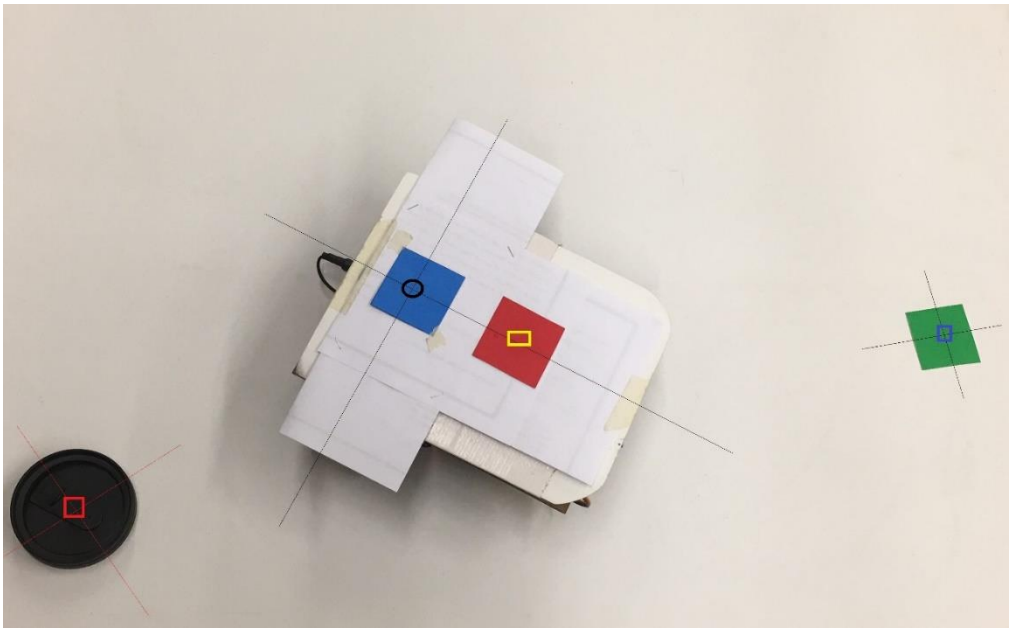

Figure 15 - Geometry of Targets to be Identified in Image Processing

Working Example of Program

Below is an image showing the output console of our C++ image processing software, We are ginven information on the centroids of our robot, the target and the osbstacle. Moreover, information about angles and distances is provided.

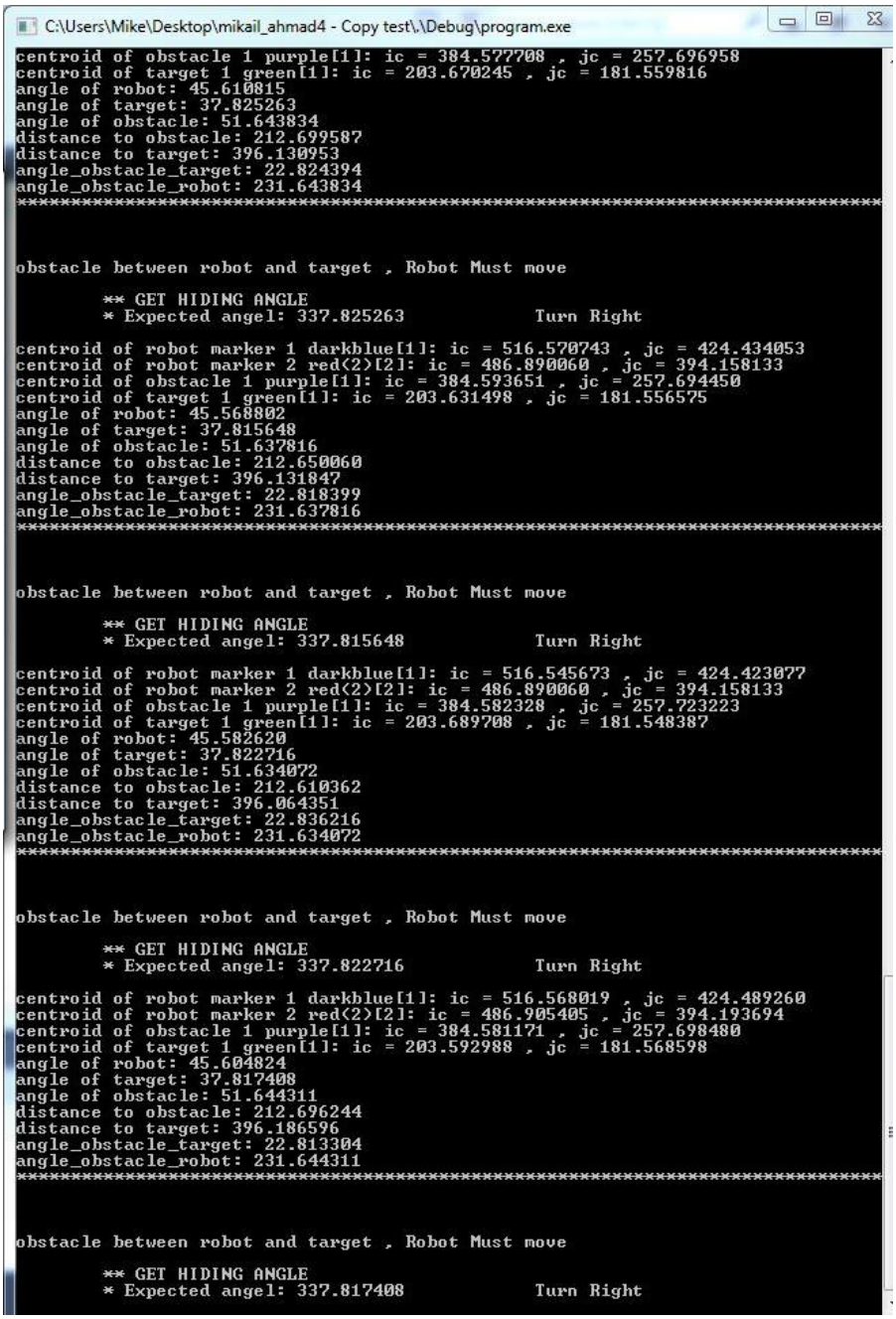

Figure 16- Console output of our Image Processing C++ Program

Using “image view.exe” allows us to have a visual understanding of the output of our image processing algorithm, we may see the identified and labeled objects, each with its respective centroid. It is this information that the algorithm uses in creating a trajectory for our robot and deciding on the appropriate moment to fire. However, the image view program does take up a lot of processing headroom, therefore for the most efficient and streamlined hardware execution, it would be best to run our code “in the dark”, so to speak. This allows the algorithm to control and direct the robot, without showing the user the intermittent steps of the image processing.

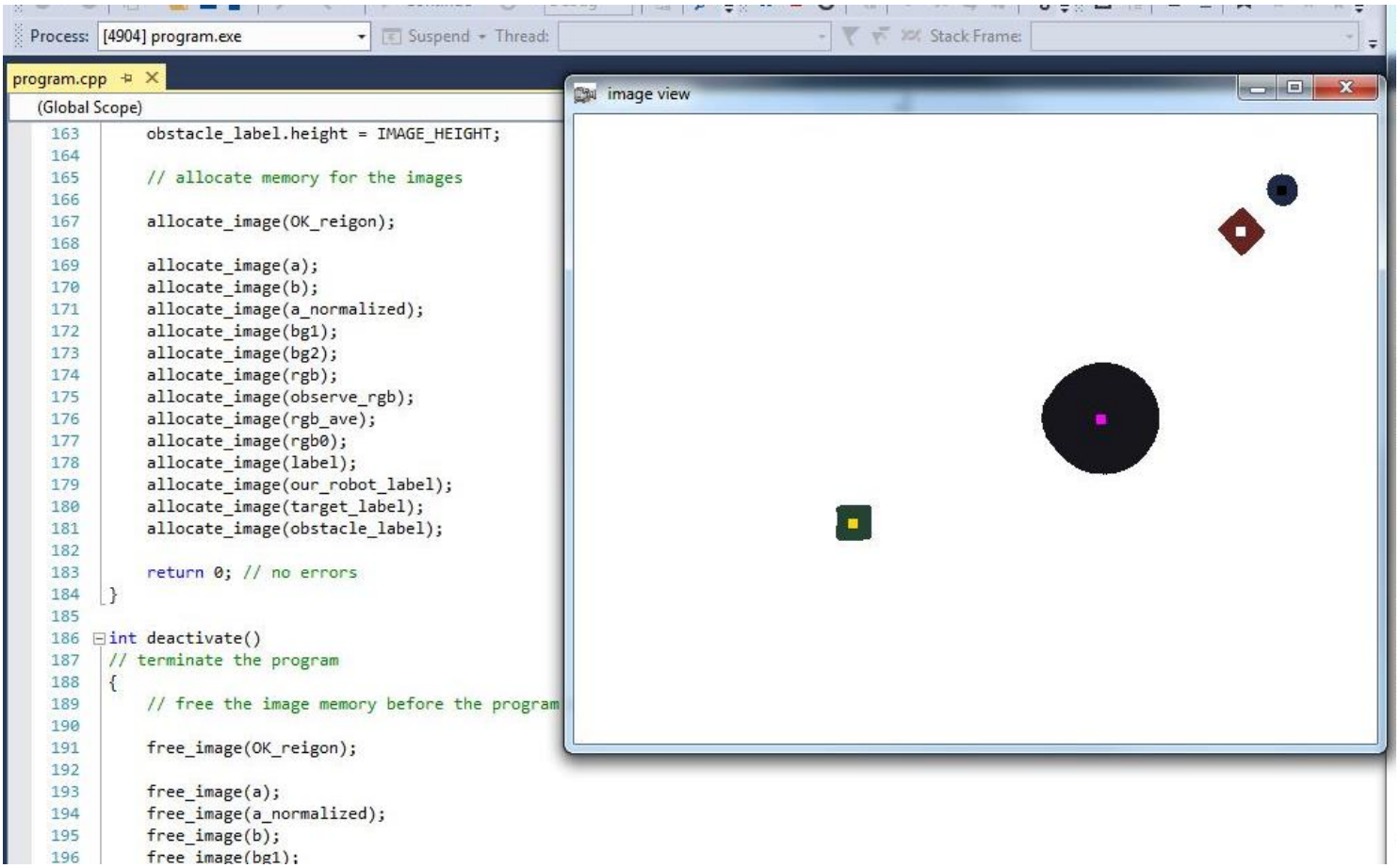

Figure 17- Image showing output of our C++ image processing algorithm. Centroid are drawn on objects of Interest

Conclusion

In this project a three wheel wirelessly controlled robot was designed. Xbee modules are used for wireless connection,wheels are driven by DC motors and DC Motor controller is used to change the direction of rotation and adjusting PWMs. Image processing program is coded in Visual studio c++ and an Arduino program is coded in IDE software to recive and execute the maneuvering commands. Inside C++ program colors of markers of the robot,obstacle and the opponent robot are identified and labels are specified. In the rest of the program by using special functions scaping, targetting and maneuvering of the robot are implemented.