Ryerson University
Department of Electrical, Computer & Biomedical Engineering
Faculty of Engineering & Architectural Science

Table Tennis Robot

EDP Project Report

Submission Date	April 17, 2020
Due Date	April 17, 2020

Name	**Student ID**	**Signature***	**Name**	**Student ID**	**Signature***
Tejinder Dhaliwal	500636899	T.D.	Henry Zou	500628304	Y.Z.
Scott Goddard	500723518	S.G	Gausshan Enthiraganthan	500688653	G.E.

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.*

Introduction

The process of fetching and collecting objects is observed in many tasks of everyday life. Streamlining and automating tasks like these using computers have been increasingly successful with the growth in computational performance. The automation of the collection of table tennis balls is a challenge that this paper explores. The Autonomous Table Tennis Robot is designed to capture ping pong balls using autonomous pathfinding and object detection from image processing. The project has a low resource approach to the solution by minimizing the costs of parts and computation. The robot sets a straight goal path for the ball but when encountering obstacles within its sensitivity, it stops, and calculates a trajectory to avoid it. The motivation of this project was to create an autonomous system that could help streamline the game of table tennis. This design is malleable to work for other applications such as collecting and mining coal, or even autonomous wheelchair guidance with the proper adjustments.

Table of Contents

Abstract

Image processing and autonomous pathfinding are two intersecting areas of technology for many important applications such as self-driving cars, or autonomous wheelchairs. This report shows how both technologies can be used together to guide a robot with collection capabilities to its goal of a ball whilst minimizing resources. The Autonomous Tennis Table Robot uses a Pi camera to scan for a ball, the located trajectory is then fed to an Arduino controller which monitors ultrasonic sensors for obstacles and controls the wheels to maneuver. Obstacle detection occurs through the Arduino controller. A benefit of having the Arduino control the sensors and motor functions of the robot is that it accounts for dynamic and static obstacle real-time collision control. This is due to the system's reactionary nature, there is no time delay of pushing the sensory information to the Raspberry Pi to make a decision on whether to stop. The communication in that direction between the systems only occurs during the stop-and-scan where the robot is stopped as protocol because of an obstacle in its bubble. After a stop and scan, histogram information on the environment in front of the robot is then passed onto the Raspberry Pi through a serial connection. The Pi calculates a rebound angle to avoid the obstacle, as well as provide direction to the Arduino controller on the next steps for movement of the robot. These directions and calculations are inspired by the Bubble Band obstacle avoidance algorithm. The separation of the two systems (Pi and Arduino) was helpful in independent development, testing, and to lighten the computational load by separating the image processing from the obstacle avoidance systems. The loop of heading straight in the direction of a ball until encountering an obstacle is repeated until the goal is no longer in sight. The capture mechanism consists of a vertical, cylindrical, rotating pillar which encapsulates the ball automatically as the robot drives over it. The design and implementation of this low-cost system is adaptable to other applications. Major sections of project development and implementation were affected due to the COVID-19 global pandemic.

Objectives

The objectives of the machine will be split into stages to ensure maximum efficiency and to modularize the final design. The planned objectives are objectives up to stage 5 and if there is extra time other stages may be added:

Stage 0: To buy the parts in the list from different retailers.

Stage 1: Using the parts, assemble a robot that can move around based upon commands from the raspberry pi.

Stage 2: Based upon the command of the raspberry pi, test the initial capture mechanisms so that it can actually capture the ball. The robot will be controlled by humans in this stage as it is just to make sure in isolation the capture mechanism will work.

Stage 3: Ball detection and moving toward the ball though the raspberry pi operating system. The commands will be based on the real-time operating systems that will be made and based upon the image data from the camera and the sensors.

Stage 4: Ball capture based upon the raspberry pi operating system. The ball capture mechanism will be tested and the movement algorithm will be adjusted so the ball will be able to end up inside the storage component.

Stage 5: Static obstacle avoidance. In the previous stages, the environment has no obstacles as to simplify testing but in this stage, simple obstacles will be added to the testing stage for example a cardboard box.

Stage 6: Pathfinding optimization. In previous stages, the machine would only get the first ball it detects. In this stage, the machine will model the path based upon the least movement needed and more advanced pathfinding algorithms.

Stage 7: Dynamic obstacle avoidance. The machine will be designed with moving obstacles into account.

Stage 8: Fault tolerance. The machine will be designed so that it can recover functionality if a component experiences a failure.

Background

Previous Autonomous Robots

The first electronic autonomous robot with complex behaviour was created by William Grey Walter in 1948. He created three-wheeled robots capable of finding their way to a recharging station when they ran low on battery power. The robots performed as expected, except when they were put in front of a mirror. Since then, the variety of tasks autonomous robots are able to perform has expanded.

The robot which inspired our project was the Tennibot[11]. It is an autonomous robot that detects a tennis ball and retrieves it. It travels around a tennis court and utilizes sensors and object detection to retrieve tennis balls. In terms of design, the Tennibot has two primary wheels which control movement, and a ball storage mechanism with 4 attached wheels. The robot captures tennis balls with two wheel-like objects which rotate to roll the ball into its storage unit. The pathfinding algorithm used by the existing Tennibot is unknown, but it can avoid both stationary and dynamic obstacles. Attached to the robot is a camera that assists with ball detection and determines the location goal for the robot. Based on the detection results, the robot moves toward the balls and picks them up until the bucket is full or the user instructs the Tennibot to stop collecting.

Although the Tennibot uses its camera for ball detection and retrieval, it is only autonomous with the help of a station. The station helps navigate the robot by indicating where on the tennis court it is. Without the station the robot can be controlled remotely, but this would defeat the purpose of its autonomous features.

Obstacle Avoidance Algorithms

The simplest type of obstacle avoidance algorithm is called “The Bug”. The algorithm directs the robot to circle around an encountered obstacle and find the point closest to the goal. From this point it travels towards the goal ensuring that it goes in the shortest path possible from its boundary. The algorithm is very slow, and only reckons the most recent sensor reading, which means sensor noise can be seriously detrimental to the efficiency of the robot. Shown below is how the bug algorithm works [10].

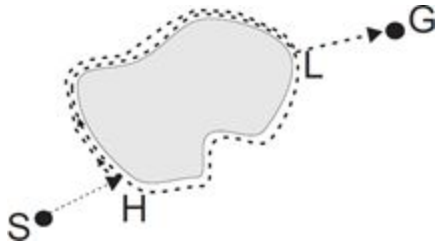

Figure 1: The bug algorithm [1]

The vector field histogram (VFH) algorithm overcomes the problem of sensor noise[1]. It does this by creating a polar histogram of a handful of recent sensor results. Unlike most obstacle avoidance algorithms, VFH considers the dynamics and shape of the robot. It works in real-time to motion plan with the help of a cartesian grid, which is set by the robot's proximity sensors. The range sensors compute the polar density histograms to pinpoint the obstacle's location and proximity. Based on the histograms, the robot routes itself to avoid the obstacles and follows its target path. This algorithm is ideal as it has minimal problems; however, it requires heavy computation which makes it difficult to implement in an embedded system. Shown below is an example of how the vector field histogram works [10].

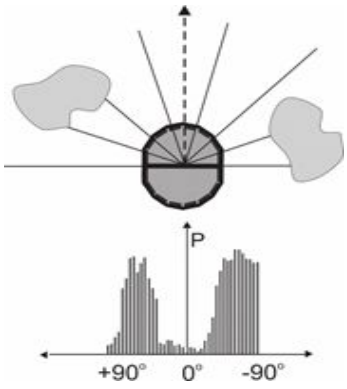

Figure 2: Vector Field Histogram[3]

Many different types of obstacle avoidance algorithms exist, however, few of them work in embedded systems and real-time. Algorithms that are easier to implement hinder performance due to sensor noise, and/or the inconsideration of the vehicle's dimensions. Other algorithms solve these problems however are harder to implement and require very strong CPU which may not fit the budget of the project. The algorithm modified and used in this project is the Bubble Band Algorithm which is further discussed in Theory

Theory & Design

Goal Detection

The goal detection algorithm was implemented using a camera. To detect the goal, the camera and OpenCV were used in conjunction to detect the goal. To get rid of the blur in the image, a Gaussian smoothing must be applied with the equation :

$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figure 3: Gaussian Smoothing [5]

The image is transformed from RGB color space into HSV color space. The image is then masked so that all colors that are not within the color value of the ball is masked out. Using this image, the contours of the image were found. The largest contour was used to draw a circle. If the radius of this circle is not of a certain size, it means the ball is not detected. To determine the distance the concept of the focal length is used to determine relative distance[6]. A measured reference distance is predetermined and a ratio between reference and camera distance is used to determine distance [2].

Modified Bubble Band Obstacle Avoidance Algorithm

The bubble band algorithm was implemented to avoid collisions with obstacles. This method incorporates pre-planning. In path planning, the vehicle follows a predetermined path which prevents excessive distension between bubbles of consecutive positions, thus drawing smoother trajectories. Essentially, the bubble band shows the greatest distance between the vehicle's position and surrounding obstacles in every direction. The shape and size of the bubble are determined by a simplified model of the robot's geometry, and by the range information provided by the sensors. The objective of the controller is to find the position where it will minimize this bubble tension. The primary advantage of this technique is that it takes into consideration the shape of the vehicle. A general disadvantage of this technique is that the area in which the vehicle will travel must be known prior to deployment. This is not a problem in our case because the area beforehand is known as it is usually set in a room. The bubble region is determined by the combination of the proximity sensors and the shape of the

robot. The bubble region would have been approximated with proper testing, but in the pseudo-code below its defined as the variable Ki.

Bubble Sensitivity Region

```
unsigned int ultrasonic_scan[N];
unsigned int bubble_boundary[N];
bubble_boundary[i]= Ki*V*t_interval
int check_for_obstacles(void)
{
for (i=0; i<N; i++)
        {
        if (ultrasonic_scan[i]<= bubble_boundary[i]
                return(1);
        else return(0);
        }
}
```

N is the number of ultrasonic sensors

V is the velocity at which the robot moves at

t_interval is the time interval between each evaluation made by the sensor data

Ki is a scaling constant, used for tuning

Without obstacles, the ultrasonic sensor can read up to 400 cm (max value)

How the Robot works

The autonomous robot at first will detect the direction of the goal. Using the direction of the goal, it will spin itself until the path towards the goal is a straight line. It will keep moving forward until an obstacle is detected in its way, and if an obstacle is found within the bubbles region it will divert in a direction with the lowest density

of obstacles. It will then continue travelling in this direction until the goal is visible again. Below a flowchart is displayed to demonstrate this process.

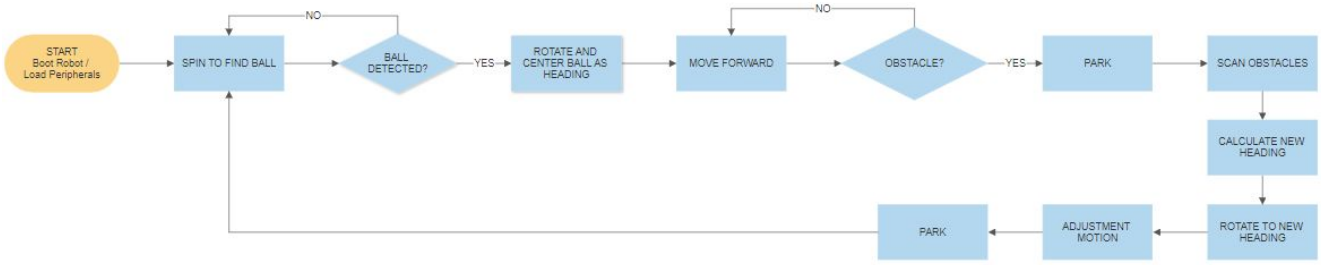

Figure 4: Autonomous Table Tennis Robot Flow Chart

This is the Flow Chart for the robot proposed in this project.

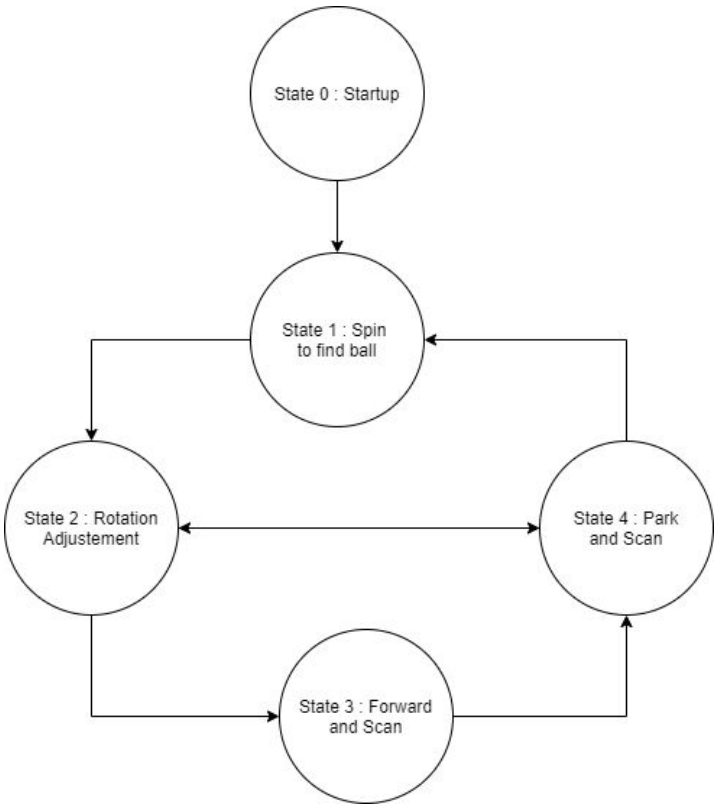

Figure 5: Finite State Machine of Robot Operation

Summary of the above States in Figure 5 :

State 1: Spin to find the ball

The Robot spins in place, while the Raspberry Pi Camera 'looks' for a ball. If a ball appears on camera, it will send the position of that ball on the screen, as well as a change to State 2 to the Arduino.

State 2: Rotation Adjustment

The goal of this state is to position the table tennis ball directly in front of the Robot and thus in the center of the PiCam's frame. This state is a feedback loop, where the robot will rotate based on the angle provided by the Raspberry Pi. After which point it will check to see if the Raspberry Pi has a new angle to rotate to (and thus the adjustment must be repeated) or if the new angle has satisfactorily positioned the ball in the center of the monitor in which case the Pi will instead send a State Change to State 3.

State 3: Forward and Scan

This state will cause the Robot to move forward while scanning a 180° region in front of it. This scan will return a distance array to the Raspberry Pi. When the robot detects an object within its sensitivity region it will move to State 4. Otherwise it will continue forward.

State 4: Park and Scan

This state will cause the robot to park, and then Scan. This scan will be used to determine the rebound angle. Then the robot will move to State 2, where it will adjust itself to face the new rebound angle.

Determining the Rebound Angle

In order to determine a new trajectory, the histogram (Figure 6.3) is formulated through Figure 6.1 to calculate a new heading known as the rebound angle. This is a weighted average shown in Figure 6.1, proposed by I. Susnea et al. in, "Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots," *Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots*. This was to be implemented after the integration of the two robot systems for proper fitting to the environment. Until the time being, the angle index with the least density of

objects was chosen. If there were two equal minima, the leftmost angle would be chosen as a protocol. This is a part of the design algorithm that can be changed and manipulated to fit and tailor to the application environment. The calculation and code for our preliminary maxima implementation are shown in Figure 18.

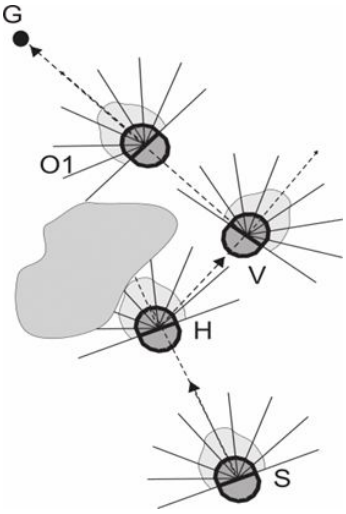

Figure 6: Rebound Response and Path[9]

$$\alpha_R = \frac{\sum_{i=-\frac{N}{2}}^{\frac{N}{2}} \alpha_i D_i}{\sum_{i=-\frac{N}{2}}^{\frac{N}{2}} D_i} \quad (3)$$

Figure 6.1 Calculating rebound angle

Where, $\alpha 0 = \pi/N$, $\alpha i = i\alpha 0$, $\alpha i \in [-N/2, N/2]$, **I=** the ultrasonic sensor sample index, **N=** total number of Ultrasonic sample cells, and **Di=** the distance value found by the sensors.

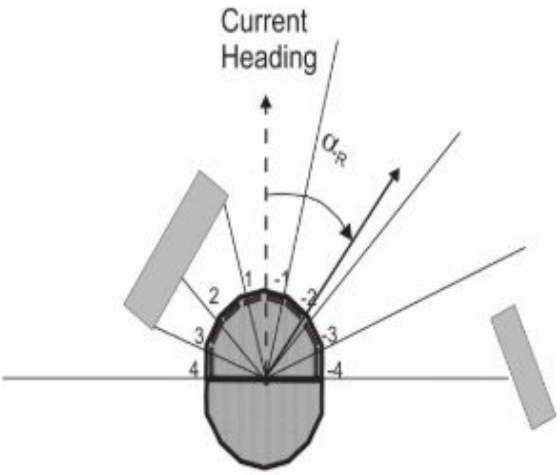

Figure 6.2 Determining Rebound angle

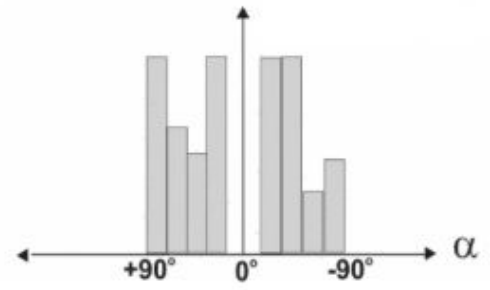

Figure 6.3 Histogram of Distance Values

Physical Design

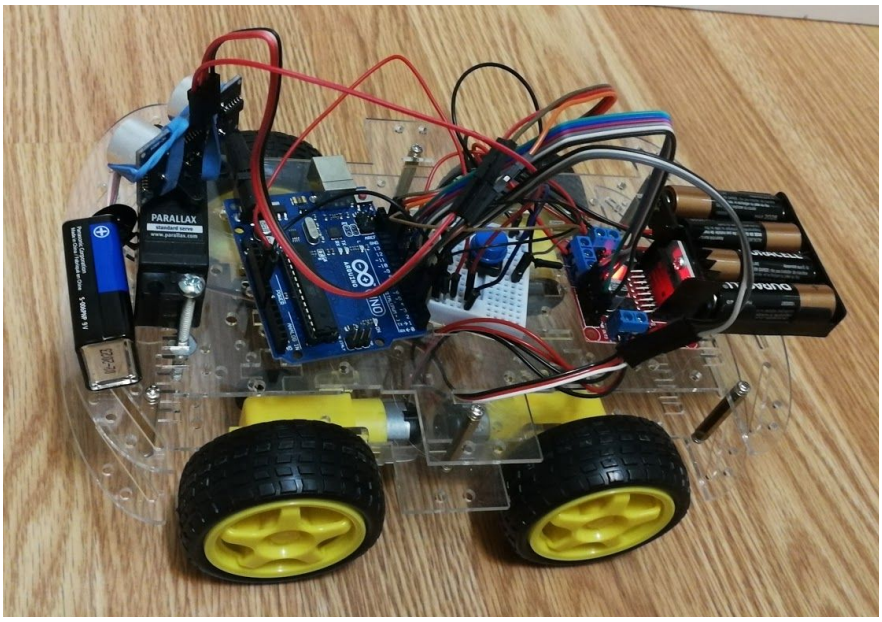

Figure 7: Robot Design

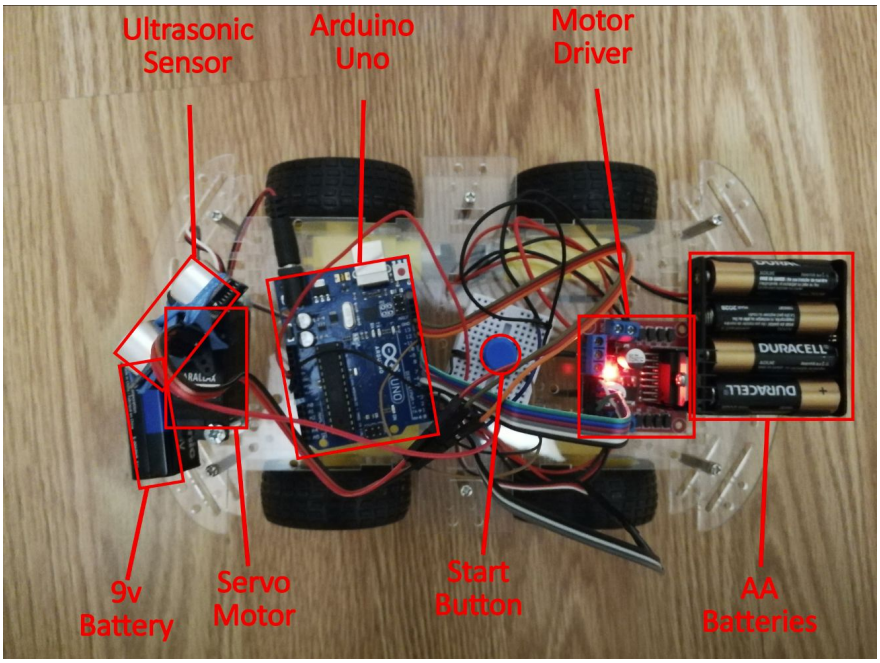

Figure 8: Robot parts Layout

The final design of the robot can be seen above in Figure 7. Unfortunately additional 3D printed parts were planned to improve both the aesthetic and the organization of the Robot, however, they were unable to be completed due to lack of access to the printlab. The specific components of this setup are identified in Figure 8.

Circuit Design

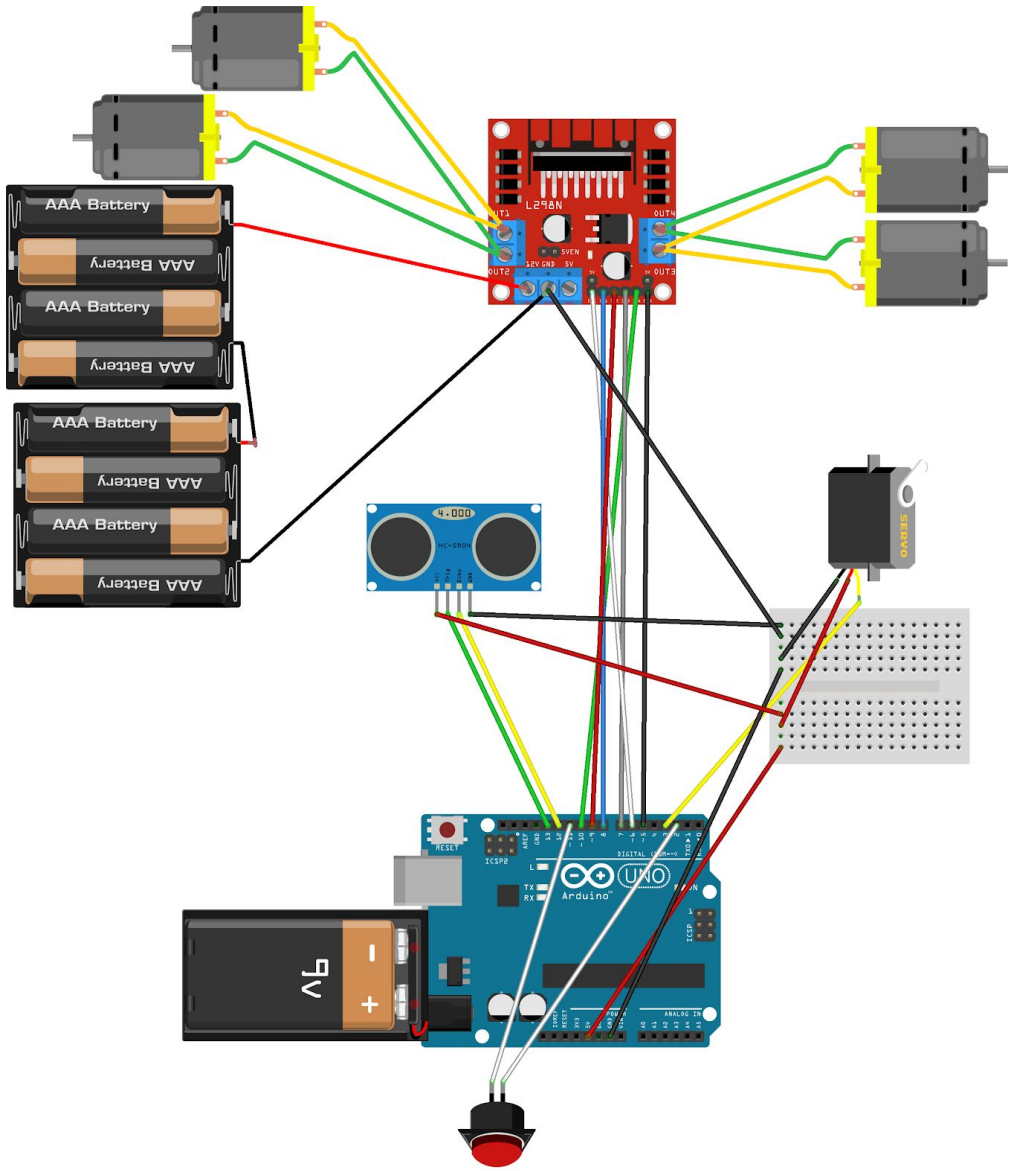

Figure 9: Circuit Layout (Created using Fritzing)

The circuit layout for the Arduino can be seen above in Figure 9. The Arduino itself is powered by a single 9v battery, while the wheels are powered by 8 AA batteries. 8 AA batteries provide 12v of power, which is ideal for the L1289 Dual Bridge Driver to power the motors. It was noted that with lower voltages we were unable to get the consistent speed with the motors, frequently having one or more motors stall. The breadboard only has 2 lines used, a 5V supply from the Arduino and a common ground. The pinout for the Arduino board can also be seen. These pins are set in the Arduino code found in the Appendix.

Alternative Designs

From the initial brainstorming to the final design, there were changes made in regards to features, parts, and architecture of systems. This section compares viable different options considered for the different aspects of the design.

In regards to parts, they are limited by budget and requirements. For image processing, obstacle avoidance, path determination, and robot control to all be burdened on one controller is very resource demanding. Thus, a Nvidia Jetson Nano was considered for the computational load. The Jetson Nano costs $149.95. This is a high price point when considering the purpose and preliminary design of the robot. Thus, we chose the Raspberry Pi 4 which comes at a price point of $89. The computational burden on the Raspberry Pi and its power demands led us to make the project be composed of two systems; one to control the robot as well as receive sensory information, and one to do the calculations/image processing. The Arduino microcontroller was chosen to run the robot system. If the Jetson Nano was chosen, the image processing will be a lot faster but with the computation power comes at a price which is to increase the power needed. The raspberry pi needs 2.5 A and the Jetson Nano uses 4A. A new power bank will be needed to be bought if the Nano is bought, power issues will be introduced.

An alternative design used other more advanced image processing software such as tensor flow, Yolo Real-Time Object Detection. However, these are much more computation heavy software and were not feasible to be run on a raspberry pi as even OpenCV did not have high frames per second and the computation time is an order of magnitudes less than the other software as they use neural networks. Also, the design did not need it as goal detection was limited to just a ping pong ball and the usage of the software is to detect what an object is which may be useful in other designs where there are multiple different goals than the singular goal of a ball[7].

Prototype

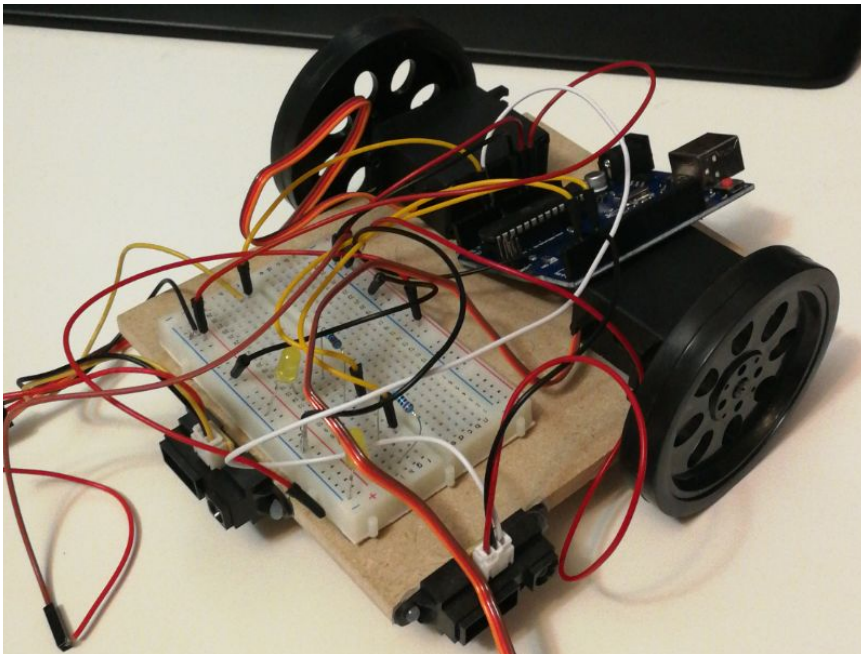

Figure 10: Robot Prototype

The initial prototype for the design can be seen above in Figure 10. This used IR sensors, which we found to be inadequate as their accuracy was too low, especially at shorter and longer distances. We upgraded to an ultrasonic sensor for the final design as it was much more accurate. Also rather than using a number of fixed position sensors, we opted to use a single rotating sensor.

Ball Capture Mechanism

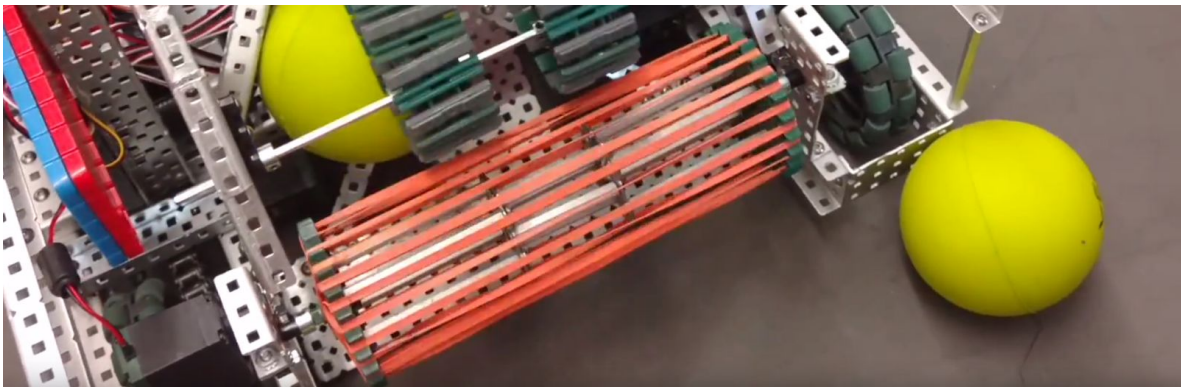

Figure 11: Ball Capture Mechanism

As the primary objective of the robot was to collect table tennis balls, we decided to use a ball capture mechanism like the one above in Figure 11, however, we were unable to implement it as the 3D print labs at the school were unavailable.

Material/Component Cost

Table 1: Component Costs

Component	**Cost(Canadian Dollar)**
Arduino Uno	27.99
RASPBERRY PI CAMERA V2 - 8MP	39.00
Raspberry Pi 4 Computer Model B 4GB	89.00
SanDisk Ultra 32GB microSDHC UHS-I Card with Adapter - 98MB/s U1 A1 - SDSQUAR-032G-GN6MA	11.56
L298 DUAL MOTOR DRIVER - 2A	14.00
HC-SR04 ULTRASONIC SENSOR	4.99
Parallax Standard Servo	14.50
4wd Robot Platform	38.00
Total	**239.04**

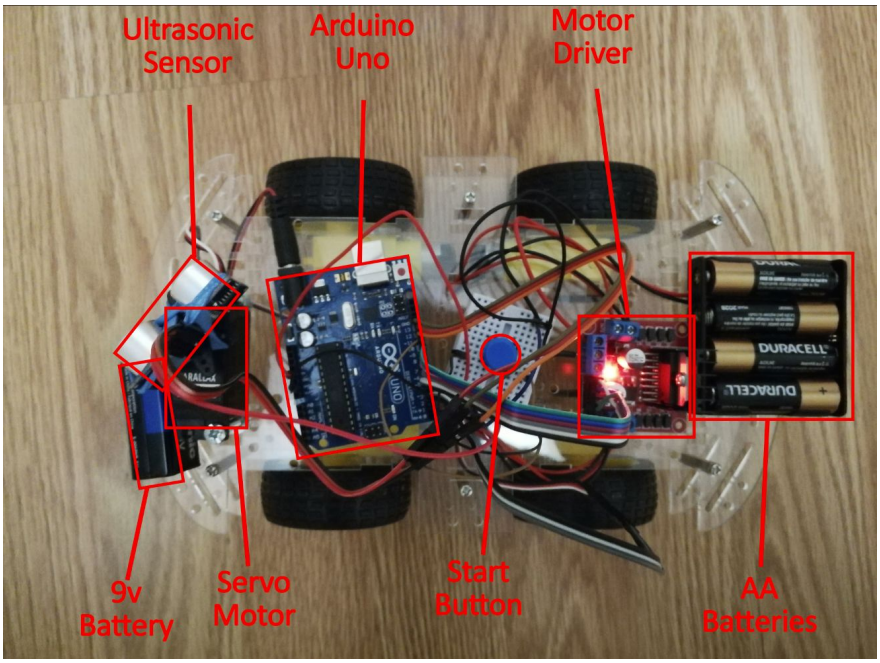

Figure 12: Layout of Components on Robot

Not Included in the above cost listing in Table 1 were also the following:

- AA Batteries and Holder
- 9V Batteries and connector
- Push button
- Breadboard
- Wires
- Prototype materials

These were neglected as they are relatively nominal compared to other parts used, and their specifications are not relevant. They can however be seen above in Figure 12.

Measurement & Testing Procedures

Goal Detection:

Before the goal detection was tested, the ball measurements were taken. The color space of the ball was obtained by taking an image of the ball with the camera. Using the image of the ball, obtain the color value of the ball by extracting two-pixel values and changing the value into HSV. Using the two-pixel values, a mask of the image was created and compared to the original image. The color range value was readjusted if any part of the ball was not included by getting the pixel color value of the position and readjusting the mask color range to include the pixel value.

For testing the goal detection, a python program running OpenCV was used to test. The program would take the video frame by frame and apply the goal detection algorithm. The detection algorithm was manually adjusted for the best model for the ball detection radius.

Obstacles Avoidance:

The servo was set up with the ultrasonic sensor and Arduino. The ultrasonic sensors sampled at 20-degree intervals. A basic bug algorithm was implemented with plans to readjust so to create the bubble algorithm. The basic bug algorithm picks the left obstacle-free angle to reconstruct.

Voltage variance and power consumption can be further optimized. When conducting field tests, the robot was noticed to drain battery sources rapidly. At this stage of the development, a mass-produced prototype would not yield reasonable customer satisfaction.

Measurement Results

Goal Detection:

The model detection for the ball was successfully implemented as shown in figure 13. However, the model detection will only work based on no large object that has the same color as the ball. The model is not able to encase the ball if the ball is at the edge of the frame or if an obstacle blocks the full view of the ball. The model however will encase the largest continuous section of the ball in a circle if it is greater than 10-pixel length. If the ball is still detected but with the wrong encasing, it is not a problem as the direction is correct.

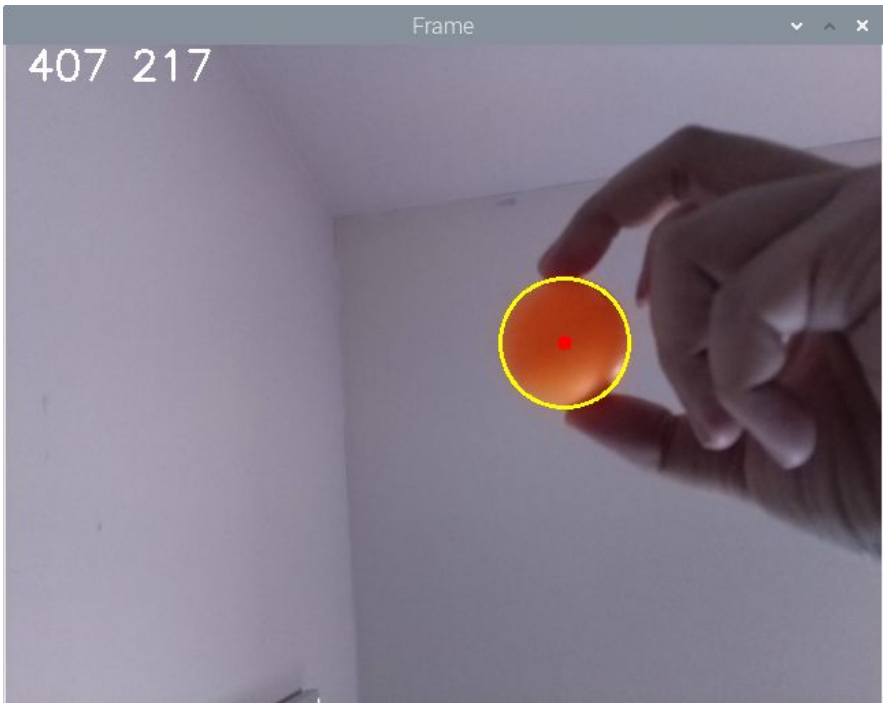

Figure 13: Ball Detection Successful

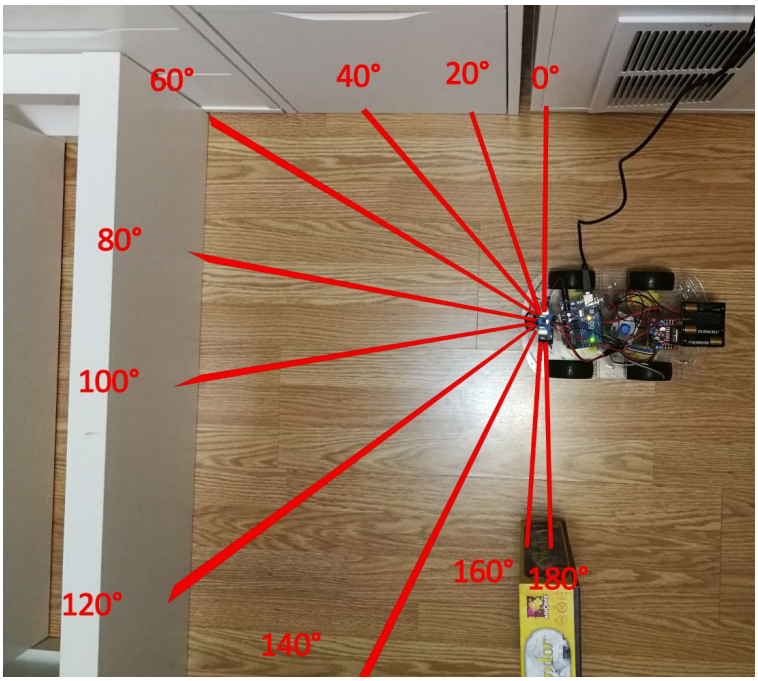

Figure 14: Sensor Angles Visualized

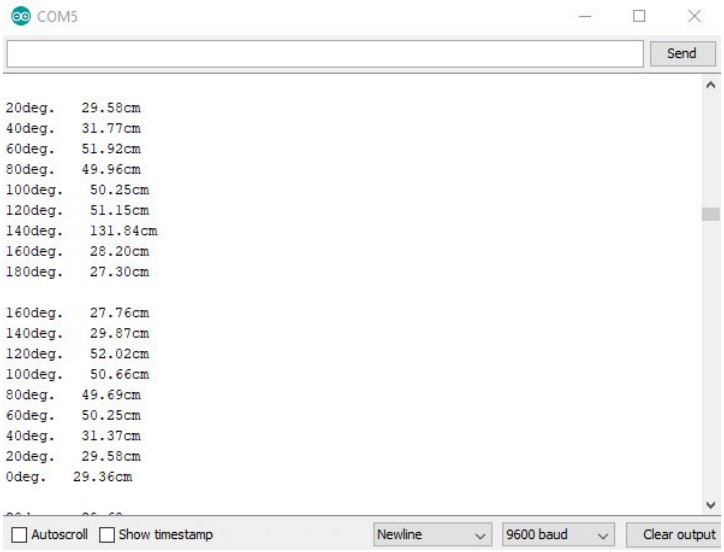

Figure 15: Sensor Angle Output

```
int distping(){ //Checks the distace at the current position
  digitalWrite(sensortrigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensortrigpin, LOW);
  duration = pulseIn(sensorechopin, HIGH);
  distance = duration*0.034/2.00; //based on speed of sound = 0.034us/cm
  // divided by 2 because the sound has to go to the object and back
  Serial.print(myservo.read());
  Serial.print("deg.   ");
  Serial.print(distance);
  Serial.println("cm");
  return distance;
}
```

Figure 16: Distance Checking Function

```
void scan(){
  for (int j = 0; j <steps; j++) {
    i+=inc;
    myservo.write(i);
    delay(100);
    distances[j] = distping();
  }
  inc = -inc;
  Serial.println(" ");

  //Send DISTANCES array to Rasperry Pi for new directional bearing
  // NOT IMPLEMENTED
}
```

Figure 17: Scanning function

A histogram (Figure 6.3) was generated based on the Ultrasonic Distance Sensor mounted to the servo motor. This was controlled by the Arduino. The relevant Arduino functions distping() and scan() can be seen above in Figures 16 and 17 respectively. Scan runs a distance ping across angles which increment by 20 degrees each time. We can see the results of such a histogram being output above in Figure 15. A visualization of these angles can be seen in Figure 14 above. These distance values were found to be accurate to within 5cm on average, which for our purposes was more than adequate. The basic bug algorithm was tested with sample distance and the greatest magnitude of the distance was picked. If multiple distances with max magnitude are detected, the leftmost angle was used.

```
def rebounddegree (vector):
    # array of degree which match the magntiude of the degrees
    degree = [0,20,40,60,80,100,120,140,160,180]
    a = max(vector)
    #list of candiate indices iterate though list compare to max value of list
    nonblockedindices = [i for i,j in enumerate(vector) if j == a]

    nonblocked = nonblockedindices[0]
    offset = 0;
    #offset will be based on the machine model
    #right now picks left most degree with sensor data with the maximum distance away
    rebound = degree[nonblocked] + offset
```

Figure 18: Rebound Angle Function

Post Analysis

The robot design components were successful. The components were able to accomplish their task. During the design process, a major obstacle when using the raspberry pi was the power bank. As when the power bank runs out of power, the power bank will need to be recharged before continuing. To migrate this issue, the official power supply should have been bought. The only problem with this is the official power supply is a wall-mounted plug so after the development phase, a power bank with sufficient power will need to be chosen. The official power supply was not brought to save on development costs. A concern is the power consumption of the entire robot as some components such as the raspberry pi are running low on power. A new power source will be needed when the components are put together. For the obstacle detection algorithm sweeps were taken at 20-degree intervals, if the obstacle size is smaller than the 20-degree interval then the obstacle will not be able to be detected. To improve scan, smaller degrees can be sampled such as 5 or 10 degrees however this will significantly increase a single 180 sweep time as the sample will need to wait the ultrasonic time overhead. The algorithm is stitching together the surroundings using the sweep. Depending on the speed of the obstacle, if the obstacle is counted in multiple scanning the obstacle avoidance will fail. Faster speed obstacles will require higher sweeping periods. To vastly improve this problem, better hardware should be used. The 4 dollars ultrasonic sensor has very limited functions as it can only scan in front of the sensor. For extremely fast-moving obstacles, better hardware such as lidar will be needed as lidar can complete the single sweep in a matter of microseconds. However, lidar cost will be in the range of 100 to thousands so many multiple times more than the 4 dollar sensor.

Conclusions

In conclusion, this project was successful in connecting learning concepts from the classroom to real-world applications. The scope of the project was defined and research on the chosen topic was conducted using techniques and resources taught in the lecture hall or provided by the professor. The design process was mapped out and a preliminary acquisition of parts was conducted to compose prototype one. Prototype two was built and tested based on hardcoded controller instructions. The image processing and goal detection were developed successfully using the Raspberry Pi and OpenCV to output a heading and instructions for the Arduino controller. Object avoidance was implemented on both systems; the sensor detection, recording and scanning on the Arduino, and the rebound angle calculation and output on the Raspberry Pi. The two system implementation and approach can be beneficial in real-world situations where the consequences of collisions can be detrimental in a setting with dynamic obstacles where the time/resources used to calculate and process the stoppage is precious.

The COVID-19 global pandemic interrupted the project, disabling the ability for the team to interface both systems and further test or modify the build. It also prohibited the use of Ryerson Universities' 3D printing machines which were to be used to build the ball capture mechanism. Nevertheless, this project has been excellent in connecting learned concepts from the classroom to real-world applications. It has been valuable for teaching leadership and teamwork skills. When faced with the inability to meet in-person throughout the duration of the project, it was found to be helpful in demonstrating how to organize and develop a project as a group, without sitting together at a single computer. This is more accurate to a real-world situation where parts of a project would have to be developed by individual developers and then combined at a later date. Throughout the project we were able to independently develop separate parts of the program, while also being able to rely on each other to solve individual problems as they arose. Resolving the integration of the two systems with the addition of the ball capturing mechanism, the robot would be fully functional.

References

[1] V. A. Bhavesh, “Comparison of Various Obstacle Avoidance Algorithms,” *International Journal of Engineering Research and*, vol. V4, no. 12, 2015.

[2] A. Rosebrock, “Find distance from camera to object using Python and OpenCV,” *PyImageSearch*, 05-Dec-2018. [Online]. Available: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/. [Accessed: 12-Dec-2019].

[3] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[4] H. Choset, A. Mills-Tettey, and V. Lee-Shue Jr, “Robotic Motion Planning: A* and D* Search,” *Carnegie Mellon University*. [Online]. Available: https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf. [Accessed: 13-Dec-2019].

[5] “Gaussian Smoothing,” *Spatial Filters - Gaussian Smoothing*. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. [Accessed: 17-Apr-2020].

[6] J. Harmer, “Understand Focal Length in 5 Minutes,” *Improve Photography*. [Online]. Available: 89https://improvephotography.com/34730/focal-length/. [Accessed: 13-Dec-2019].

[7] A. K. Jain, “Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino,” *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018.

[8] “OpenCV,” *OpenCV*, 14-Oct-2019. [Online]. Available: https://opencv.org/. [Accessed: 14-Dec-2019].

[9] I. Susnea, A. Filipescu, G. Vasiliu, G. Coman, and A. Radaschin, “The Bubble Rebound Obstacle Avoidance Algorithm for Mobile Robots,” *IEEE Xplore*, 11-Jun-2010. [Online]. Available: https://ieeexplore-ieee-org.ezproxy.lib.ryerson.ca/. [Accessed: 17-Apr-2020].

[10] I. Susnea, V. Minzu, and G. Vasiliu, “Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots,” *Simple, Real-Time Obstacle Avoidance Algorithm for Mobile Robots*, Jan. 2009.

[11] R. Vatti, M. Patwardhan, V. D. Pawar, K. Belsare, S. Badgujar, A. Kangare, and S. Pawar, “An Autonomous Tennis Ball Picking Robot,” *International Journal of Computational Engineering Research (IJCER)*, vol. 7, no. 7, Jul. 2017.

Appendices

Arduino Code

```
#include <Servo.h>
#include <LiquidCrystal.h>

int leftspeedpin = 5;
int rightspeedpin = 6;
int leftonpin = 10;
int leftoffpin = 7;
int rightonpin = 9;
int rightoffpin =8;
int frontmotorpin = 4;

int sensortrigpin = 13;
int sensorechopin = 12;

int startout = 11;
int startin = 2;

int servopin  = 3;
int i = 0;
int inc = 20; // amount in degrees the sensor will move for each sample
int steps = 180/inc;
int distances[10]; //array to hold the distances. PASSED TO PI

double duration;
double distance;
int button;

int startspeed = 220;
int slowspeed = 220;
int speedchange = 20;
int leftspeed = startspeed;
```

```
int rightspeed = startspeed;

int state = 0;  // holds the state
int c = 0;  // arbitrary counter
int angle = 3000; // angle of ball position, determines how long the robot spins for
// this angle is received from the raspberri pi

Servo myservo;

// *0.034/2
// ANALOG WRITE 0-255 FOR PWM
void setup() {

  Serial.begin(9600);
  //Wheel Control pins
  pinMode(leftspeedpin, OUTPUT);
  pinMode(rightspeedpin, OUTPUT);
  pinMode(leftonpin, OUTPUT);
  pinMode(rightonpin, OUTPUT);
  pinMode(leftoffpin, OUTPUT);
  pinMode(rightoffpin, OUTPUT);
  //Servo Pin
  pinMode(frontmotorpin, OUTPUT);
  //Ultrasonic Sensor Pins
  pinMode(sensortrigpin, OUTPUT);
  pinMode(sensorechopin, INPUT);
  //Start Button Pins
  pinMode(startout, OUTPUT);
  pinMode(startin, INPUT);

  myservo.attach(servopin);
  myservo.write(90);
  //Ultrasonic Sensor Pin
  digitalWrite(sensortrigpin,LOW);
  // Start Button Pin
  digitalWrite(startout, HIGH);
```

```
  //Wheel Speed Control Pins
  analogWrite(leftspeedpin,leftspeed);
  analogWrite(rightspeedpin,rightspeed);
  //Wheel directional control pins
  digitalWrite(leftonpin,LOW);
  digitalWrite(leftoffpin,LOW);
  digitalWrite(rightonpin,LOW);
  digitalWrite(rightoffpin,LOW);

}

void loop() {
  if (state == 0){ // state 0, unstarted (Manually start via button)
    for (int i = 0; i <= 10; i++) {  // just some looping to make sure the button gets
filtered a bit so we dont start without input
      delay(10);
    button = digitalRead(startin);
    if (button == 1){
       c++;
      }
    }
  if (c >= 5) { // Makes sure button was held down for at least 50ms
    c=0;
      state = 2; // Go to rotational state 2
    }
    resetspeed();
    state = 3;
  }

  if (state==1) { // main state, runs scan continuously
    //this state also allows the vehicle to move based on other states, and keeps the
front scanning
    scan();
      // Get State from Pi HERE
    // NOT IMPLEMENTED
  }
```

```
  if (state == 2){ //state 2, spin on spot. exit from this state controlled by PI
    // The objective of this state is to reorient the direction to that which the ping
pong ball is in

  leftspeed = slowspeed;
  rightspeed = slowspeed;
  if (angle>0){
    right();
  } else {
    left();
  }
  delay(10*angle); // Rotation Duration proportional to the size of the angle this should
approximate
  // how long the rotation should be, then we will re-run this loop and try again if it
is not close enough
  // Until the raspberry pi exits this loop
  park();

  // Get State from Pi HERE
  // NOT IMPLEMENTED

  }

  if (state==3) {
    // Forward and Scan
    forward();
    state = 1;
  }
    if (state==4) {
    // Park and Scan
    park();
    state = 1;
  }
```

```
}

void scan(){
  for (int j = 0; j <steps; j++) {
    i+=inc;
    myservo.write(i);
    delay(100);
    distances[j] = distping();
  }
  inc = -inc;
  Serial.println(" ");

  //Send DISTANCES array to Rasperry Pi for new directional bearing
  // NOT IMPLEMENTED
}

int distping(){ //Checks the distace at the current position
  digitalWrite(sensortrigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensortrigpin, LOW);
  duration = pulseIn(sensorechopin, HIGH);
  distance = duration*0.034/2.00; //based on speed of sound = 0.034us/cm
  // divided by 2 because the sound has to go to the object and back
  Serial.print(myservo.read());
  Serial.print("deg.   ");
  Serial.print(distance);
  Serial.println("cm");
  return distance;
}

void resetspeed(){  // sets both left and right speed to their initial values
  leftspeed = startspeed;
  rightspeed = startspeed;
}

void turnaround(){  // turn 180degress in place (unused)
```

```
  leftspeed = startspeed;
  rightspeed = startspeed;
  analogWrite(leftspeedpin,leftspeed);
  analogWrite(rightspeedpin,rightspeed);
  reverse();
  delay(2000);
  left();
  delay(1500);
}

void reverse(){  //move backwards (unused)
  digitalWrite(leftonpin,HIGH);
  digitalWrite(leftoffpin,LOW);
  digitalWrite(rightonpin,HIGH);
  digitalWrite(rightoffpin,LOW);
}

void left(){  // rotate left in place
  digitalWrite(leftonpin,HIGH);
  digitalWrite(leftoffpin,LOW);
  digitalWrite(rightonpin,LOW);
  digitalWrite(rightoffpin,HIGH);
}

void right(){  // rotate right in place
  digitalWrite(leftonpin,LOW);
  digitalWrite(leftoffpin,HIGH);
  digitalWrite(rightonpin,HIGH);
  digitalWrite(rightoffpin,LOW);
}

void forward(){ // move forward
  digitalWrite(leftonpin,LOW);
  digitalWrite(leftoffpin,HIGH);
  digitalWrite(rightonpin,LOW);
  digitalWrite(rightoffpin,HIGH);
```

```
}

void park(){  // stop all motion
  digitalWrite(leftonpin,LOW);
  digitalWrite(leftoffpin,LOW);
  digitalWrite(rightonpin,LOW);
  digitalWrite(rightoffpin,LOW);
  resetspeed();
}
```

Raspberry Pi Code

Python Code for opencv window which shows the center point

```
from imutils.video import VideoStream

import cv2
import imutils
import time
import functools
import math
def main():

    #using the hsv color space define the balls minimium and maximum color space values ie what shade of orange
    ballLower = (5, 100, 100)
    ballUpper = (15, 255, 255)
    #preset center of the ball to -1,-1 and radius to -1
    point=(-1,-1)
    radius = -1
    vs = VideoStream(src=0).start()

    # allow the camera to beginning
    time.sleep(2.0)
    # keep looping
```

```
while True:
    # grab the current frame
    frame = vs.read()
   #grab the size of shape
    height, width, channel = frame.shape



    # resize the frame, blur it, and convert it to the HSV
    # color space

    blurred = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
    # construct a mask for the color
    mask = cv2.inRange(hsv, ballLower, ballUpper)
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)
    # find contours in the mask
    region= cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    region= imutils.grab_contours(region)
    center = None
    # at least one contour was found
    if len(region) > 0:

        #find the largest contour
        c = max(region, key=cv2.contourArea)
        #use largest contour draw a smallest circle to enclose that contour
        ((x, y), radius) = cv2.minEnclosingCircle(c)

        pt = (x, y)

        # only proceed if the radius meets a minimum size
        if radius > 10:
            # draw the circle and centroid on the frame,
            cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)
```

```
            cv2.circle(frame, (int(x), int(y)), 5, (0, 0, 255), -1)
        #if radius is smaller than 10 object not found
        else:
            pt = (-1,-1)
                #if no contours then object not found
    else:
        pt = (-1, -1)
    #change the pt from list to string
    text = (' '.join(map(str, pt)))


    #write coordinates of x,y onto the image
    cv2.putText(frame, text, (15,25), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255), 2,0)

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    # if the 'q' key is pressed, stop the loop
    if key == ord("q"):
        break


# close all windows
cv2.destroyAllWindows()
def anglemm (pts, rad, width, height):
    referenceradiuspix = 229.358442
    referencedistanceradius = 45
    referencedistance = 212
    ratio =  referencedistance /referencedistanceradius
    newdistance = ratio * rad
    oldpixel =  referenceradiuspix /referencedistanceradius
    newpixel = oldpixel * rad

    middle = width / 2
    (x,y) = pts
```

```
    distancex = ((x-middle)* newpixel )

    theta = math.degrees(math.atan(distancex/newdistance))
    return  str(theta)
l = oldpixel * rad

    middle = width / 2
    (x,y) = pts
    distancex = ((x-middle)* newpixel )

    theta = math.degrees(math.atan(distancex/newdistance))
    return  str(theta)
```