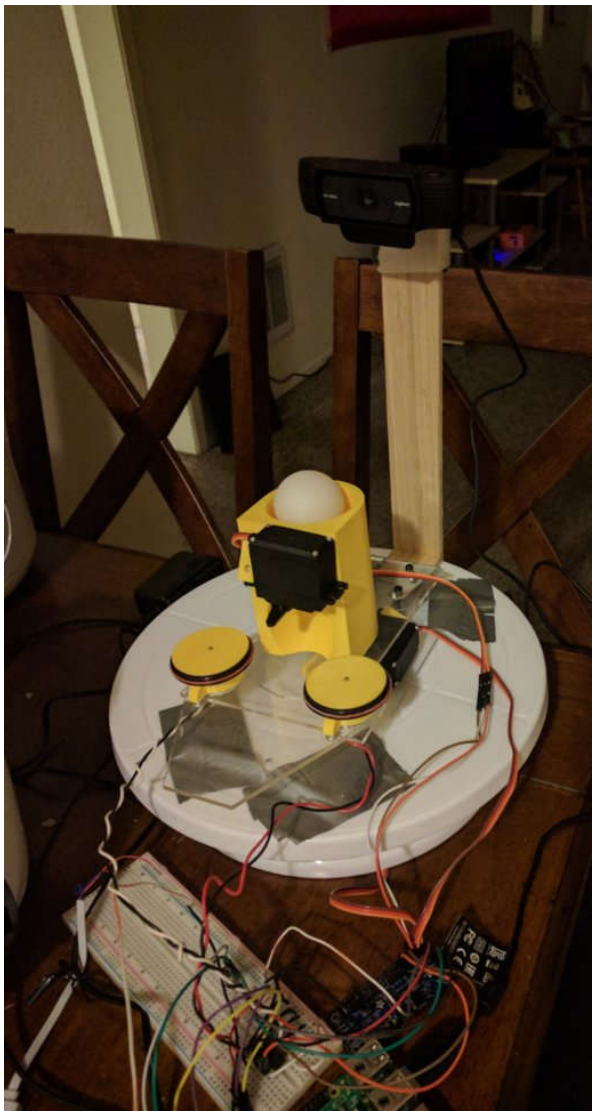
Ping Pong Trainer



Cal Poly Computer Engineering Senior Project

By Aaron Atamian

Advised by Andrew Danowitz

June 16, 2017

Contents

Introduction

Project Overview

For ping pong players, it can be beneficial to practice even when nobody else is able or willing to play. Additionally, players may want to be able to improve their returns on a certain part of the table. Currently, there are several automated ping pong serving machines. There are several flaws with these systems, however. They are not very accurate and do not allow you to set targets. Also, many of them only serve to a fixed location and do not rotate automatically to desired locations.

Project Outcome

The goal of this project is to produce a system that will track circular targets and try to hit them with ping pong balls. The system will use an attached webcam to detect the targets and estimate where they are in 3D space. This will be relayed to a program that drives servos such that it can hit the targets in the current frame. This allows the user to set targets at specific spots on the table where the player may have trouble returning the ball.

Engineering Specifications

Bill of Materials

Item	Vendor	Quantity	Unit Cost	Tax	Total
Cake Turntable	Amazon	1	9.99	0	9.99
2 Pcs Electric Mini Motor	Amazon	2	4.99	0	9.98
Assorted M3 Nylon Screws	Amazon	1	8.9	0	8.9
3" Red Color-Coding Sticker Dots (Pack of 500)	Amazon	1	15.99	0	15.99
MG996R Digital Servo (pack of 4)	Amazon	1	26.99	0	26.99
Logitech C920 Webcam	Best Buy	1	61.99	4.8	66.79
Lasercut Parts P1	Ponoko	1	34.74	1.92	36.66
Lasercut Parts P2	Ponoko	1	15.37	0.42	15.79
Adafruit Servo Driver and H-bridge	Adafruit	1	22.55	0	22.55
Super Glue, O-rings, Drillbits	Home Depot	1	10.93	0.85	11.78
Samsung 32GB MicroSD Card	Amazon	1	9.99	0	9.99
Raspberry Pi 3 w/ 2.5A Power Supply	Canakit	1	42.99	0	42.99
				Total	278.4

Adjustments

An earlier version of the bill of materials had a different webcam. The plan was to use two Logitech C270 webcams with a powered USB hub for stereo vision. Due to time constraints, it was decided that a single webcam would be sufficient. Implementation details are explained later in the report as to how this works to estimate distance.

The Logitech C270 was initially used as the single camera, but because of compatibility issues encountered, I decided to go with a different camera altogether. It is worth noting that there are known issues with that specific camera when paired with a Raspberry Pi [1].

Target User

This trainer would have the most appeal to somebody who competes in ping pong tournaments and needs to perfect their ball return. Because this person is always practicing, they would sometimes have a hard time getting friends to play with them. The player would now be able to practice with an automated serving machine.

Circuit Schematics

Motor Control Wiring

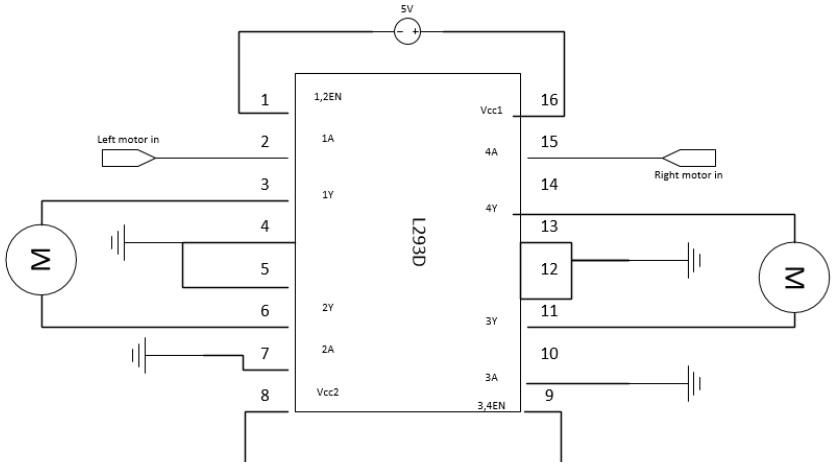


Figure 1 L293D Wiring

Figure 1 shows the wiring for the motor controller. The PWM input for both the right and left motor are connected to channels 3 and 4 on the Adafruit I2C board respectively. This is demonstrated on the next schematic. Also, worth noting is that one of the two motors must be wired backwards or else one motor will try to feed out the ball while the other will try to feed it back – preventing the ball from ever getting launched.

Servo Controller Wiring

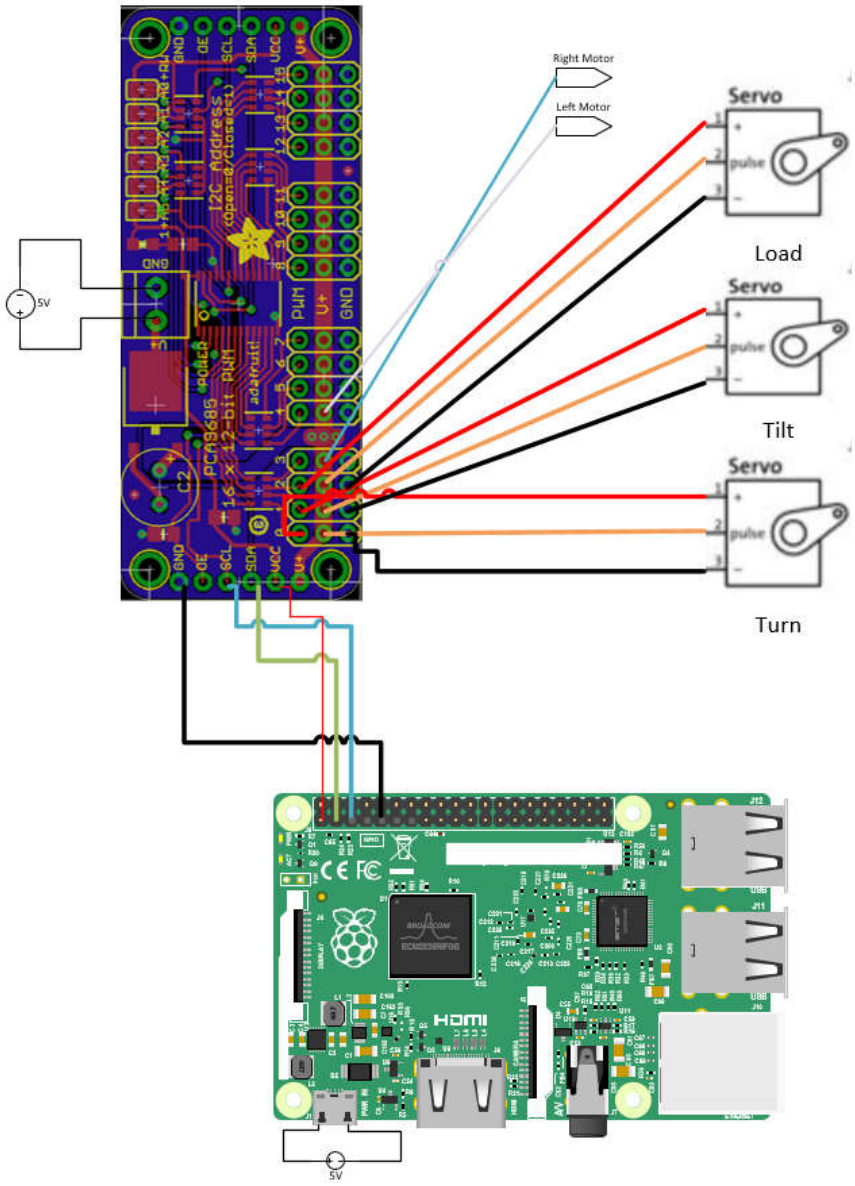


Figure 2Servo and Pi Wiring

Above, one can see the Raspberry Pi wired to the servo controller which is in turn wired to three of the servos and the two motors. The outputs go to the respective inputs on the first circuit diagram. It is *very* important that the source to the Raspberry Pi is different than the 5V source to the I2C breakout board. Therefore, the breakout board is not wired to the 5V GPIO ports on the Pi. Also, for best results, power the Pi with a source that outputs at least 2A.

Background

This project is implemented using OpenCV and Python and is run off the Raspberry Pi. The system is split into two main portions. The first being the computer vision and the second being the mechanical/electrical.

The computer vision is broken down into four steps: image segmentation, thresholding, identification, and distance estimation. Upon experimentation, it was found that red targets would be the easiest to differentiate from the surroundings. Thus, the algorithm thresholds HSV color spaces based on the color red. Next, the software identifies circular objects. Then, using the circumference of the detected circles and the known focal length of the camera, a distance estimation was created.

The launcher itself is based heavily off an Instructable [2] with some modifications to allow the launcher to turn as well as tilt up and down. The servos are wired to an I2C breakout board that is in turn wired to the Raspberry Pi.

Final Design Process

Installing OpenCV and Configuring Raspberry Pi

One of the preliminary tasks that proved time consuming was installing OpenCV onto the Raspberry Pi. I followed an online tutorial for the installation [3]. The process for installing OpenCV on the Raspberry Pi took about a day to do – likely due to the slow read and write speeds of the microSD card.

After installing OpenCV, I worked on getting the webcam working. After having issues with my first camera choice, I settled on the Logitech C920 as there seemed to be no issues listed in the resource referenced. This camera proved to work right away with no extra software installations [1].

Hardware

Building the Launcher

The ping pong ball launcher used is heavily based off the Instructable article referenced [2]. All .stl files from the Instructable were 3D printed using the Innovation Sandbox at Cal Poly's campus. The parts were of good quality and worked once the launcher was put together. The next part from the Instructable was the laser cut base. I decided that acrylic would be the best material to use. For the vendor I used to laser-cut, the part had to be in .eps format – an Adobe Illustrator file. I attempted to use the provided

Figure 3- First Attempt at Laser-cutting

.dxf file, and imported it into Illustrator. Once I marked the cuts the appropriate color per Ponoko's website (the laser-cutting service), I submitted the part to be laser-cut. Once the part arrived, I realized that the measurements were off (see Figure 1). I then went over the cuts in illustrator and printed the file on paper to make sure the measurements matched. Once I received the newly submitted part, all holes and cuts matched up.

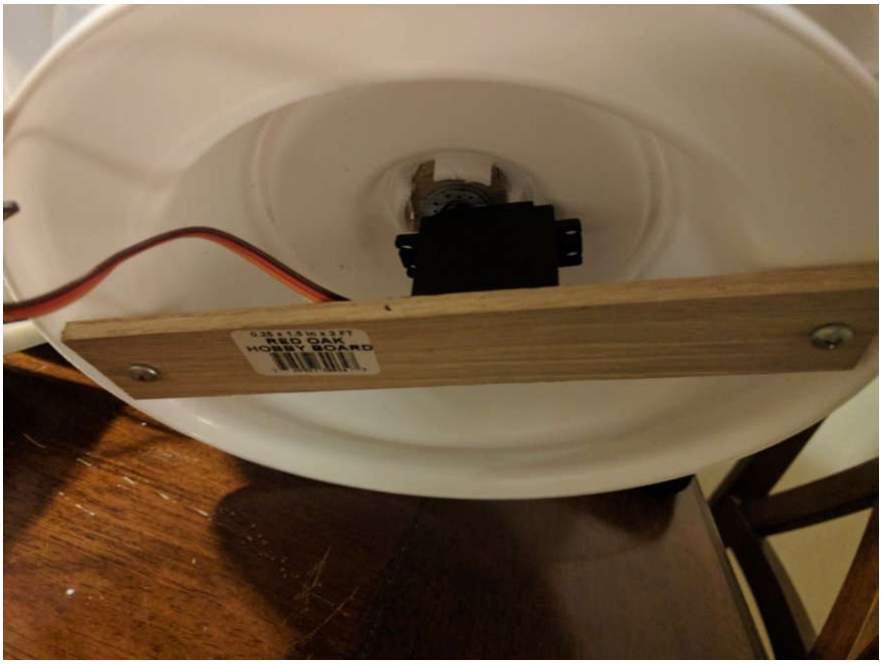


Figure 4 Servo Mounted Under Cake Turntable

After following the Instructable, I decided to add one important modification – the ability for the launcher to turn left and right. In order to accomplish this, I decided to use a cake turntable as the base. Underneath the center of the cake turntable, I glued a small circular wooden piece to the top. The plastic servo horn of the MG996R was attached to the wooden piece. Once the servo was attached to the servo horn, it was possible to turn the launcher.

Most parts of the launcher are either secured with screws or super glue. In some ways, this proved to be problematic as the super glue did not bond plastic on plastic well. This is why the bottom of the cake turntable uses a piece of wood in between it and the servo horn.

Assembling the Circuit

Next, the final circuit needed to be put together. First, the motors needed wires soldered onto them. They were tested by simply putting voltage across them. After following the circuit in the engineering specifications above, the next step was to develop a servo driver that could use the Adafruit servo driver to control the servos and motors. The completed circuit can be seen in Figure 5.

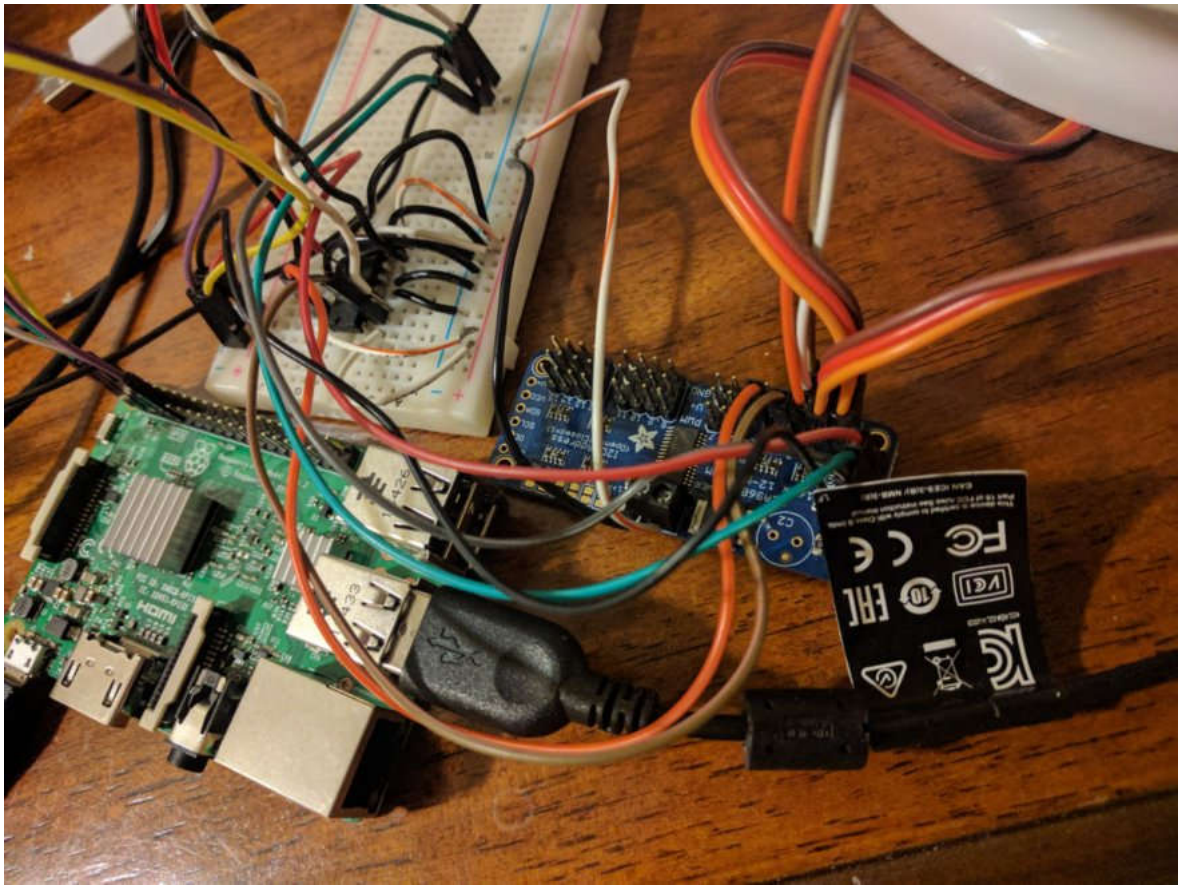


Figure 5 Final Assembled Circuit

Software

Image Segmentation and Distance Calculation

The first part of the image processing portion is the segmentation stage, which isolates potential targets from the rest of the frame. This is done by performing a binary segmentation on the various components of the HSV color space. To do this on the Raspberry Pi, first use OpenCV's `cvtcolor` function. This converts the color space of the image to HSV. Then, based off a color picker from an image editing tool, I found the expected values in HSV color space for red. The image editing tool I used (Gimp) used a different scale for hue values than OpenCV, which initially caused issues with

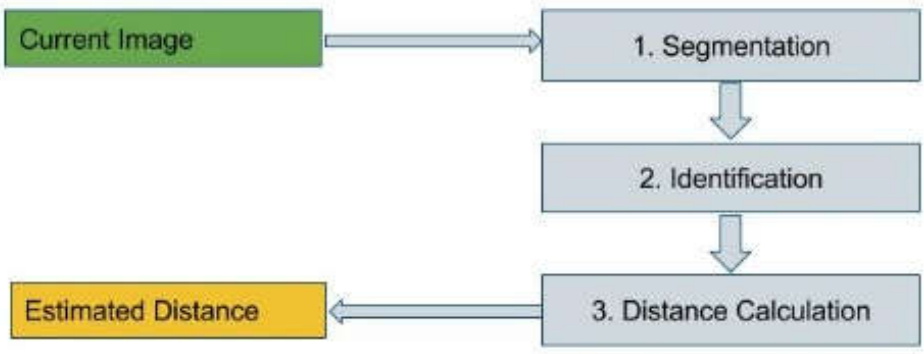


Figure 6 Flow Diagram of Image Processing

segmentation. Upon solving this, the next step is to use the `inRange` function which took in the converted image plus lower and upper bound arrays for pixel values to highlight. Then, this outputs a binarization used by the next step.

Next, the identification stage produces coordinates for likely targets from the segmented image. It does this by analyzing each contour and determining if it is a suitable target. Both the area and the circularity are considered when determining whether something is a target or not. This is done by using OpenCV's `findCountours` method which is then passed into SciPy's `label` function. Between these two functions, the program can identify what is a target when used in conjunction with the `contourArea` and `contourHull` functions. Based off experimentation, I determined what the ideal value for minimum area and circularity to yield the most accurate results.

Finally, the last stage of image processing is the distance calculation stage. This distance calculation is based off the focal length of the camera and the known width of the target. The algorithm implements the formula $F = (P \times D) / W$, where F is the focal length, P is the width of the detected object in pixels, D is the distance from the camera, and W is the width of the actual object. This also means that to get a number for focal length, we need to run a calibration to output a number. This is done by putting a target a set distance from the camera. After running the calibration, a focal length is generated and that number is set as a constant to be used in future runs of the program [4].

Servo Driver

After it was verified that everything was wired up properly, it was time to focus on developing a program to drive the servos connected to the I2C breakout board. Thankfully, Adafruit has an API for communicating with the servo board readily available [5].

The `set_pwm_freq` function is used for setting the frequency of the PWM pulses and takes in a single numerical value in hertz. For best results, 60 Hz was used as the frequency. The `set_pwm` function which takes in a number for the channel (0-15), a numerical value out of 4096 to specify when the pulse will go high and low was also used throughout. There are constants for each servo and motor corresponding to the channel they are connected to. Additionally, there is an array in the code used to keep track of the current value of each servo. Lastly, functions were created to drive each and both motors [5].

At first, a program that was controlled in the command line via keyboard strokes was used to test the system. This allowed me to verify that the servos worked and was then integrated into the complete system. After making sure the servo driver worked, I then came up with a function that angles and shoots a ping pong ball at a target when given a depth estimation. The servo attached to the turn

table turns so that the targets X coordinate is in the center, then the launcher is angled and the ball is launched at a speed based off the distance.

Testing

To test the functionality of the system, it was decided that it would be best to test it indoors as that is generally where ping pong tables are located. I did not have access to a ping pong table so the launcher was placed onto a kitchen table. Red stickers were put onto a piece of white paper and put at a location on the table - like in Figure 7. As seen in Figure 8, the segmentation came out

Figure 7 Similar Image to the One Segmented

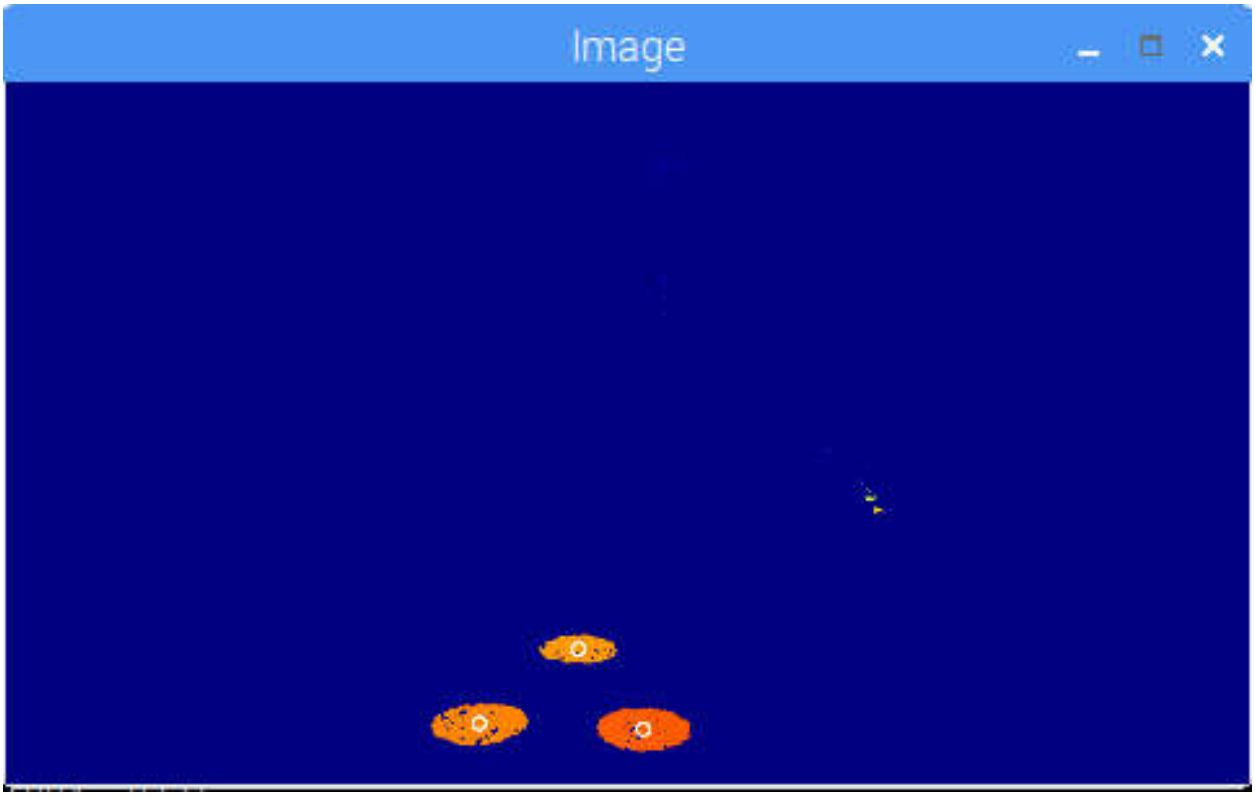


Figure 8 Image Segmentation

rather well for the three objects.

Once the segmentation proved to be accurate, it was time to test the distance estimation. Figure 9 shows distance estimation for the three targets. The program put them at 25.1 inches, 25.3 inches, and, 34.0 inches away from the camera lens. This proved to be fairly accurate. When measured, the actual distances were 26.5 inches, 26.6 inches, and 36 inches. This provides a percent error of about 5.5%.

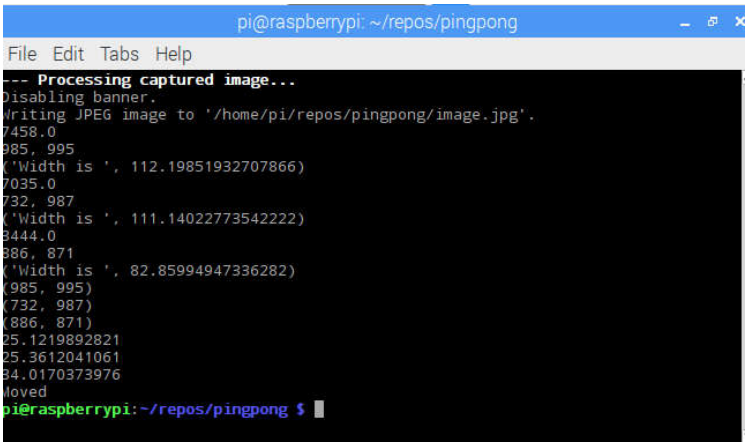


Figure 9 Distance estimations

Last, it was time to test the accuracy of the ping pong ball launcher. This is difficult to demonstrate here, but as seen in the demonstration [6], the launcher was not very accurate in the end. This is due to the lack of physics used in the algorithm to aim the ball. Due to time constraints, I was unable to acquire a device to measure the velocity of the ball leaving the launcher. This meant I was unable to come up with a correct acceleration to be used in kinematics equations [6].

Limitations

In testing, it became apparent that there are a few limitations. Firstly, the further the object is away from the camera, the less likely it will be given an accurate distance. This means that the launcher will not be accurate for distances past 7 feet or so. Next, the image segmentation does not seem to work well when in direct sunlight. Thus, it is advised to only use the system indoors.

Conclusion

This project proved to be rather engaging and time consuming to implement. With all the facets to the project, this proved a rather daunting task. Building the launcher proved time consuming while the electrical side was rather straightforward. While I could implement this project, if I had to do it over again, I would do a project that did not require me to create something physical so I could focus more on the electrical and software side.

Although I had doubts initially, this project has proven that a single camera can be used to provide a distance estimation. The computer vision side proved to be rather difficult and time consuming as it required extensive research into the OpenCV library. Overall, I have become impressed with the processing power of the Raspberry Pi 3 and will try to use it in future side projects.

Appendix

Github Repository of Relevant Code: https://github.com/evilaaron11/pingpong/tree/master

References

[1] "RPi USB Webcams," eLinux.org, 31 March 2017. [Online]. Available: http://elinux.org/RPi_USB_Webcams. [Accessed 20 May 2017].

[2] F. Dias, "Arduino Controlled Ping Pong Balls Launcher," Instructables, [Online]. Available: http://www.instructables.com/id/Arduino-controlled-Ping-Pong-Balls-Launcher/. [Accessed 15 January 2017].

[3] A. Rosebrook, "Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3," 18 April 2016. [Online]. Available: http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/.

[4] A. Rosebrock, "Find distance from camera to object/marker using Python and OpenCV," pyimagesearch, 19 January 2015. [Online]. Available: http://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/. [Accessed 23 May 2017].

[5] K. Townsend, "Adafruit 16 Channel Servo Driver with Raspberry Pi," Adafruit, 4 May 2015. [Online]. Available: https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/overview. [Accessed 1 May 2017].

[6] A. Atamian, "Ping Pong Trainer Test," 8 June 2017. [Online]. Available: https://youtu.be/Am4a5kJhysA. [Accessed 8 June 2017].

[7] "Using a standard USB webcam," Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/documentation/usage/webcams/. [Accessed 20 May 2017].