ME-GY 6933: Advanced Mechatronics
Spring 2020

Final Report

Motion Tracking Camera System

Haoran Wu, Haoran Zhou, Anderson Cone

14 May 2020

1. Introduction

The Raspberry Pi is a powerful electronics platform that enables easy use of hardware and software to create highly customizable mechatronics systems for productive and recreational use. Acting as a small self-contained computer, the Raspberry Pi can interface with a vast array of hardware components and can take advantage of many software libraries to interact with these hardware systems as well as with other microcontrollers. The Raspberry Pi, especially in conjunction with another microcontroller, is capable of performing nearly any digital or mechatronic task.

In this project, the group created a motion tracking system. The capabilities for such a system are numerous. The primary need--that which the group addressed directly through construction of the physical prototype detailed in this report--is that of a motion tracking camera. Many tasks may be completed when the user has a camera capable of tracking their motion and keeping them centered within the frame at all times. A recreational user of the system could easily take selfies with this system without having to worry about keeping him or herself in the frame. A professor could use this system to track their motion as they teach using a series of blackboards, automatically training the camera on the teacher and, by proxy, the board they are working on. The system could be installed in a home as a security device, capable of detecting intruders and tracking their motion through a room; Similarly, this device could be set up in a young child's room as a parental surveillance device, allowing parents to keep a close eye on the child remotely. Clearly, there are numerous use cases for such an autonomous motion tracking system outfitted with a camera.

With only minor adjustments to the motion tracking system, other hardware setups may be created on the same tracking base system. For example, a Nerf gun or similar harmless projectile launcher could be installed to create an interactive dodgeball-type game. A phone or tablet could be mounted on the system to keep the screen oriented towards the user while they make a video call or watch a TV show. Any object which a user may want to have continually pointing at them would benefit greatly from the capabilities of this system.

2. Background and Motivation

As was briefly mentioned in the Introduction, there exist many needs and use cases for motion tracking technology. Perhaps the strongest among them is the need for motion tracking technology in the classroom. Especially in this era of social distancing, the ability for professors teaching remotely to be able to conveniently show their students what they are working on as they write on a board or perform a demonstration is paramount. Often, when a lecture is being recorded in a classroom,

the professor must either frequently manually adjust the camera to ensure that the correct whiteboard is in the frame, or otherwise must hire an individual to operate a camera and perform this function. At present, this need is amplified as many professors find themselves teaching from home, or from otherwise empty school buildings. Thus, the need for a camera that follows their movement is also amplified.

With this need in mind, our group aimed to create a motion tracking camera which would keep the subject centered in the frame at all times. To achieve this goal, a system utilizing webcams, stepper motors, a Raspberry Pi, and an Arduino was created. One webcam is stationary and is fixed to the front of the prototype, providing the motion tracking capabilities to the system using OpenCV. Whenever a person enters the frame of this stationary webcam, the OpenCV software identifies their position and relays this information to the rest of the system. The second webcam is used to record and/or photograph the subject. This camera is mounted on a 2-axis swivel system, allowing it to pivot left-to-right as well as up-and-down, accurately maintaining the subject in the frame. These two webcam systems operate using the Raspberry Pi ecosystem.

The user--in this case, a professor--is also equipped with a handheld controller for manual operation of the camera setup. When the system is switched to Interactive (manual) Mode using the terminal, the Autonomous Mode motion tracking is switched off. The manual control system, run by an Arduino UNO, features four directional control buttons with which they may directly control the swivel of the recording camera to their liking. This information is relayed to the Raspberry Pi via serial communication, which then actuates the stepper motors to position the recording camera.

3. Methodology

A. Hardware Design

Table 3.A.a below shows the components used to create the prototype. Each component’s associated cost and the total cost of the project are shown as well.

Item #	Part Name	Quantity	Price (USD)
1	Cana Raspberry Pi 4 4GB Starter Kit	1	99.99
2	Arduino UNO board	1	23
3	Adafruit Stepper Motor	2	28
4	Adafruit Motor Hat for Raspberry Pi	1	22.5
5	USB Webcam 1080p	2	50
6	Voltage converter	1	6.5
7	Buttons	4	1

8	Wood board	2	4
9	Jump wires	N/A	N/A
10	Portable charger	1	18.77
11	Standoffs for Pi HATS	1	0.75
		TOTAL	242.53

Table 3.A.a: Component list and cost breakdown.

The physical design of the project can be divided into three main parts: the Raspberry Pi 4B board and Adafruit motor HAT, the Arduino UNO board, and the camera base.

An Adafruit motor HAT is connected to the Raspberry Pi board in order to supply power to two Adafruit NEMA-17 stepper motors. Each stepper requires 12V power, so the inclusion of the motor HAT simplifies the process of connecting an external power supply. To this end, a portable charger provides power to stepper motors through the voltage convertor. The wiring diagram for the Raspberry Pi half of the project is shown in Figure 3.A.1 below.

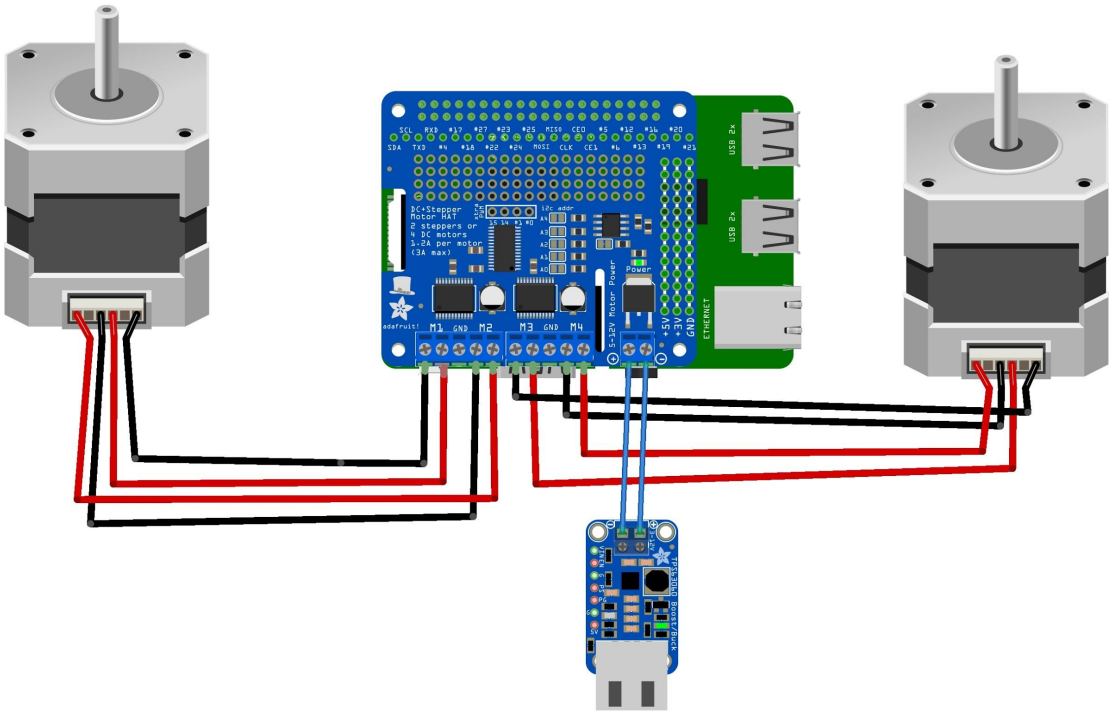

Figure 3.A.1: Raspberry Pi wiring diagram.

Two USB ports of the Raspberry Pi are utilized for this project. One USB port is connected to the Arduino UNO to perform the serial communication with the handheld user controls. The second USB port is connected to a USB hub which allows for

interfacing with the two FHD 1080p fisheye webcams used to track and record the user. The Raspberry Pi board, connected USB inputs, and motor HAT are shown in Figure 3.A.2 below.

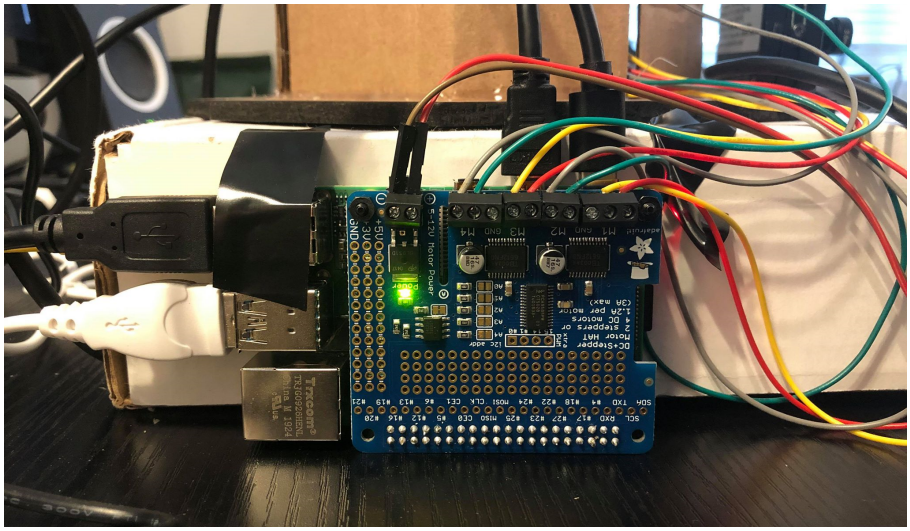

Figure 3.A.2: Raspberry Pi setup.

In the Arduino UNO portion of the system, a handheld controller gives the user the ability to position the camera manually. Four buttons move the camera up, down, left, and right while the system is in Interactive Mode. The wiring diagram for the Arduino half of the project is shown in Figure 3.A.3 below.

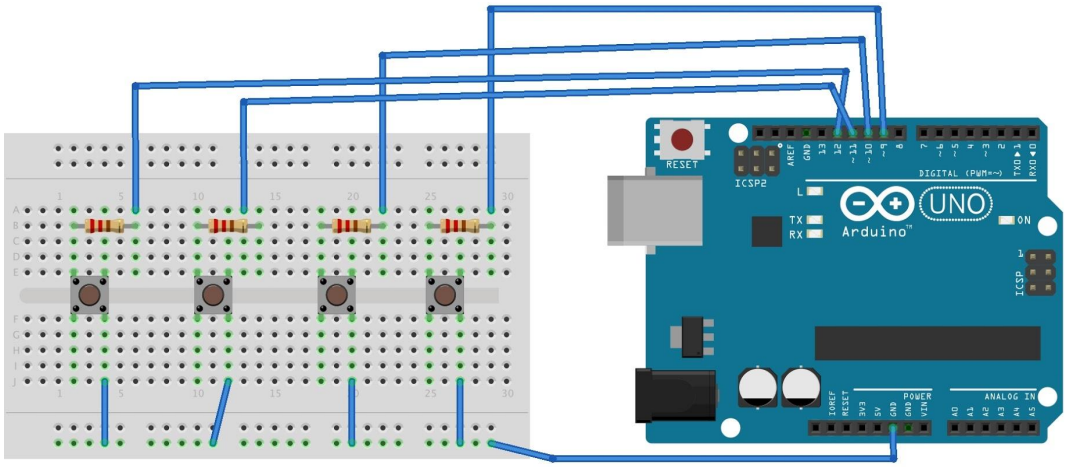

Figure 3.A.3: Arduino wiring diagram.

The USB port of the Arduino is linked to the Raspberry Pi USB port for serial communication between the two devices. The Arduino UNO microcontroller, USB connection, and handheld user controller are shown below in Figure 3.A.4, and the

complete setup with both Raspberry Pi and Arduino subsystems is shown in Figure 3.A.5 below.

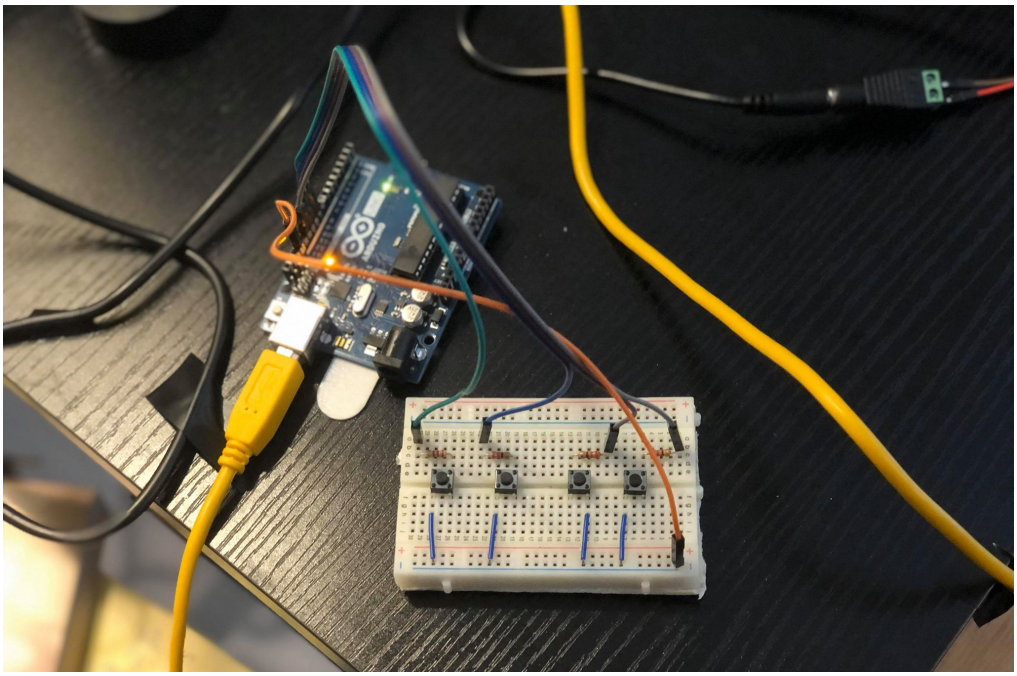

Figure 3.A.4: Arduino setup.

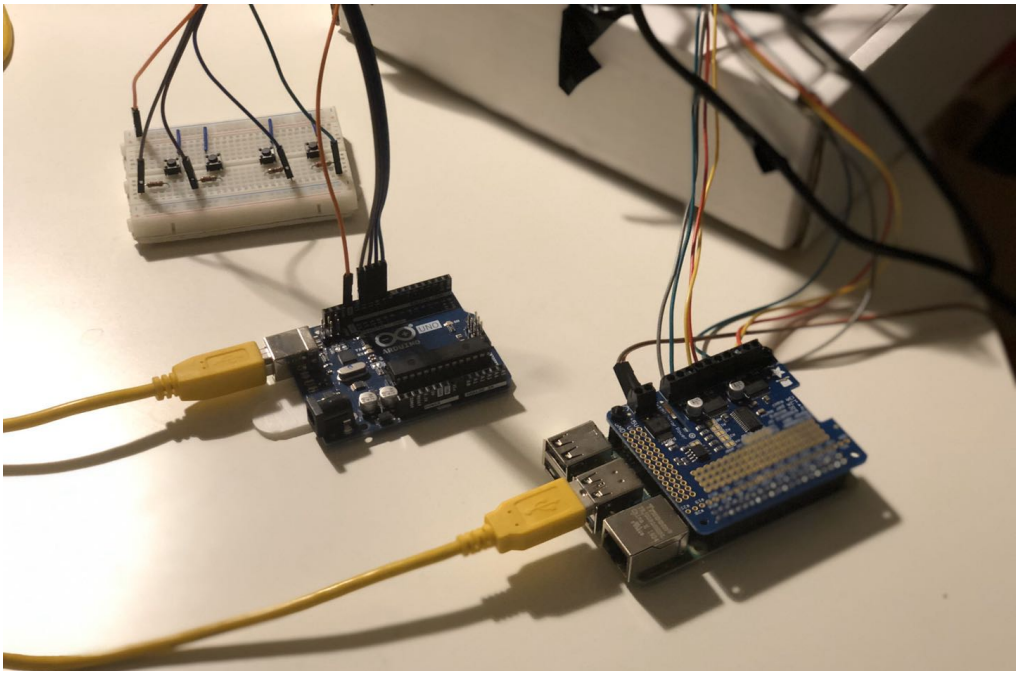

Figure 3.A.5: Complete setup.

The base system sits upon a cardboard box so as to be raised above the table surface, ensuring that the field of view of the motion sensing camera is not interrupted.

The stationary camera C1 is mounted on a wood plate. The first stepper (Motor A) is mounted on the wood plate to enable yaw rotation (horizontal, left-to-right motion) of the upper structure. The upper structure holds the second stepper (Motor B) which enables pitch rotation (vertical, up-and-down motion) of the recording camera C2. The base setup, with Motors A and B as well as Cameras F1 and F2, is shown in Figure 3.A.6 below.

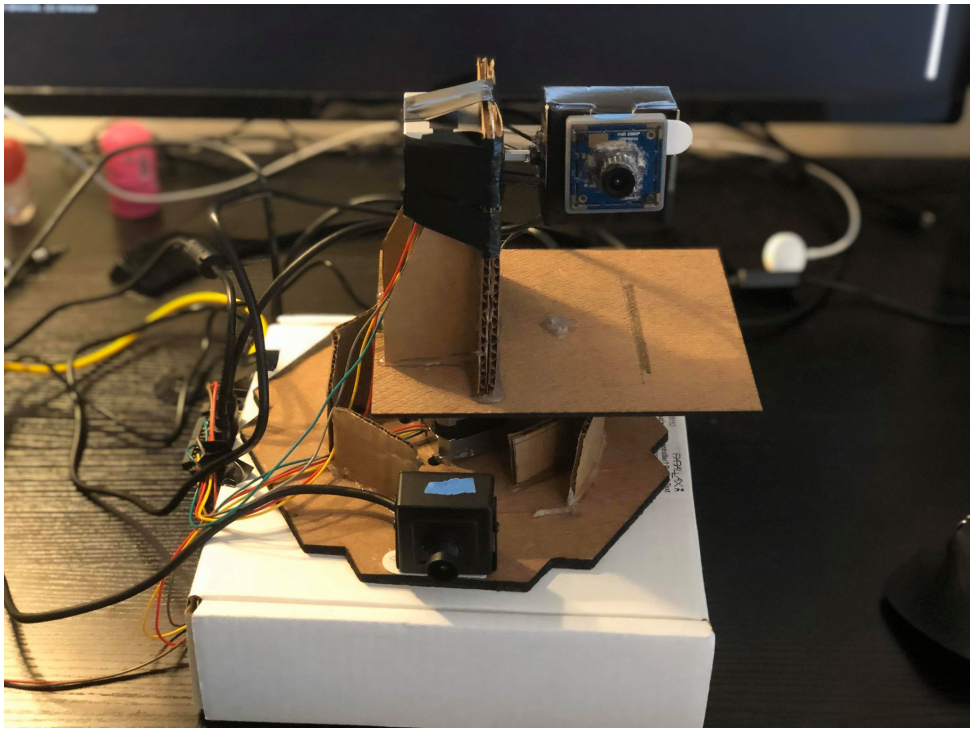

Figure 3.A.6: Base setup.

B. Code Programming

The code for this project is effectively split into two sections: the Autonomous Mode's motion tracking and the Interactive Mode's manual user control. Autonomous Mode was coded using Python3 and OpenCV4.3 environments. First, the pure motion tracking was coded to test the capabilities of the tracking system.

Both cameras and the servo motors are turned on. Stationary camera C1 captures the first frame of the video recording and the image is resized and Guassian blurred. This process is repeated for each frame. As consecutive frames are compared, the difference between the two frames is calculated using built-in OpenCV functions. This results in moving objects--notably the user--being marked with green rectangles for easy viewing on the computer monitor. This recording and calculating process repeats indefinitely until the user presses "Q" on the keyboard to quit the recording process.

With the motion tracking software in place, code was then created to actuate the stepper motors to keep the moving subject in the frame of moving camera C2. This was done using Adafruit stepper libraries compatible with Python3. As the OpenCV code detected a user moving to their right (when facing the camera), the yaw stepper Motor A was programmed to rotate the base assembly to the left to mirror this motion and keep the user in the frame. Similarly, if the subject moved up or down in the frame, the pitch stepper Motor B would move up or down to match this movement, as no mirroring is needed in the vertical direction. The power provided to each stepper motor varied, as yaw stepper Motor A required additional power due to the increased load created by the upper assembly. Pitch stepper Motor B required less power, as it only had to actuate the moving camera C2.

In Interactive mode, no OpenCV functionality was needed, as the whole system was switched to manual user control. The four breadboard-mounted buttons “Up”,”Down”,”Left”, and ”Right” were connected to the Arduino. The Arduino then converted the button inputs into binary signals and sent them to the Raspberry Pi through serial communication. The Raspberry Pi then actuated the steppers using these input signals.

The code for the Arduino and Raspberry Pi are attached in the report in the Appendix section. The following is a brief description of the functioning of the Arduino and Raspberry Pi programs. For the Arduino code, the button pins are defined in the setup, and the loop continuously sends the button signals to the Raspberry Pi through serial communication. For the Raspberry Pi code, all necessary libraries are imported; then, we classify the OpenCV video process, including capturing, resizing, blurring, comparing and marking to yield the video and the motion tracking capabilities; next, the output from the OpenCV motion tracking code was fed into the stepper motor code to actuate the motors to keep the subject in frame during Autonomous Mode. In Manual Mode, the Arduino output signals are fed into the motor code to actuate the motors to the user’s liking.

In conclusion, we programmed the motion tracking software using OpenCV, the motor movement based on the built-in Adafruit stepper library, serial communication between Arduino and Raspberry Pi, and the Arduino signal conversion and transmission functions.

4. Tests and Results

A. Motion Detection Test

In this first test, the motion detection capabilities of the system were tested using

only stationary camera C1. Camera C2 and motors A and B were not used in this test. Adequate lighting was produced to ensure proper function of the motion tracking algorithm. Upon initialization, no objects were moving in the frame. Then the test subject entered the frame and the repeated image capture of the algorithm detected this motion. Each moving object (his torso, legs, hand, and an erroneous signal created by a piece of furniture, in this case) was marked with a green rectangle. The test subject moved steadily around the frame and the motion detection code was found to operate as expected, maintaining the subject bound within a green box. A screenshot of this first test is shown in Figure 4.A.1 below. With this test successfully completed, the next test could begin.

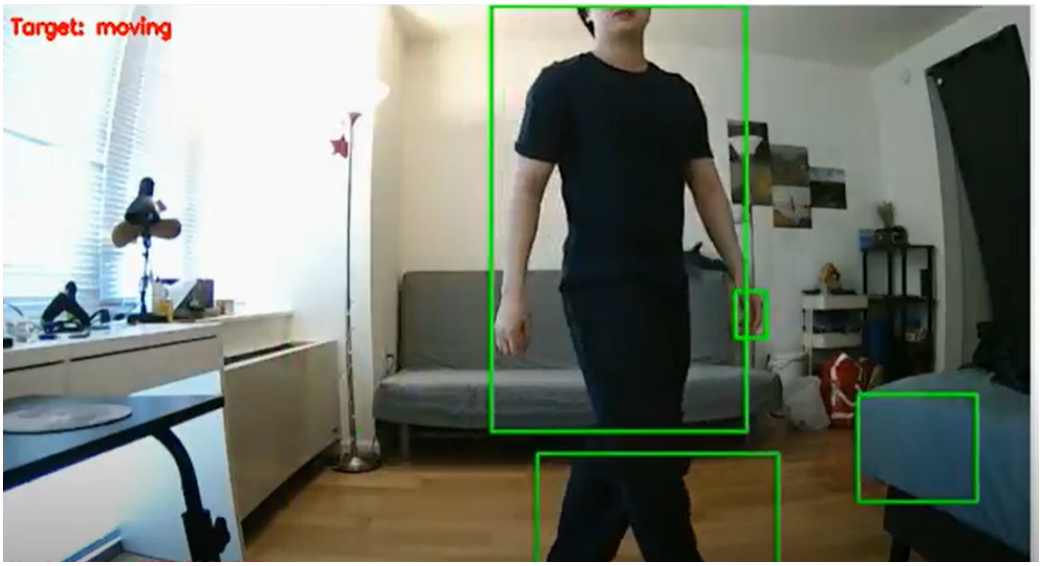

Figure 4.A.1: Motion detection visual output.

B. Interactive Mode Test

In this test, the handheld user controller, Arduino code, serial communication, and stepper motors were tested. Each button on the handheld controller was activated several times and the two stepper motors were seen to actuate appropriately, moving the tracking camera C2 up, down, left, and right as expected. Additionally, the output signals were mirrored on the terminal for further assurance of proper code functionality. With the knowledge that buttons pressed on the Arduino system properly actuated the stepper motors in the Raspberry Pi system, the test was successfully completed. A screenshot of this test being performed is shown in Figure 4.B.1 below. The output signals can be seen in the terminal on the let as the test operator presses the manual control buttons on the right.

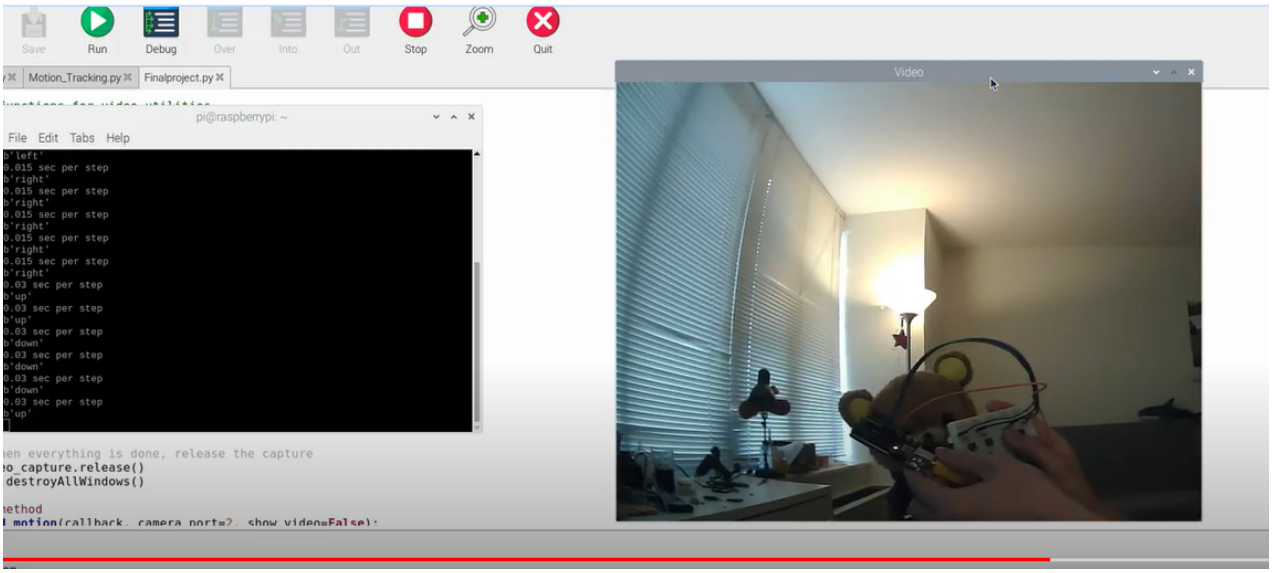

Figure 4.B.1: Interactive Mode test screenshot.

C. Motion Tracking Mode Test

In this test, the motion tracking capabilities were tested in conjunction with the stepper motor capabilities. As shown in Figure 4.C.1 below, the stationary camera C1 (shown in the top left) just began to track the test subject's movement as described in Test A, bounding the subject in the green box. The lower camera screen is the output from the moving camera C2, which has yet to move. In Figure 4.C.2, camera C1 has maintained its tracking on the test subject, and Motor 1 has yawed the system to the right to center the subject in the frame of moving camera C2. With these results, the test was successfully completed; the system was capable of autonomous motion tracking.

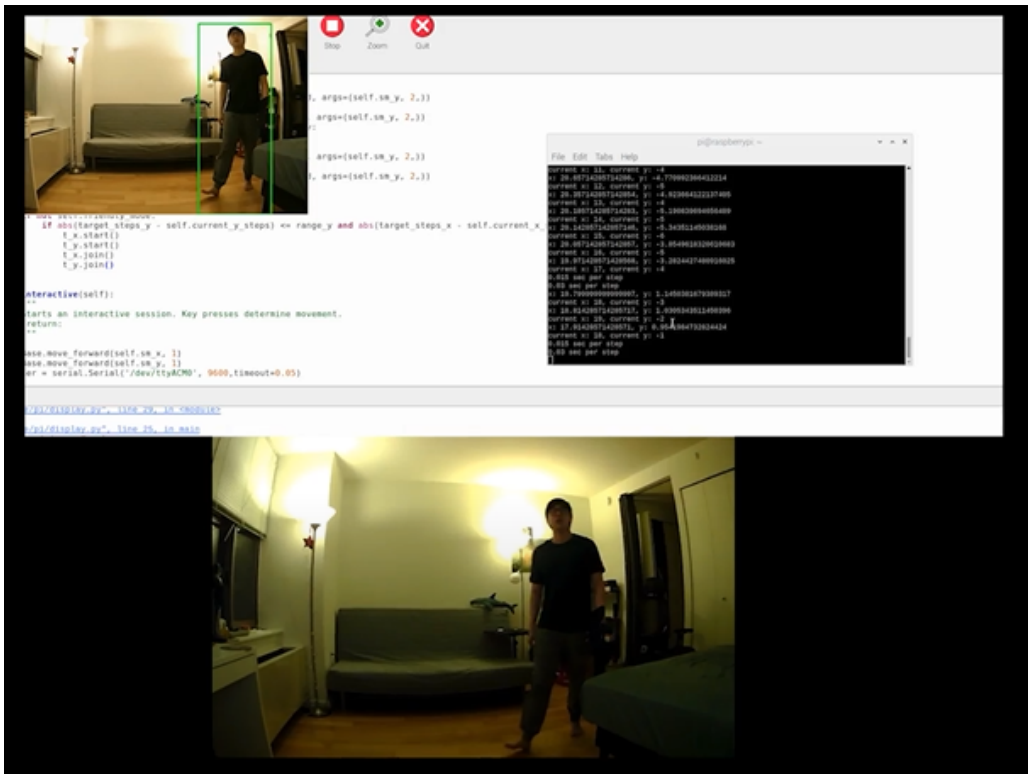

Figure 4.C.1: Motion tracking test screenshot before autonomous adjustment.

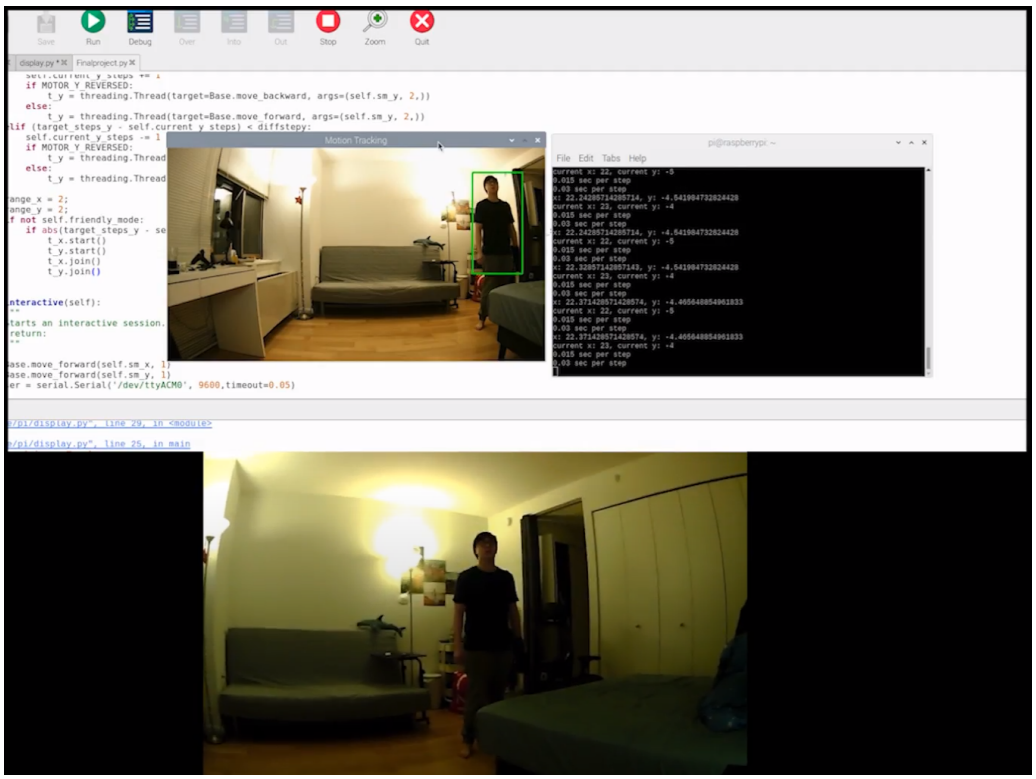

Figure 4.C.2: Motion tracking test screenshot after yawing right.

5. Conclusion and Future Work

In this Raspberry Pi and Arduino combined project, a motion tracking camera system was successfully created using USB webcams, stepper motors, OpenCV, and Raspberry Pi; serial communication between the Raspberry Pi and Arduino systems was also successfully created to enable manual user control of the camera system. All tests were completed successfully and all desired performance metrics were met.

The project shows strong potential in fields requiring the capability of switching between motion tracking and manual control including security (home security camera, childing monitoring) and entertainment (selfie, vlogging, and dodgeball-type games).

Given additional time and access to resources and tools, several improvements can be made to the projects form and function. First, the wood and cardboard structure can be replaced with a 3D printed custom housing designed using a CAD software such as solidworks. This would streamline the design of the project by allowing for custom mounting geometries and connections for the various components. This upgrade would also yield a more polished appearance than the rough, prototypical appearance of this current version. Second, replacing the moving camera C2 with a smartphone would be beneficial. Smartphone cameras are much more powerful than the simple webcam currently in use, and recording video and images on a smartphone has many more uses, given the connectivity of a phone. Thus, integration of the smartphone camera software with the Raspberry Pi/OpenCV environments would be very beneficial. Third, upgrading the Arduino-to-Raspberry Pi connection from wired serial communication to wireless RF communication would enable the user to roam around the room with greater freedom without having to worry about the length of their serial communication cable. Finally, upgrading the motion tracking algorithm would be highly beneficial, as the current algorithm is strongly negatively affected by poor lighting conditions. That is, the system struggles to accurately track moving objects/subjects when lighting is dim. Improving the reliability of the algorithm in these lighting conditions would improve user satisfaction.

6. Acknowledgements

We would like to express our sincere gratitude to Professor Kapila for providing his invaluable guidance, comments and suggestions throughout the semester and on this project. We would also like to thank our classmates for their questions, comments, and suggestions.

7. Appendix

Arduino Code

```
int left = 9;
int right = 10;
int up = 11;
int down = 12;

void setup() {
  Serial.begin(9600);
  pinMode(left,INPUT_PULLUP);
  pinMode(right,INPUT_PULLUP);
  pinMode(up,INPUT_PULLUP);
  pinMode(down,INPUT_PULLUP);
}

void loop() {
   if(Serial.available()){
      if (digitalRead(left) == 0){
        Serial.print("left");
        delay(300);
    }
      if (digitalRead(right) == 0){
        Serial.print("right");
        delay(300);
    }
    if (digitalRead(up) == 0){
        Serial.print("up");
        delay(300);
    }
      if (digitalRead(down) == 0){
        Serial.print("down");
        delay(300);
    }
 }
}
```

Raspberry Pi Code

```
try:
    import cv2
except Exception as e:
    print("Warning: OpenCV not installed. To use motion detection, make sure you've properly configured OpenCV.")

import time
import _thread
import threading
import atexit
import sys
import termios
#import contextlib
import serial

import imutils
import RPi.GPIO as GPIO
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor, Adafruit_StepperMotor

### User Parameters ###
```

```
MOTOR_X_REVERSED = False
MOTOR_Y_REVERSED = False

MAX_STEPS_X = 30
MAX_STEPS_Y = 15


########################
class VideoUtils(object):
    """
    Helper functions for video utilities.
    """
    @staticmethod
    def live_video(camera_port=0):
        """
        Opens a window with live video.
        :param camera:
        :return:
        """

        video_capture = cv2.VideoCapture(camera_port)
        video_capture.set(cv2.CAP_PROP_FRAME_WIDTH,800)
        video_capture.set(cv2.CAP_PROP_FRAME_HEIGHT,800)
        video_capture.set(cv2.CAP_PROP_FPS,60)

        while True:
            # Capture frame-by-frame
            ret, frame = video_capture.read()

            # Display the resulting frame
            cv2.imshow('Video', frame)

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

        # When everything is done, release the capture
        video_capture.release()
        cv2.destroyAllWindows()

    @staticmethod
    def find_motion(callback, camera_port=3, show_video=False):

        camera = cv2.VideoCapture(camera_port)

        time.sleep(0.1)

        # initialize the first frame in the video stream
        firstFrame = None
        tempFrame = None
        count = 0

        # loop over the frames of the video
        while True:
```

```
# grab the current frame and initialize the occupied/unoccupied
# text

(grabbed, frame) = camera.read()

# if the frame could not be grabbed, then we have reached the end
# of the video
if not grabbed:
    break

# resize the frame, convert it to grayscale, and blur it
frame = imutils.resize(frame, width=700)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)

# if the first frame is None, initialize it
if firstFrame is None:
    print ("Waiting for video to adjust...")
    if tempFrame is None:
        tempFrame = gray
        continue
    else:
        delta = cv2.absdiff(tempFrame, gray)
        tempFrame = gray
        tst = cv2.threshold(delta, 5, 255, cv2.THRESH_BINARY)[1]
        tst = cv2.dilate(tst, None, iterations=2)
        if count > 30:
            print ("Done.\n Waiting for motion.")
            if not cv2.countNonZero(tst) > 0:
                firstFrame = gray
            else:
                continue
        else:
            count += 1
            continue

# compute the absolute difference between the current frame and
# first frame
frameDelta = cv2.absdiff(firstFrame, gray)
thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

# dilate the thresholded image to fill in holes, then find contours
# on thresholded image
thresh = cv2.dilate(thresh, None, iterations=2)
c = VideoUtils.get_best_contour(thresh.copy(), 5000)

if c is not None:
    # compute the bounding box for the contour, draw it on the frame,
    # and update the text
    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    callback(c, frame)

# show the frame and record if the user presses a key
```

```
            if show_video:
                cv2.imshow("Motion Tracking", frame)
                key = cv2.waitKey(1) & 0xFF

                # if the `q` key is pressed, break from the lop
                if key == ord("q"):
                    break

        # cleanup the camera and close any open windows
        camera.release()
        cv2.destroyAllWindows()

    @staticmethod
    def get_best_contour(imgmask, threshold):
        contours, hierarchy = cv2.findContours(imgmask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        best_area = threshold
        best_cnt = None
        for cnt in contours:
            area = cv2.contourArea(cnt)
            if area > best_area:
                best_area = area
                best_cnt = cnt
        return best_cnt

class Base(object):
    """
    Class used for Base control.
    """
    def __init__(self, friendly_mode=True):
        self.friendly_mode = friendly_mode

        # create a default object, no changes to I2C address or frequency
        self.mh = Adafruit_MotorHAT()
        atexit.register(self.turn_off_motors)

        # Stepper motor 1
        self.sm_x = self.mh.getStepper(400, 1)      # 200 steps/rev, motor port #1
        self.sm_x.setSpeed(10)                      # 5 RPM
        self.current_x_steps = 0

        # Stepper motor 2
        self.sm_y = self.mh.getStepper(200, 2)      # 200 steps/rev, motor port #2
        self.sm_y.setSpeed(5)                       # 5 RPM
        self.current_y_steps = 0

    def motion_detection(self, show_video=False):
        """
        Uses the camera to move the Base. OpenCV ust be configured to use this.
        :return:
        """
        VideoUtils.find_motion(self.__move_axis, show_video=show_video)

    def __move_axis(self, contour, frame):
        (v_h, v_w) = frame.shape[:2]
```

```
        (x, y, w, h) = cv2.boundingRect(contour)

        # find height
        target_steps_x = (2*MAX_STEPS_X * (x + w / 2) / v_w) - MAX_STEPS_X
        target_steps_y = (2*MAX_STEPS_Y*(y+h/2) / v_h) - MAX_STEPS_Y

        print ("x: %s, y: %s" % (str(target_steps_x), str(target_steps_y)))
        print ("current x: %s, current y: %s" % (str(self.current_x_steps), str(self.current_y_steps)))

        t_x = threading.Thread()
        t_y = threading.Thread()

        # move x
        diffstepx = 0;
        if (target_steps_x - self.current_x_steps) > diffstepx:
            self.current_x_steps += 1
            if MOTOR_X_REVERSED:
                t_x = threading.Thread(target=Base.move_forward, args=(self.sm_x, 2,))
            else:
                t_x = threading.Thread(target=Base.move_backward, args=(self.sm_x, 2,))
        elif (target_steps_x - self.current_x_steps) < diffstepx:
            self.current_x_steps -= 1
            if MOTOR_X_REVERSED:
                t_x = threading.Thread(target=Base.move_backward, args=(self.sm_x, 2,))
            else:
                t_x = threading.Thread(target=Base.move_forward, args=(self.sm_x, 2,))

        # move y
        diffstepy = 0;
        if (target_steps_y - self.current_y_steps) > diffstepy:
            self.current_y_steps += 1
            if MOTOR_Y_REVERSED:
                t_y = threading.Thread(target=Base.move_backward, args=(self.sm_y, 2,))
            else:
                t_y = threading.Thread(target=Base.move_forward, args=(self.sm_y, 2,))
        elif (target_steps_y - self.current_y_steps) < diffstepy:
            self.current_y_steps -= 1
            if MOTOR_Y_REVERSED:
                t_y = threading.Thread(target=Base.move_forward, args=(self.sm_y, 2,))
            else:
                t_y = threading.Thread(target=Base.move_backward, args=(self.sm_y, 2,))

        range_x = 2;
        range_y = 2;
        if not self.friendly_mode:
            if abs(target_steps_y - self.current_y_steps) <= range_y and abs(target_steps_x - self.current_x_steps) <=
range_x:
                t_x.start()
                t_y.start()
                t_x.join()
                t_y.join()


    def interactive(self):
```

```
"""
Starts an interactive session. Key presses determine movement.
:return:
"""

Base.move_forward(self.sm_x, 1)
Base.move_forward(self.sm_y, 1)
ser = serial.Serial('/dev/ttyACM0', 9600,timeout=0.05)

n=1
print ('use button to control base')
try:
   while(n==1):
      #print("asd")
      ser.write(str.encode('start'))
      response = ser.readall()
      step1 = 5
      step2 = 10

      if response == b'up':
         if MOTOR_Y_REVERSED:
            Base.move_forward(self.sm_y, step1)
            print(response)

         else:
            Base.move_backward(self.sm_y, step1)
            print(response)

      elif response == b'down':
         if MOTOR_Y_REVERSED:
            Base.move_backward(self.sm_y, step1)
            print(response)

         else:
            Base.move_forward(self.sm_y, step1)
            print(response)

      elif response == b'left':
         if MOTOR_X_REVERSED:
            Base.move_backward(self.sm_x, step2)
            print(response)

         else:
            Base.move_forward(self.sm_x, step2)
            print(response)

      elif response == b'right':
         if MOTOR_X_REVERSED:
            Base.move_forward(self.sm_x, step2)
            print(response)

         else:
            Base.move_backward(self.sm_x, step2)
            print(response)
```

```
                #print("das")
            except:
                ser.close()
                pass

    @staticmethod
    def move_forward(motor, steps):
        """
        Moves the stepper motor forward the specified number of steps.
        :param motor:
        :param steps:
        :return:
        """
        motor.step(steps, Adafruit_MotorHAT.FORWARD,  Adafruit_MotorHAT.INTERLEAVE)

    @staticmethod
    def move_backward(motor, steps):
        """
        Moves the stepper motor backward the specified number of steps
        :param motor:
        :param steps:
        :return:
        """
        motor.step(steps, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.INTERLEAVE)

    def turn_off_motors(self):
        """
        Recommended for auto-disabling motors on shutdown!
        :return:
        """
        self.mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
        self.mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
        self.mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
        self.mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

if __name__ == "__main__":
    t = Base(friendly_mode=False)

    user_input = input("Choose an input mode: (1) Motion Detection, (2) Interactive\n")

    if user_input == "1":

        if input("Live video? (y, n)\n").lower() == "y":
            t.motion_detection(show_video=True)
        else:
            t.motion_detection()
    elif user_input == "2":
        if input("Live video? (y, n)\n").lower() == "y":
            _thread.start_new_thread(VideoUtils.live_video, ())
        t.interactive()
    else:
        print ("Unknown input mode. Please choose a number (1) or (2)")
```