Lab 5: Visual Servoing

Group 5

Juliette Mitrovich, Sheila Moroney, and Sujani Patel

Lab Summary

The purpose of this lab was to use a Raspberry Pi with an Arduino to control a servo based on visual movement of a tennis ball. While setting up the Raspberry Pi with OpenCV and Python, we ran into some issues. We finally got it to work and, in the process, learned how to use Unix commands in the terminal and how to change the python shell version. We learned some basic python and used OpenCV commands to get the camera working and change the thresholding values of the image. Then, we used a few more OpenCV commands to find the tennis ball's center and draw a circle around it. Finally, we learned how to use serial connection for the Pi and Arduino to communicate. We wrote an Arduino program for a sensor to map its rotation to the x position of the tennis ball in the camera's image. We wired up an additional servo to move the tennis ball back and forth so we could collect consistent data. The tennis ball was very heavy and hard to move with the servo, so, for our final setup, we drew a circle on a stiff piece of paper with a highlighter to replicate the tennis ball. Our setup is shown below, in Figure 1. Overall, this lab was very helpful for us, since we are not very familiar with python. This lab will help us a lot for the competition lab.

Working Principles

Visual Servoing systems use feedback information from a vision sensor to control the motion of an object. As shown below in Figure 1, our camera is mounted stationary on a Raspberry Pi. It is then used to get the coordinates of a tennis ball shape, which is mounted to a servo motor that moves consistently back and forth. Then based on those coordinates, an additional servo motor moves to a relative location. Thus, the overall working principles of visual servoing in this lab are the camera, controlled by the python algorithm, finding the coordinates of the tennis ball and sending that information through a serial connection to an Arduino. The Arduino controls the servo by mapping the received coordinates to the servo rotation values.

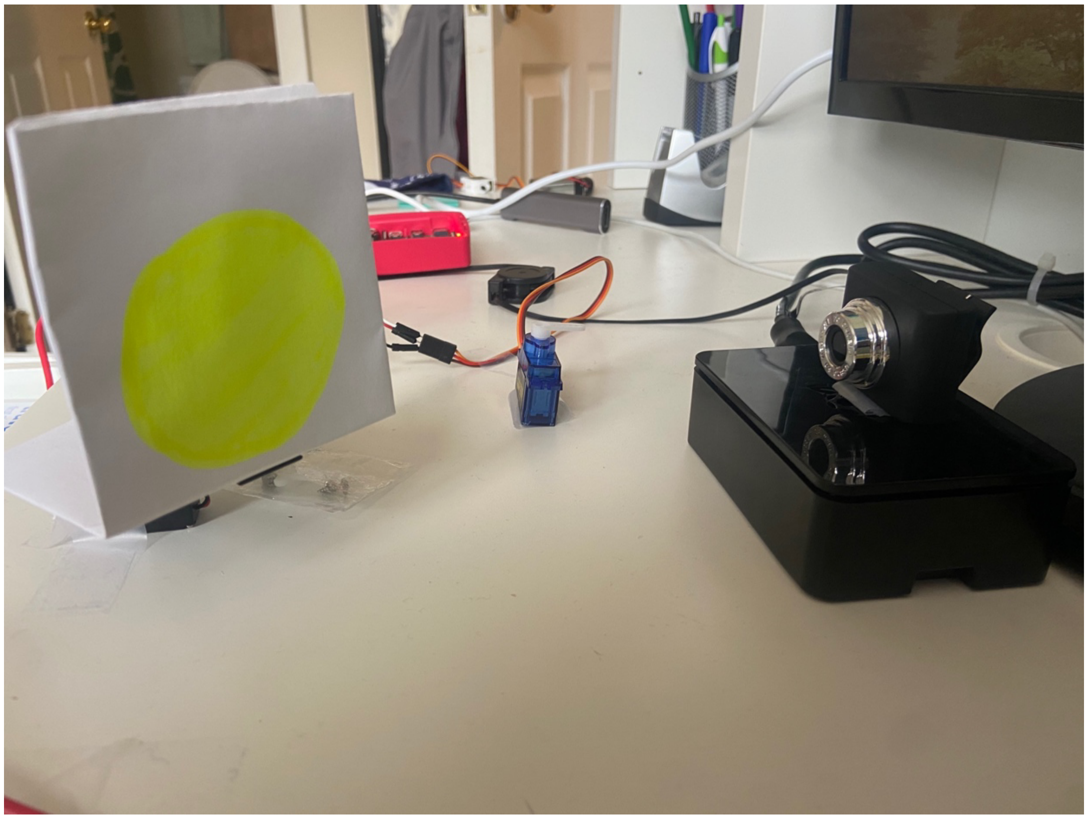
Figure 1. Camera, servos, and "tennis ball" setup

Algorithm Development

For this lab, we had three separate algorithms we had to develop. First, we created an Arduino code that would sweep a servo motor from 0 to 180 degrees at various frequencies found in Figure A-1 in Appendix A. Attached to the arm of the servo is the "tennis ball" the camera must track. To do so, we started with the servo sweep example sketch, readily available to you when you download the Arduino IDE. Within the sketch, there are two for loops, one that iterates up from 0 to 180, and one that iterates down from 180 to 0, allowing you to cover the full motion of the servo motor. To change the frequency, we simply changed the value by which the for loop was counting. The actual frequency was found later by taking the inverse of the amount of time it took to complete one cycle from the collected data. The values at which we iterated, and their corresponding frequencies can be found below in Table 1.

Table 1. Three different frequencies of the Arduino sweep program

Iteration Value	Corresponding Frequency (Hz)
+- 1	.252
+- 2	.505
+- 3	.791

The second program we created was a python program on the Raspberry Pi. The goal was to isolate the tennis ball, find its center coordinates, and send the horizontal center coordinate to the Arduino. This code can be found in Figure A-2 in Appendix A. We started with the example code given in the lab description and followed the instructions very closely since none of us had coded in Python before. The code starts by importing all the necessary libraries. Next, it initializes the serial communication with the Arduino. It was important that we had the correct port name, as well as a baud rate equal to the one on the Arduino for the data to successfully send. The next part of the code is where we isolated the green color of the tennis ball and created a mask so the camera would only pick up the ball. The upper and lower limits for the green were chosen by experimenting with the values. These also changed based on the time of day due to different lighting conditions. Now that we've isolated the ball, we used "cv2.findContours()" to allow us to create a circle around the tennis ball. From there, we used the circle to find the center coordinate. Before we sent the data through the serial port, we had to do two things. First, we had to scale down our values so they would range from 0 to 255, instead of 0 to 650. This is because Arduino can only read values from 0 to 255 because of its resolution. Second, we had to make sure that when we sent the int variable, it was indexed as an array so the Arduino could read it. The rest of the code tells you to release the capture when everything is done.

The final program we created for this lab was the Arduino code that reads the incoming data from the serial port and estimates the angle of the first servo output by going to that angle. The first step in this code was to read the data from the serial port and store it in a variable. To make sure we were only reading serial data if it was there, we placed the Serial.read() function within an if statement saying, if there is greater than 0 bytes of the serial buffer, go ahead and read the data. Since the data coming from the Pi had a range of 0 to 255, we needed a way to change that range to 0 to 180 to allow you to program the servo correctly. We accomplished this by using the map function. The final step of the code was to place the mapped value into the servo.write() function. In the end, we were successfully able to track the center of the tennis ball with the servo motor, via serial communication.

Tracking Performance Data

Aside from getting the servo motor to be able to track the center of the tennis ball, the other important part of this lab was to observe how changing the frequency at which the tennis ball moves affects the tracking ability. From watching the movement of the tracking servo as the tennis ball moves at different frequencies, we could see that the lag in the tracking servo increased as the frequency increased. To corroborate this, we collected data at each frequency. The data we collected were the time, the position value sent from the Raspberry Pi, and the position value received on the Arduino. Below, you can find the graph for frequency = .252Hz, .505Hz, and .791Hz in Figure 2, Figure 3, and Figure 4, respectively.

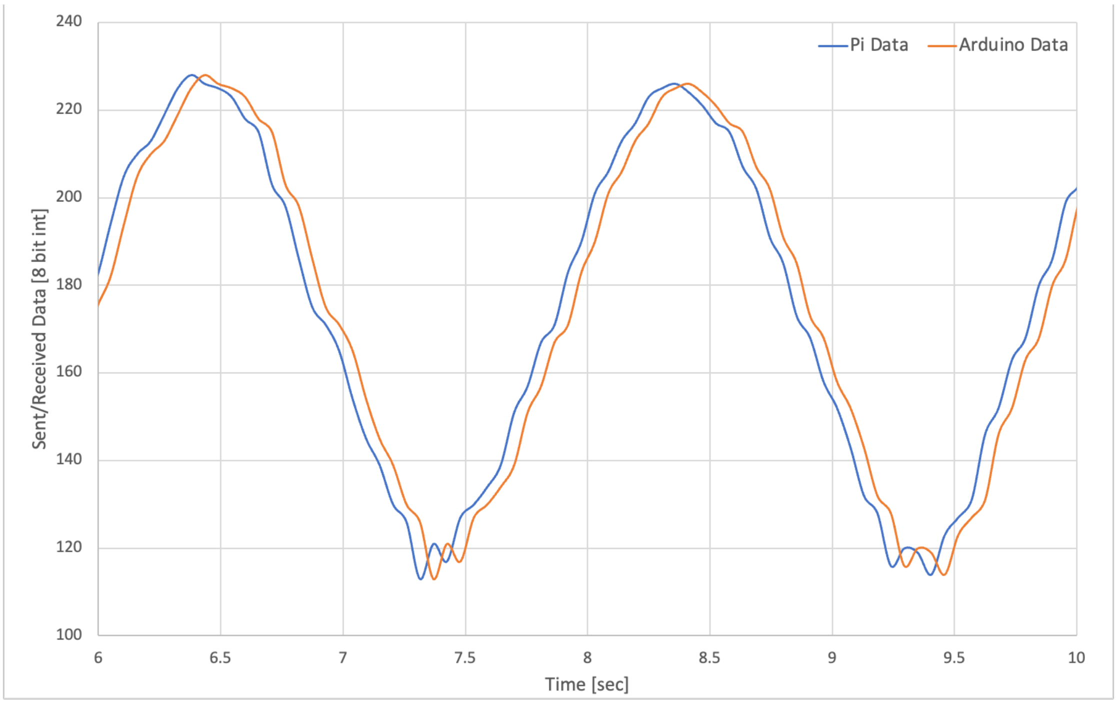

Figure 2. Graph of data sent by the Pi and received by the Arduino at f = 0.252 Hz

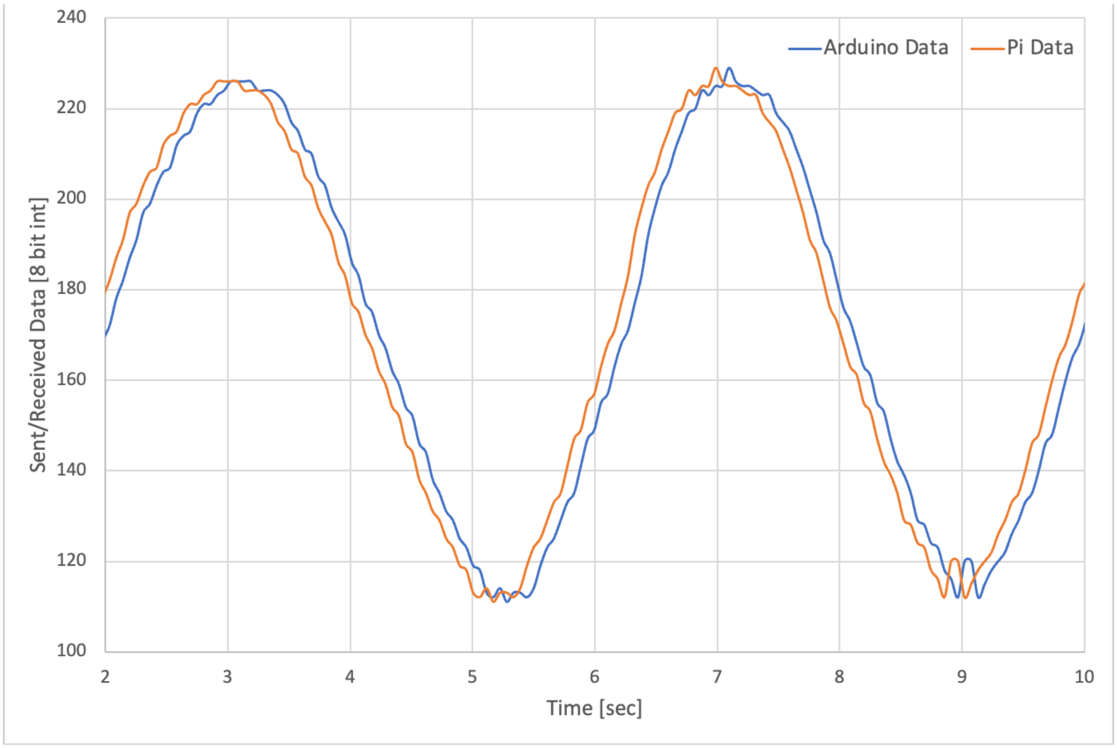

Figure 3. Graph of data sent by the Pi and received by the Arduino, oscillation frequency = 0.505 Hz

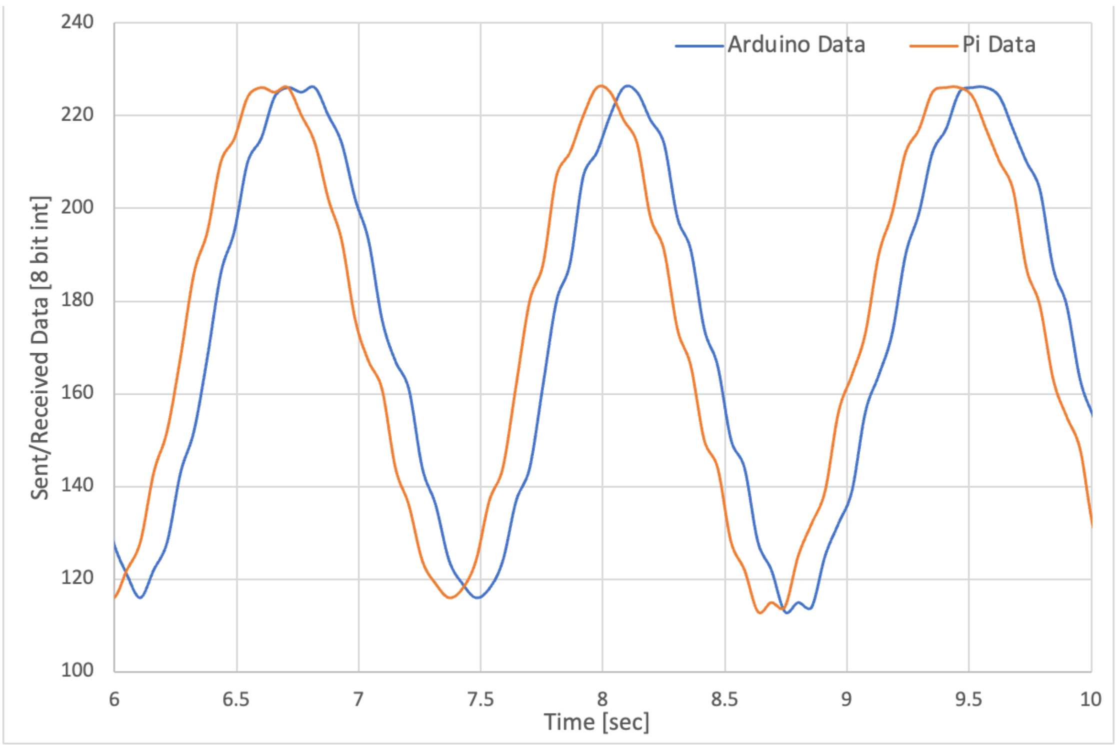

Figure 4. Graph of data sent by the Pi and received by the Arduino, oscillation frequency = 0.791 Hz

From the 3 graphs, you can see that there is a lag between the time when Raspberry Pi reports it is at one location and when the Arduino reports it gets to that location. To quantify this, we calculated phase lag in degrees. The equation is as follows:

$$\text{Phase angle} = 360º * f * \Delta t \quad (1)$$

Please see Table 2 below for the phase lag that occurred at each frequency.

Table 2. Phase lag at each corresponding frequency

Frequency (Hz)	**Phase Lag (º)**
.252	4.99
.505	19.99
.791	31.3

These calculations confirm what we initially saw. The lag in tracking increases as frequency of the servo moving the tennis ball increases.

Conclusion

We all really enjoyed this lab. It was all our first-time using Python, and it was amazing to see the capabilities of the Raspberry Pi and OpenCV. Unfortunately, we were not able to get the tracking to work when we mounted the camera onto the tracking servo motor. This is something we will continue to investigate as it may be beneficial for the competition lab.

Appendix A

Figure A-1. Source code for the Arduino, moving the tennis ball at a fixed frequency

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int pos = 30;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 30; pos <= 160; pos += 3) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 160; pos >= 30; pos -= 3) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

Figure A-2. Source code for the Raspberry Pi, isolating and sending the tennis ball's horizontal center coordinate

```
# ME 545 Lab 5
# Group 5: Juliette Mitrovich, Sheila Moroney, Sujani Patel
# Isololate a tennis ball from an image, find its center coordinates, and send
# the value through the serial monitor

# Needed libraries
import numpy as np
import time
import string
import cv2
import serial

# initiate serial communication with Arduino
ser = serial.Serial('/dev/ttyACM0',9600)

# get the video from the camera using cv2
cap = cv2.VideoCapture(0)

while(True):
    ret, frame = cap.read()
    # set the upper and lower limit to isolate the green color of the tennis ball
    lower_green = np.array([27,35,30])
    upper_green = np.array([67,255,255])

    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    # create a mask for the picture and erode it to isolate the tennis ball
    mask = cv2.inRange(hsv, lower_green, upper_green)
    mask = cv2.erode(mask, None, iterations = 2)

    # find the contours of the tennis ball
    cnts = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
    hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]

    # create a circle aroud the tennis ball and find the center coordinate
    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        center = (int(x), int(y))
        radius = int(radius)
        if radius > 3:
            # display the circle on the image
            cv2.circle(frame, center, radius, (0, 255, 255), 2)
            cv2.circle(frame, center, 1, (255, 0, 0), 10)
            x_send = int(x*255/650)
 # change x value so it ranges from 0-255
            print (x_send)
 # print the value being sent for performance tracking
            ser.write([x_send])
 # send the data through the serial port

    cv2.imshow('mask',mask)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


cap.release()
cv2.destroyAllWindows()
ser.close()
```

Figure A-3. Source code for the Arduino, tracking the tennis balls movement

```
/*
 * ME 545 Lab 5 - Tennis Ball Tracker
 * Receives coordinates from the Raspberry Pi about where the tennis ball is
 * Camera situated on servo moves accordingly to make tennis ball always in the center of the
frame
 */

 #include <Servo.h>
 Servo servoTrack;

int servoX; // initialize variable to store data coming in from serial monitor
int pos_track; // initialie variable for position of the servo

void setup() {
  Serial.begin(9600); // turn on the serial monitor and ensure baud rate is = to Pi

  servoTrack.attach(9); // attach servo to pin 9
  servoTrack.write(90); // set the servo to the middle (90º) on start up
}

void loop() {
  // if there's data on the serial buffer, read and store the data
  if (Serial.available() > 0) {
    servoX = Serial.read();
  }
  // map the values from the serial port to have a range of 0-180 (for the servo)
  pos_track = map(servoX, 255, 0, 180, 0);

  servoTrack.write(pos_track); // set the servo to the mapped position
  delay(55); // delay 55ms to allow servo to reach it's position

  Serial.println(servoX); // print the serial read data to the serial monitor to track performance
}
```