PongBot Final Report

A Butka Industries Design
Luis Osegueda, Rachel Conway, Dan Lane

I. INTRODUCTION

PongBot is a robotic device designed for the purpose to outperform other human competitors at the sport of beer pong. The official rules will be referenced marginally within this document, but can be found here. The primary objective is to throw ping pong balls from one end of an eight-foot (approximated to 2.4m) table towards 6-10 solo cups arranged at the opposite end. “Scoring” a ball within a cup causes the opponent to remove said cup from the table, and once all of one team's cups are removed, the opposing team has won. At a conceptual level, launching a ping pong ball at a consistent trajectory into the air and hitting a 4.75 inch diameter target with a high level of accuracy and consistency seems like a trivial problem to solve, however in practice is much more difficult.

II. DIVISION OF RESPONSIBILITIES

- Dan was responsible for the mechanical properties of PongBot but received assistance on the mechanical side from Rachel with initial value calculation for spring length and trajectory calculation, as well as from Luis who assisted in printing the majority pan/tilt assembly as well as the table clamp brackets.
- Luis was responsible for the electrical systems of Pongbot but received assistance from Dan on validating the short-circuit protections of the power supply as well as soldering cables to the DC motor as well as the buttons. Luis was also responsible for the

Arduino code but received major assistance from both Dan and Rachel with regards to debugging, troubleshooting, and ultimately getting the code operational.

- Rachel was responsible for PongBot's computer vision but received guidance from Dan with regards to OpenCV implementation and threshold/erosion filters. Luis also helped with serial communication between the Arduino and Pi.
- A special thanks to Campus Safety for unlocking IC-104 countless times for our group meetings.

III. TECHNICAL ANALYSIS

a.) Mechanical System

In the original design meeting, many forms of propulsion for a ping pong ball were discussed and evaluated. In the end, a compression spring system seemed to provide the best system for storing and applying equivalent amounts of kinetic energy to each shot. Targeting systems were also discussed, and a pan/tilt mechanism seemed like the most straight forward approach. Designs were drafted in Fusion 360, and 3D printed in PLA plastic unless otherwise noted.

Initial calculations found that a spring with a k-constant of around 1.0 lbs/in would provide us the best margin of error with compression. Obtaining a spring with around this constant was more difficult than originally anticipated. Most springs available at hardware stores or on Amazon are not sold with spring constant in mind, rather they are sold for specific application. Due to the bespoke nature of our design, we turned to McMaster-Carr for the

solution. There we found a 36-inch long spring with a known spring constant, and from there we were able to adjust for the modified length of spring, etc.

McMaster-Carr pt. # 9662k76: 36”		
K for 36” spring segment:	0.2 lbs./in	0.03502 N/mm
K for 6.5” spring segment:	0.2lbs./in*(36/6.5) = 1.108 lbs./in	0.19398 N/mm

Table 1. McMaster-Carr part number and unit conversions for the spring constant from imperial to metric

Once the initial spring and launch mechanism was tested, however, we realized our equations needed to be adjusted. It is not just the mass of the spring we are trying to accelerate forward, but also the mass of the plunger and rack system that mechanically pulls it back. This plastic component is an order of magnitude larger (0.01368 kg) than the mass of the ping pong ball (0.0027 kg), so our equations needed to be adjusted to accommodate. With updated equations we were able to design and print a new test assembly and resume testing and achieve some realistic results.

$$v_0 = \sqrt{\frac{kx^2}{m}} = \sqrt{\frac{(193.98N/m)(0.048m)^2}{0.0027kg + 0.0137kg}} = 5.22\ m/s \tag{1}$$

$$F * d * radius = \left(\frac{193.98N}{m}\right)(0.048m)(.01375m) = 0.128Nm \tag{2}$$

Equation (1) is the initial velocity calculation for the ping pong ball with our new spring with consideration of the mass of the spring-loaded plunger. Equation (2) is the torque calculation required for the DC motor to draw back the spring, where the radius term is the pitch radius of our pinion gear.

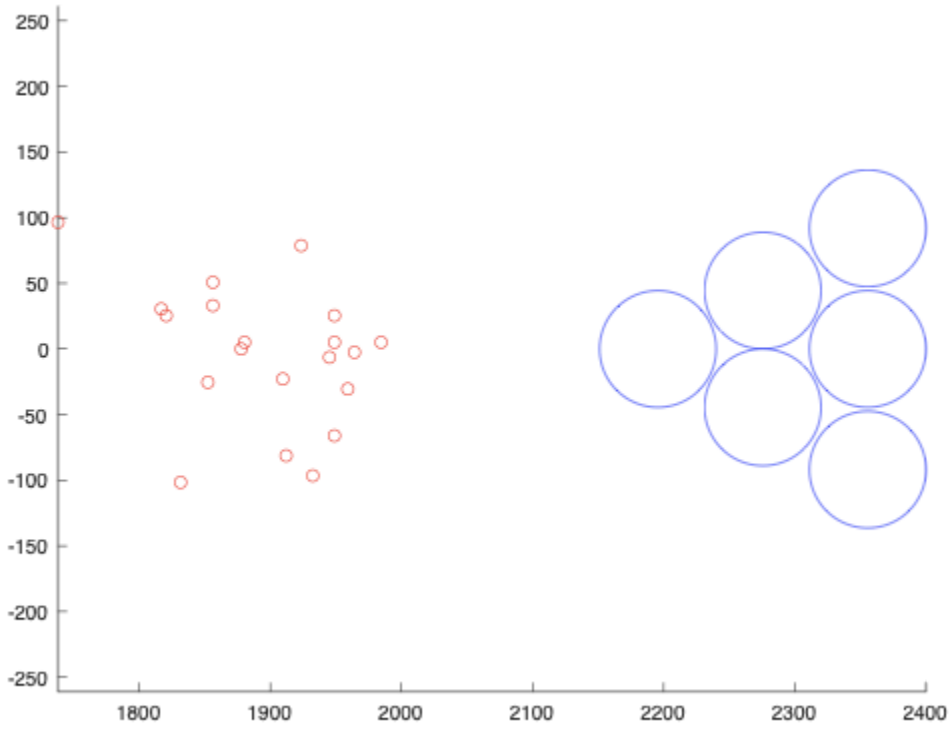

Figure 1. Initial testing of the launch mechanism. The initial position of the launch mechanism was at 0mm, with an initial height of 260mm. Blue circles are the target positions, and the red circles is the actual range achieved (20 launches). X and Y axes are in millimeters.

Figure 1 shows initial results with the 2nd version of the launch tube. The bias is only a few hundred millimeters from the target, but the variance is far too high with a standard deviation of 63.4 mm in the x, and 53.3 mm in y directions. These results were perplexing, as all of the forces involved were carefully designed to be inline with our tilt axis and shouldn't have altered the firing angle. Initially, we thought that the error in bias was due to friction below the rack system or due to a tolerance issue in the plunger/barrel system. After testing lubricants and making slight variations to the plunger and tube, we found that the main issue in our bias and variance was stiffness in the overall design. The heavier DC motor produces a lot of torque, and 1.1 lbs./inch does not seem like much, but over the full distance of throw it was enough to distort

the overall geometry of the structure while drawing back the spring. This second revision of our launch mechanism had a captive spring that would not allow the plunger to advance past a certain point and would arrest it's motion suddenly causing more vibration to the system. Through some slow motion video we determined that the length of the barrel that the ping pong ball is loaded into was too long, and needed to be shortened as the system vibrations were being magnified and imparted on the ball at the end of the launch tube.

The third and final redesign of the launch tube still captured the spring inside the barrel, but allowed for the plunger to move further forward and re-compress the spring after firing. This reduced much of the vibration, but did not resolve the warping imparted to the system between the spring force and the torque of the motor. The previous iteration of the launch tube was functionally 3 separate pieces: The bevel gear to adjust the tilt, a left section that attaches to the bevel gear and comprises the left half of the barrel, and a right component that comprised the right half of the barrel, a mount for the DC motor and attached to the right side of the pan/tilt mechanism (See figure 2A). All of these pieces were fastened together with M3 bolts. We revised this design after seeing how the system warped during the spring compression phase of each launch which was revealed during slow-motion video. The bevel gear, left and right bracket sections were merged into a single component and stiffening ribs were added Due to the limitations on the size of objects that can be produced on our 3D printer, the barrel sections had to be made into separate components.. Figure 2 below shows the design revision to a unibody design for the components that attach to the pan/tilt mechanism.

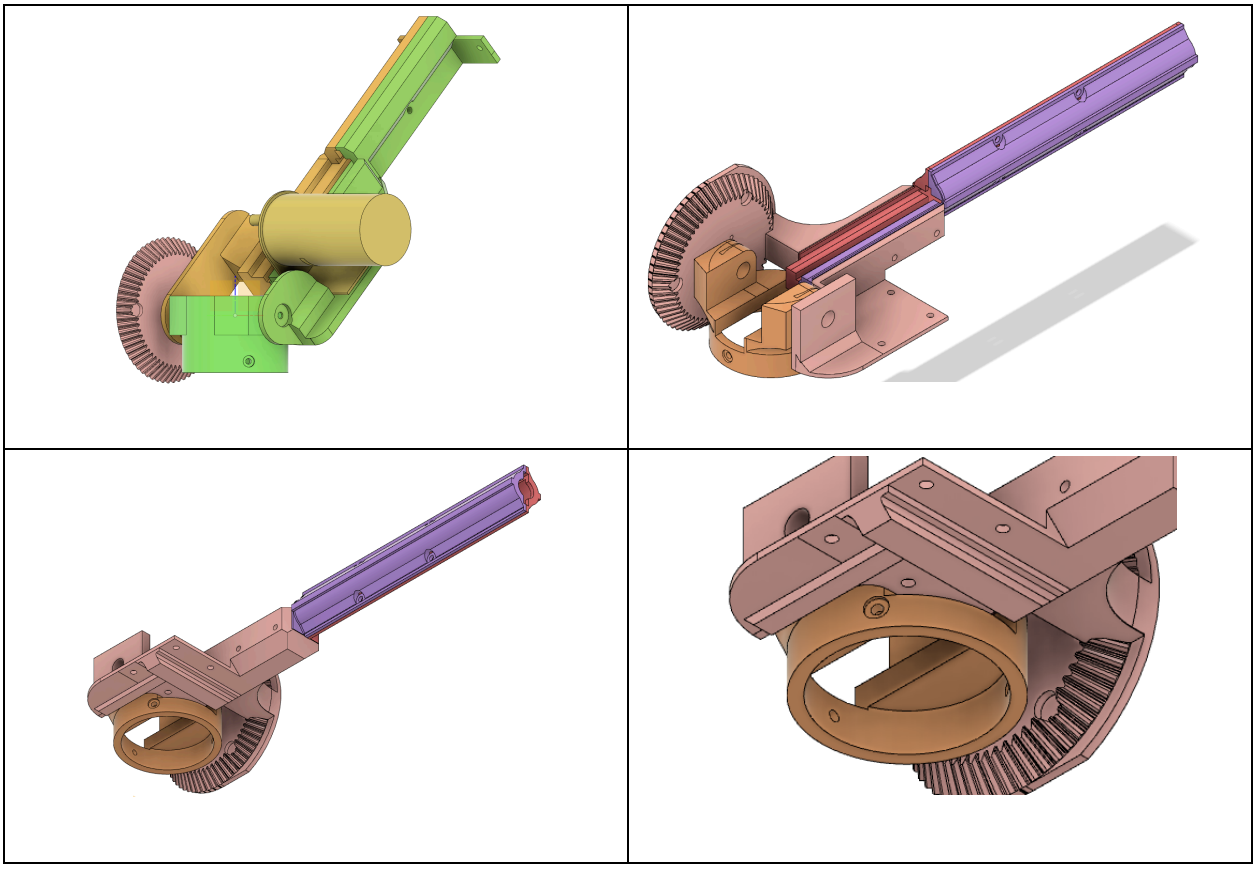

Figure 2. A (top-left): PongBot Assembly Version 2 comprised of 3 separate components. The bevel gear, left, and right brackets that enclose the spring/plunger and fasten the barrel to the pan/tilt assembly. B (top-right): PongBot Assembly Version 3comprised of 3 separate parts. The bevel gear and bracket components have been merged to a single piece and the barrel is still in two components fastened in place with M3 bolts. C & D (bottom-left, bottom right):. PongBot Assembly Version 3 with a view of the ribbing added to prevent flexing along the ribs axis. *Note: Colors identify individual components of assemblies.*

The pan/tilt mechanism used for the motion component of pongbot is powered by two stepper motors. Both of the stepper motors are placed in the base of the unit and use gears to transfer motion from the spindle to the launcher. The gears of the mechanism are unique in that they allow both stepper motors to work in tandem to provide a smooth panning motion. The gears are set up to act as a differential. When both stepper motors spin in the same direction, this results in a panning motion. If one of the motors stays locked in place and the other motor moves in either direction, a tilting motion is achieved from the system. Varying speeds and directions of travel from the stepper motors results in a smooth motion of both pan and tilt.

The gears used in the mechanism are also unique in that they have a chevron pattern. This type of gear is known as a herringbone gear. This design allows for both smooth and quiet operations at any speeds or loads. Another feature of this gear design is that it keeps all connecting gears lined up. This is especially beneficial in the case of the pan/tilt mechanism of pongbot as the mechanism itself is entirely 3D printed. This results in some slight play between moving components which can result in gears being slightly misaligned. This would have led to less than desirable smoothness of motion and introduced the potential for unpleasant noises. The motion from the steppers undergoes a 3:1 gear reduction before reaching the launcher mechanism. This allows the steppers to be configured in full-step mode which conserves higher torque output, while still having a higher positioning accuracy. The gear reduction also increases torque output of the system which proves to be beneficial to achieve accurate positioning. With this in place, each step from the motors correlates to 0.6 degrees of change in position of the launcher.

b.) Power System

The electrical components of the pongbot system are mostly powered by a regulated power supply. The specific power supply that pongbot uses is a switching power supply made by Mean Well. Having a regulated psu ensures that the output voltage is constant, regardless of what the current draw of the system may be, up to its rated maximum. A switching supply offers high efficiency and was readily available at the time of purchase, while a linear regulated supply was harder to find. The psu used provides an output voltage of 24 VDC and has a maximum current output of 14.6A. While the overall system should consume no more than 6A at this voltage, the psu was similarly priced as a 10A psu and gives a greater overhead should higher current

demands arise. The supply works with an input voltage of either 115VAC or 230VAC which works well with US outlet voltages.

The supply has several protections built in. These protections include short circuit protection, overload protection, over voltage protection, and over temperature protection. The short circuit protection came in handy during testing of certain components as mistakes were made with regards to wiring and accidental shorting to ground. This protection has not only spared the psu, but also the electrical components of pongbot. In addition to the protections that are built into the psu, pongbot also has a rocker switch between the main power cord and the input terminals of the supply. This rocker switch acts as an emergency stop should any problems arise that require immediate power down of the system. The rocker switch also has a built-in 5A fuse to protect against any short circuits or over current situations that are not corrected by the psu itself.

The pong ball launcher uses a spring to launch the ball. The spring is attached to a rack and pinion gear system. The pinion gear is attached to a geared DC motor which operates at 24 VDC and has a rated current of 0.33 A. This makes it directly compatible with the Mean Well psu found in the system. Power from the PSU is fed into a DC motor driver circuit which allows for time-based control of motor functions.

The motion of the pan/tilt mechanism is accomplished through the use of nema 17 stepper motors. These motors were chosen for their positional accuracy, high holding torque, and relatively cheap price. The specific stepper motors that are in use are 40mm long and bipolar. The motors have a phase current of 1.5 A which provides a holding torque of 42 N.cm. The stepper motors were previously driven by clones of the Pololu A4988 stepper motor driver. This

driver allows for 1/16th microstepping, has an operating voltage of 8 VDC to 35 VDC, and provides 2 A per coil. The microstepping is not used in pongbot's case as higher torque is preferred. The drivers were set to operate at a voltage of 24 VDC which falls in line with the voltage provided by the psu. While these drivers can provide 2 A per coil, this is with active cooling. This was not initially considered when choosing a driver and as a result, the drivers got very hot and ultimately burnt out.

In order to combat this issue, pongbot made the transition from the A4988 drivers to new DRV8825 drivers. These new drivers provide 1/32nd microstepping, a larger operating voltage range, and 2.2 A per coil. The key difference with these new drivers is that they can provide 1.5 A per coil without active cooling. The manufacturer even states that they can do this without a heatsink. Since the stepper motors operate at 1.5 A per phase, pongbot will still implement the use of heatsinks in order to keep temperatures under control. After implementation, the new drivers achieved high temperatures, even with heatsinks, but ultimately remained stable and functional. The choice to use these new stepper motor drivers proved to be a more cost-effective option than actively cooling the old drivers, and provided a more reliable solution for stepper motion control.

All stepper motor drivers used are connected into a CNC shield that was made for an Arduino Uno, but is compatible with an Arduino Mega. While this shield was not made for pan/tilt mechanisms, it allows for ease of operation of stepper motors. The Arduino Mega runs completely custom code to control movement of the motors of the pan/tilt mechanism since none of the active libraries for the shield worked for pan/tilt motions. The Arduino accepts positioning data, sent through USB-based serial communications, from a Raspberry Pi. The Raspberry Pi

used in pongbot is a version 4 model B board with 2GB of ram. The Pi is powered by an independent power supply, seperate from the main supply. The psu it uses provides it with 5.1 VDC and up to 3.0 A through the Pi's USB-C port. The Arduino Mega and the Intel RealSense D415 camera used are powered via USB cables from the Pi.

PongBot has a main control input box, which allows the user to operate the unit as intended. This box contains two buttons wired into a breadboard, which are then connected to the Pi's GPIO and GND pins. Two external pull-up resistors on the breadboard were initially used in order to get the buttons working properly, but ultimately ended up using the internal pull-up resistors of the Pi.

c.) Arduino Code

The microprocessor responsible for controlling the pan/tilt mechanism, as well as the pull-back of the rack from the DC motor, is an Arduino Mega. The Mega has a cnc shield made for an Arduino Uno connected into it. Since this shield was not meant for a Mega, and also not meant for pan/tilt motion applications, the libraries and code that came with it were useless. Custom code was written in order to control the stepper drivers connected to the shield and ultimately control the stepper motors.

The code starts with a simple setup loop that initializes serial communications and configures the pins used as either inputs or outputs. After some testing, the team realized that when the Pi would initialize serial comms on its end, the Arduino would reset the code it was running. In order to combat this, the Arduino holds at this state until something is received on its end. The Arduino then proceeds to execute a homing function and a centering function at the end

of the setup loop. The homing function relies on a limit switch connected to one of the Arduino's digital pins and is critical in order to achieve accurate positioning.

The code then proceeds by running through the main loop. This consists of running a serial input function, a movement calculation function, a movement execution function, and a ball firing function. The serial input function reads in data if there is something available in the serial buffer. It only reads the data if it is a non-zero, positive integer. It then parses the data in order to separate the one integer into two separate position points. This information is then used by the movement calculation function, which takes the current position of the unit and compares it to the new desired position in order to find out how much movement is required.

The movement execution function takes in the required movements and analyzes it in order to find out the direction of travel. It then sets the desired direction of each motor through the use of the driver's direction pins and executes the motion. This is done by creating a square wave with a pulse width of 12,000 microseconds and having the number of pulses match the number of steps required to move to its new position. The ball firing function controls the pull-back of the rack by spinning the DC motor of the launcher for a fixed length of time. This is done by controlling the gate of a mosfet in order to control the amount of time the motor is powered.

d.) Vision System

The primary function of the PongBot's vision system is to locate and identify a target sitting roughly 8 feet away. Further, the vision system will provide depth and horizontal distance data for each of the targets. Since PongBot's vision system requires both an RGB camera and a depth

sensor, it makes use of Intel's RealSense D415 Depth Camera. The RealSense D415 utilizes two infrared cameras that are combined to provide depth data. Therefore, at any given time the camera can produce four images: RGB, depth, left infrared and right infrared.

The RealSense camera is connected to a Raspberry Pi 4 Model B with 2GB RAM using a USB 3.0 connection. Our camera, however, is having problems recognizing the USB 3.0 connection and is instead operating with USB 2.1. This problem may be a result of the 3rd party USB cable or simply a camera hardware issue. This is a common problem with the D415, however, so there is support for running the camera with a USB 2 connection using the Linux SDK libraries.

Raspbian Buster was installed on a 32 GB micro-SD card which is well above the recommended 16 GB minimum. Intel's open-source, cross-platform library, SDK 2.0, was also installed on the Pi. SDK 2.0 provides four major components: the librealsense2 library, example code, tools and wrappers. The librealsense2 library contains an API that provides access to the depth camera stream. Finally, OpenCV (Open-Source Computer Vision Library) was installed.

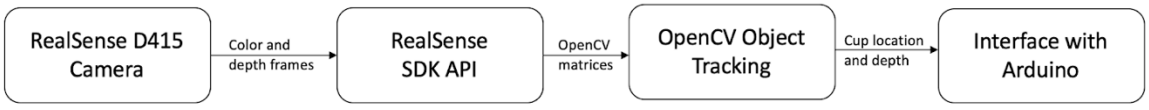

Figure 3: Overview of the camera application.

Figure 3 shows an overview of the camera application. First, the RealSense SDK Application Programming Interface interfaces with the RealSense camera to extract both color and depth frames. The final python script, pongbot.py, configures and produces an image

pipeline using the pyrealsense2.pipeline function. The pipeline grabs 8-bit BGR frames, since OpenCV utilizes BGR format. It also grabs the depth frames in z16 format, which provides depth values for each pixel. The OpenCV library is used for object detection, and finally cup location and depth are passed to the Arduino via a serial connection.

The pongbot.py script is set to run on boot, by calling the script in the RaspberryPi's rc.local file. The script is initialized when either the cup or bucket button is pressed. If the button is being pressed for the first time since the Pi booted, it will open a serial connection with the Arduino. Once serial communication is opened, the Pi will open an image pipeline with the RealSense camera. There is a 5 second delay at this point to give the camera adequate time to automatically adjust its exposure.

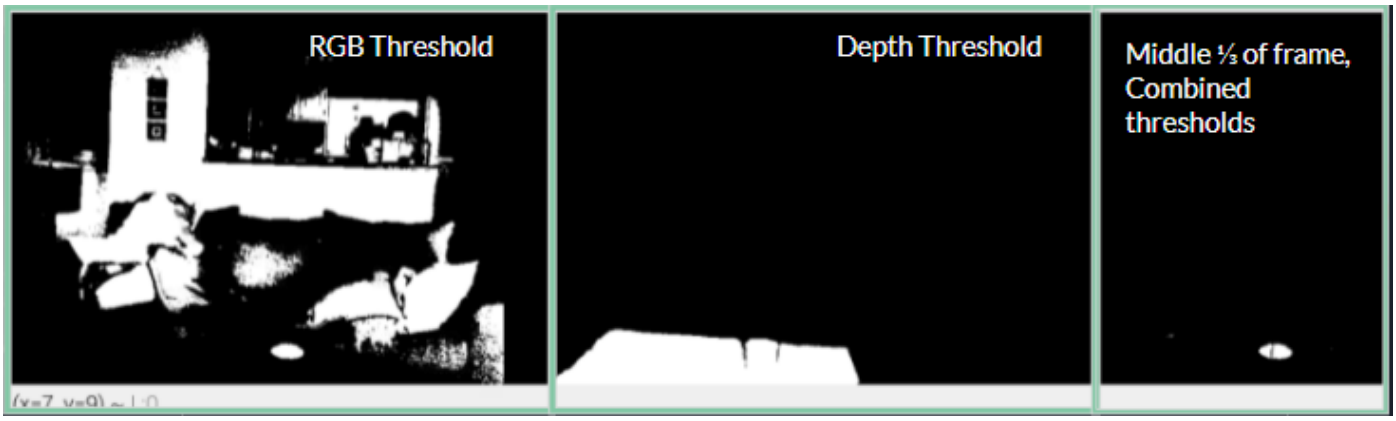

Figure 4: The leftmost image shows the RGB image with a threshold filter applied. The middle image shows the depth image with a threshold filter applied. The rightmost image shows the middle 33% of the frame, with RGB and Depth thresholds overlapped.

Once the image is taken, it is important to note that the depth stream is produced by two independent infrared cameras, and is already rectified when it passes through the image pipeline. The RGB stream is not, and therefore, the RGB and depth streams must be aligned. This ensures, for example, that the pixel at location (50,50) on the RGB stream corresponds to the pixel at

location (50,50) on the depth stream. Luckily, the librealsense library is equipped with a function that handles this.

After obtaining the aligned color and depth frames, the OpenCV API is used to convert the camera frames into OpenCV matrices that can be manipulated. A threshold filter is applied to both the RGB frame and the depth frame, such that only the brightest colors and closest objects are given values of 1. The threshold limits are calibrated specifically for Room 104 of the Willie Miller Instructional Center. The middle third of both RGB and depth thresholds are then combined and each pixel is given a final value of 1 if and only if both the RGB and Depth threshold values are 1. Figure 4 shows an example of this process.

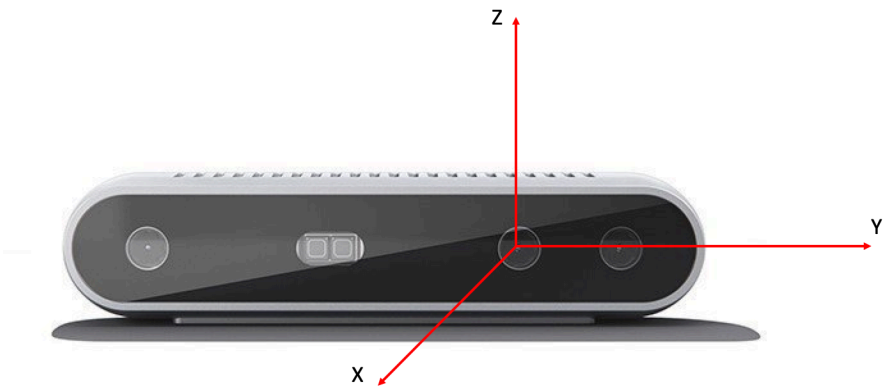

Figure 5:Coordinate System utilized by the RealSense D415 Camera.

The overlapped threshold image is passed through OpenCV's "erode" function in an attempt to separate cups that are touching. Then, another OpenCV function, findContours, is used to find the cups, and all bounding boxes and centroids are found. Targets are discarded if their bounding box is below some hard coded pixel size. This gets rid of any erroneous bright spots on the table.

Finally, the Pi pulls the distance for all final targets, and chooses the one closest to the PongBot. The closest target's depth and pixel location is passed to the RealSense function, convert_depth_to_coord, to find the (x,y,z) location in realspace, as shown in Figure 5. The closest value in the calibration table (for either cups or buckets) is found, and the corresponding tilt step value is passed to the Arduino. The pan step is found using the simple trig relation, $\theta = arctan(y/z) * (180/\pi)$. The three digit pan step and two digit tilt step values are concatenated into a 5 digit string and are passed to the Arduino.

IV. ANALYSIS OF PROJECT

PongBot's project scheduling was fairly accurate for the first 10 weeks of the project, as most of the design and production was completed by that point. We were smart to leave room in the final weeks to account for schedule slip, as most projects will always run into some unforeseen issue. Ours wound up being on the software side of things as we had some phantom code that was running in the background, causing confusion in the accuracy of our vision and targeting system. These issues were remedied by removing old versions of the code base from the Raspberry Pi's file system.

Our initial budget was $300 for the project. Our final total cost was $392.68. The majority of the overage was placed into various small components that were needed that could not have been accounted for at the outset, but a few items that were overlooked. These items include the XL Solo cups, buckets, and outdoor ping pong balls, as well as the main power cable, and a micro SD card for the Raspberry Pi. Overall we are happy with the budget on this project.

Pongbot's final demonstration went very well. The team showed up early and had the entirety of the system set up before class started. As per the requirements, the first shots were made at the buckets. The requirements specified an accuracy threshold of 80% when targeting buckets, but this was exceeded greatly with an achieved accuracy of 100% during the demonstration. After a total of ten shots to guarantee this, the buckets were switched out for the XL solo cups. While no accuracy thresholds were guaranteed, pongbot managed to win the game, only missing a few shots. Actual accuracy for the cups was not measured, as no threshold value was specified. Overall, Pongbot's final demo went without any issues and surprised the crowd with its spectacular performance.

V. GUIDANCE

If there were ever to be a group planning to achieve a similar feat as Pongbot's, it would be of great importance that they heed the following statements. When using an Arduino with serial communications, be aware that opening serial on the other end of the line will cause the Arduino to reset and re-run the setup code. If a Raspberry Pi is used, take note of CPU temperatures as they can climb high and cause instability. Turn it off and give it a chance to cool down every now and then, take this time to relax as well. When driving stepper motors, the A4988 stepper driver seems like a good option for its very cheap price. This can cause headaches when it burns out due to not having active cooling such as a heatsink and fan combo. In this case, use the DRV8825 drivers as they do not need a fan for most operations and are cheaper as a result, heatsinks are still required.

If precision of the system is not what you want it to be and nothing seems to be working right, take a slow motion video with your phone. It can help you catch small vibration issues that can lead to drastic reduction in precision. Python works well on the Raspberry Pi, but when you are done developing the code and choose to have it run on boot, be aware that other code in the same directory can run randomly when it's not supposed to, leading to duplicated system outputs. If using normal DC motors for motion, get some that have encoders to accurately keep track of position as time-based positioning is not as accurate. Take these words of wisdom and apply them to similar projects in order to avoid major headaches and wasted time.

VI. BILL OF MATERIALS

Item	Cost
Arduino/Expansion Board/Stepper Drivers	$25.64
2x Nema17 Stepper Motor 38mm	$19.98
Power Socket Inlet/Rocker Switch/5A Fuse	$7.99
Intel Realsense D415	$99.99
Mean Well 24V 14.6A DC Switching Power Supply	$29.00
Raspberry Pi 4 Model B, 2GB RAM	$46.11
DC Motor - 24v	$12.00
PVC pipe	$5.00
Spring	$5.00
Jumper cables / misc cables	$10.00
PLA Filament	$22.00
3mm Bearing Balls (Qty: 100)	$4.97
Raspberry Pi Power Cable	$10.00
IRF520 MOSFET	$1.00
1N4007 Diode	$1.00
2x Breadboard	$5.00
Arduino Mega	$16
ABS Filament	$5
C13 Power Cable	$7
Misc. nuts/bolts/washers	$10
Micro SD card	$10
XL Solo Cups	$15
Outdoor Ping Pong Balls	$10
Ace Hardware 2.5 qt. red buckets	$15
TOTAL	$392.68

VII. REQUIREMENTS

ID #	Description
RQ1	PongBot shall operate in room 104 of the Willie Miller Instructional Center with all overhead lights turned on.
RQ2	Games shall be played on two 2x5 foot tables, aligned on their shorter sides to create a 2x10 foot table.
RQ3	PongBot shall be mounted on one end of the 2x10 foot table centered on the centerline of the long dimension of the play area; the play area is to be defined as the full width of the table starting from 6.5 to 8 feet from the end of the table with pongbot.
RQ4	PongBot shall propel a STIGA Outdoor Table Tennis Ball to any target placed within formations defined in RQ9.
RQ5	PongBot shall shoot at two specific targets: ● Target 1:Buckets with modified white interior and diameter of 8 inches, height of 7 and 3/8ths inches. ● Target 2: XL red solo cups with a top cup diameter of 4.75 inches and height of 6 inches.
RQ6	Targets must be standing on the table, with the smaller diameter of the target making contact with the table.

RQ7	Prior to play, PongBot shall be calibrated by using buckets in the 3-cup triangle configuration as designated by Figure 7. Each bucket shall be filled with at least 2 inches of water.
RQ8	Each game of water pong shall commence with six targets placed in a 6-cup triangle orientation with the maximum cup distance 8 feet from the base of the PongBot.
RQ9	PongBot shall operate with the following legal orientations as referenced in Figure 7: 6 cup triangle, 3 cup triangle, 4 cup diamond formation, and 3 cup line. The center of each formation aligned with the centerline of the long dimension of the table, and the back row aligned with the 8 foot mark as referenced in figure 8. Each cup shall be filled with at least 1/3 and no more than 1/2 with water.
RQ10	A standard game allows each team one re-rack to one of the orientations defined in RQ9 once two targets have been removed.
RQ11	When a ping pong ball lands inside of a target during gameplay, the opposing team must remove the target.
RQ12	The accuracy of PongBot shall be 80% when targeting buckets.
RQ13	The spring shall be compressed by pulling back the connecting rod via a rack and pinion.

RQ14	The pinion gear shall have a pitch diameter of 1.070866 inches (27.2mm) and contain only 5 neighboring teeth of 16 concentric teeth placed on one side of the gear.
RQ15	The rack and pinion spring shall be moved by a 24-volt DC motor with a gearbox reduction of 5000:15.
RQ16	The rack and pinion drive motor shall not draw above 900mA under load.
RQ17	PongBot's launch mechanism shall be able to obtain a range of motion of between 0 and +90 degrees of pitch (tilt).
RQ18	PongBot's launch mechanism shall be able to obtain a range of motion of 90 degrees of pan with +/- 45 degrees both left and right of the centerline down the length of table.
RQ19	PongBot shall be powered with a standard AC power outlet of 110 VAC and draw no more than 14.6 A of current at 24 VDC.
RQ20	PongBot shall have an E-stop rocker switch located at the main power connection.
RQ21	PongBot's RealSense D415 camera shall be mounted 5 feet from the base of the pongbot, at a height of 14 inches above the table.

RQ22	PongBot's Realsense D415 camera shall be powered through USB via the Raspberry Pi.
RQ23	The Raspberry Pi shall be powered via a 5 VDC input through its USB-C port.
RQ24	PongBot shall employ a computer vision system capable of detecting position and distance of targets within a 1-5 foot distance from the camera.
RQ25	PongBot's computer vision shall function when only playable targets are present on the table located in the camera's middle third field of view. No other objects, hands, ping pong balls, etc. may be present.
RQ26	PongBot shall use the position information from the vision system to calculate optimal trajectory autonomously.
RQ27	PongBot shall have two "my turn" buttons to designate between targeting buckets and cups.
RQ28	PongBot shall only begin the detection and launch sequence upon manual activation of the "my turn" button.
RQ29	PongBot shall keep the spring un-compressed until the "my turn" button is pressed to activate ranging and launch.

RQ30	PongBot's "my turn" button shall be pressed at the discretion of the current users.
RQ31	For the purposes of the final demonstration, the current users shall be Daniel Lane, Rachel Conway, and Luis Osegueda.
RQ32	After manually loading one ball, PongBot shall be capable of autonomously firing with every manual press of the "my turn" button.
RQ33	PongBot shall be able to target and launch one ping pong ball within 45 seconds of the "my turn" button being pressed.
RQ34	The pan/tilt motion shall be powered by stepper motors operating in full-stepping modes.
RQ35	The pan/tilt mechanism shall have a 3:1 gear reduction for all motions.
RQ36	The pan/tilt mechanism shall provide a resolution of 0.6 degrees of motion in all axes.
RQ37	Stepper motors shall be driven using DRV8825 stepper drivers with heatsinks.
RQ38	Stepper motor drivers should operate with a reference voltage of 0.6 VDC.

RQ39	PongBot, all associated hardware, and play environment shall not be interfered with outside of setup, buttons, and necessary repairs.

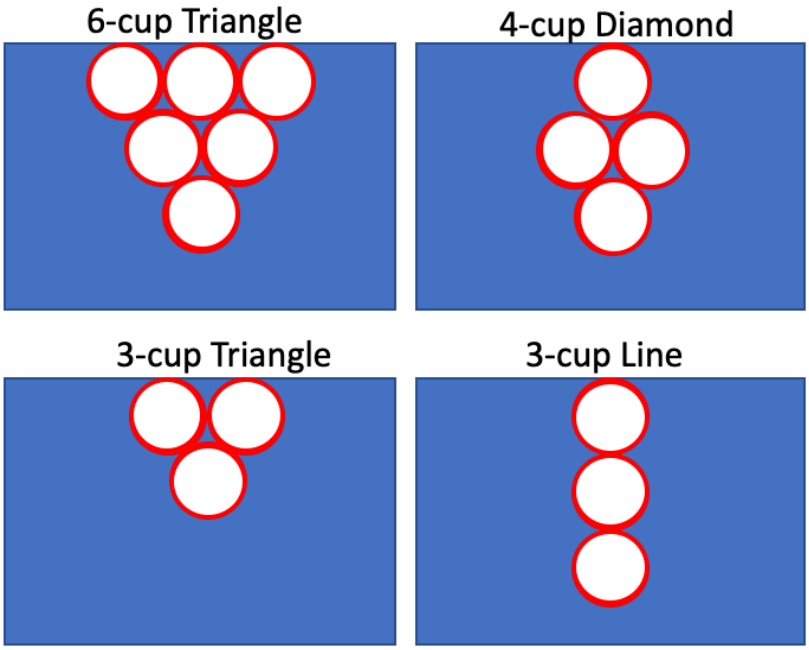

Figure 7: Allowable target formations for PongBot.

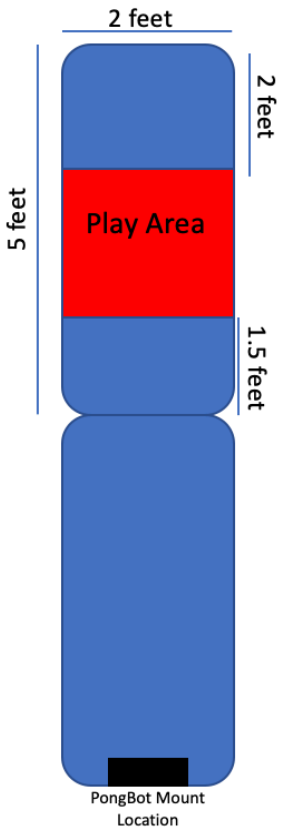

Figure 8: Play Area dimensions with reference to PongBot Mount Location.