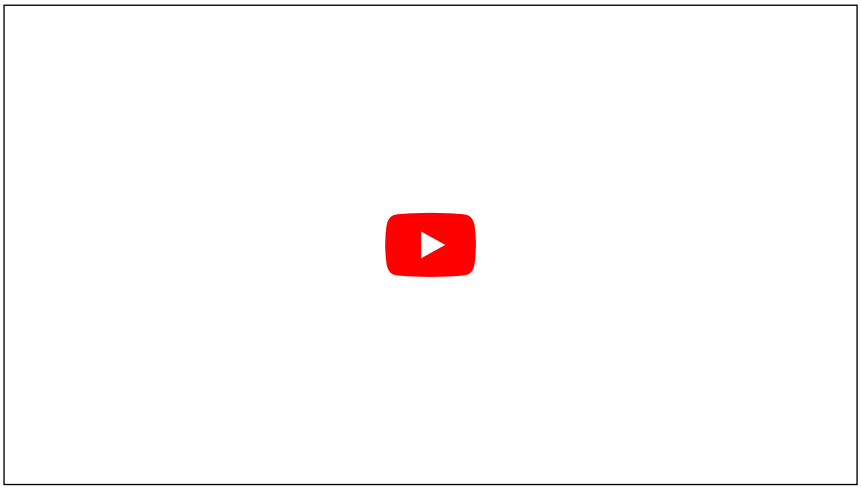
Pi Billiard Assistant

Provide visual shooting aid for billiard beginners
An ECE5725 Project By Zim Gong and Yixin Zang.



Demonstration Video

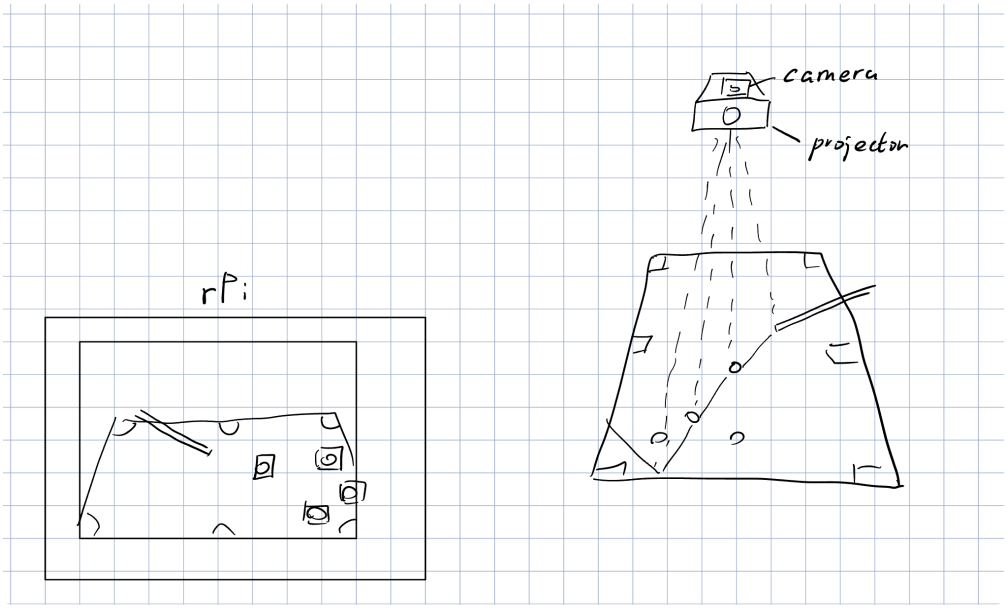
Introduction:

The Pi Billiard Assistant is an embedded system that consists of a Raspberry Pi 4, a Raspberry camera and an external laptop providing a dynamic trajectory estimation based on player"s cue direction. The system is capable of calculating the trajectory with a collision between cue ball and cushions. Players can better adjust the shot by referring to the trajectory and explore more potential solutions to snooker from laptop display. The system is installed with an external frame for video capturing and leaves space for future stretch designs and implementations.

Objective:

- Identify the position of cue stick, determine the aiming direction
- Identify the table and establish the boundary to detect any occurrences of collision and calculate the trajectory with reflection
- Dynamically change the trajectory in real-time to optimize user experience
- The system is generally portable and has a competitive cost

Drawing and Demo Setup



Initial Drawing

Frame for Demo

Design

Hardware

The external hardware components include four parts: a Raspberry Pi 4 as the central processing unit, a Raspberry camera to catch the video of Billiards, a laptop to display the trajectory video and a projector that could project the trajectory line to the table.

In order to hold the components, a frame was required. The team measured the dimension of the pool table, which is 44 inches by 88 inches. Then the minimal distance from the table to the camera to catch the whole table and verified based on a mathematical equation. The laboratory provided a 6 feet 2 inches height frame. A 3.2 feet and 3 inches wooden stripe was appended to the frame by drilling holes and fastening screws. Thus, an inverted L-shape frame was built for holding Raspberry Pi 4 and the camera. The Raspberry Pi 4 was taped on the top of the arm and the camera was stuck underneath the arm. Additionally, a bailed button was implemented on a breadboard as a physical exit button to stop the system securely and attached to the side of the arm. A power stripe was added to power the Raspberry Pi and was treated as a counterweight which further stabilized the system. Alternatively, a power bank could be used as the power source.

Table and Cue Stick Detection

The general idea of table detection was to implement a color mask to filter out the same color as the table, which provided a contour of the pool table. The mask filtered out the irrelevant information from the video including surroundings, outer/wooden part of the table. A clean table capture is the foundation of the image processing. The color limitation of the camera and normal indoor lights significantly affected the color detection from OpenCV RGB color scheme. Therefore, HSV color scheme was applied since HSV is a user oriented scheme which was more flexible in later calibration. To distinctively display the boundaries of the pool table, the team utilized OpenCV to generate a Convex hull of the counter. The convex hull contains the boundary coordinate information for later physics calculation.

Similarly, the cue detection implemented the HSV for white color as the most of cues appear white luster. Through OpenCV, we further generated a Hough Line to identify the cue stick within radius, length constraints. To simulate multiple cue positions, we reduce the length of the line by half for each iteration until the cue is detected. It guarantees the length of the cue within the camera does not affect the cue recognition. Before we draw the trajectory of the cue, we need to first determine the aiming direction. After detecting the cue, we do not have the direction information. By introducing Euclidean distance, we calculated the distance between the endpoints of the line and the center of the table. We set the one closed to the center as the aiming direction. The trajectory can be regarded as an extended cue stick if there is no boundary collision. We set a cue simulation to predict the estimated line that the ball would travel

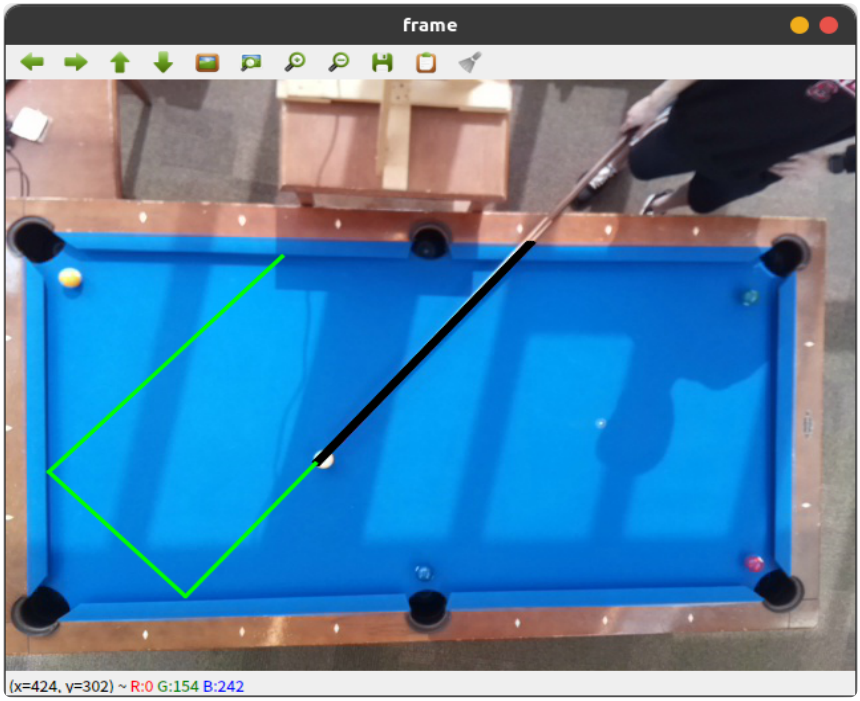
Simulation Physics

The project uses a self-developed physics system to monitor the moving path of the tip of the stick and print out trajectories accordingly.

The simulation is implemented by tracking the tip of the cue stick. A moving vector is assigned to the tip from the detected cue stick line. Its length is regularized by dividing its euclidean length. THe tip is simulated for at most 100 steps until it hits the table boundary.

The table boundary is defined by the convex hull found before. It is characterized by all its edges. To detect a collision, the point has to be computed with all different edges. If one of the edges shows collision, its vector is going to be r eturned and the point changes its speed vector according to the returned vector.

Note that the detected table is cropped out by color, which includes the part that is out of the table boundary. To fix this issue, collision detection compares the point two steps from the current position with the boundaries.

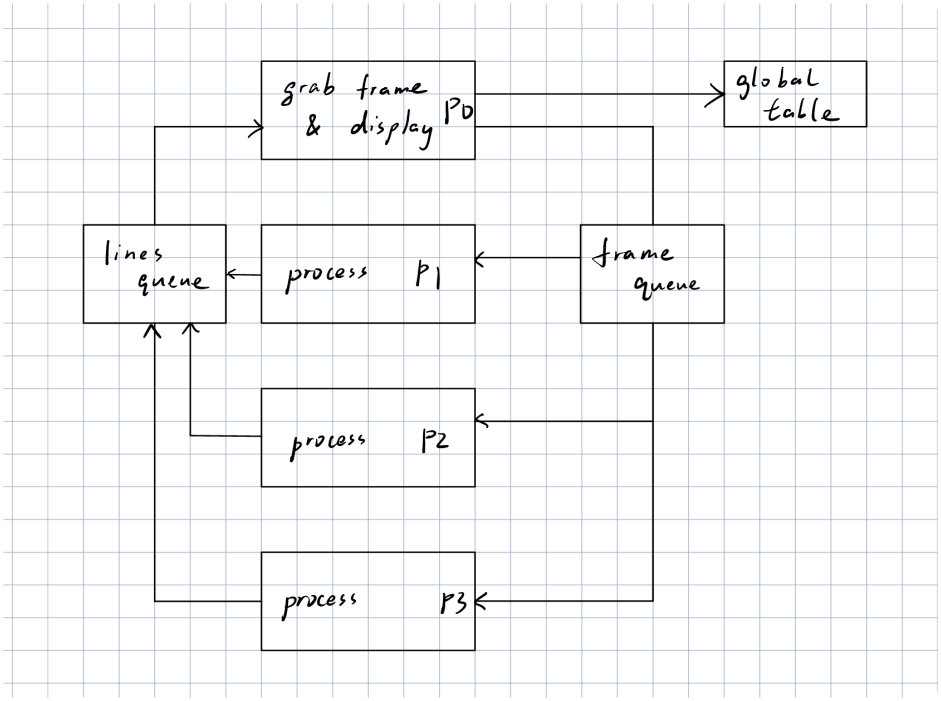


Sample Trajectory Result

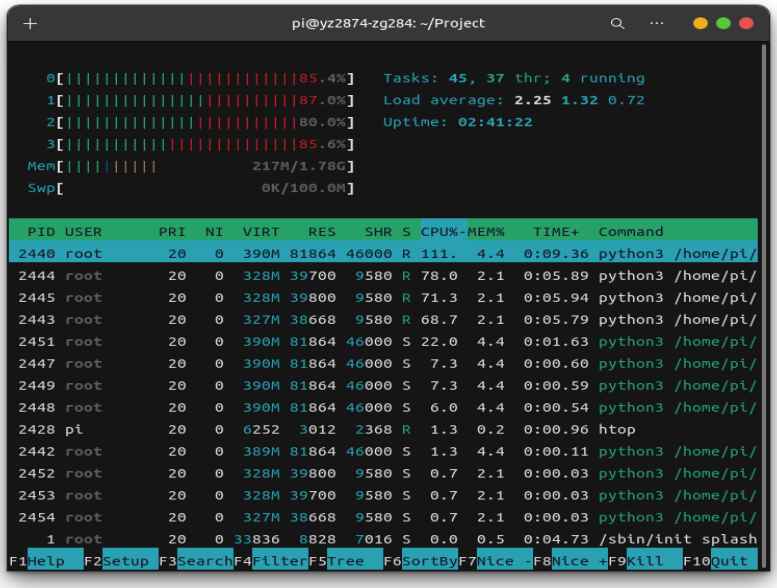
Multiprocessing

The system used python multiprocessing library to achieve higher CPU utilization and get higher frame rate. Initially, the CPU usage was not satisfying. The utilization rate was around 60% each core and varies with time. It resulted in great latency with the input video and could only run with 5 FPS. In comparison, after reorganizing the code with multiprocessing, the CPU utilization achieved more than 80% each core all the time and can even go up to 100%, making 30FPS frame input possible.
The multiprocessing algorithm separates the system into the 4 following processes.The processes communicate with each other through queues.

- Process 0 is the main running process. It grabs frames from the camera, sends them into the frame queue, receives predicted trajectories from the line queue, renders them to the frame and sends it out to the communication channel with the laptop. It is also responsible for detecting the table boundaries during boot up and sending it to other cores.
- Process 1, 2 and 3 are parallel working processes. They have the same job of detecting the cue stick from frame, computing physics and asserting trajectories.



New Schematic



CPU Utilization with Multiprocessing

Pi & Laptop Communication

The Raspberry Pi communicates with the laptop through both zmq channel and ssh.

- zmq channel is used to transfer rendered frames from the Raspberry Pi to the laptop. It connects through "tcp://" + laptop IP + ":5555" and sends frames encoded with the base64 library. On the other hand, the laptop receives frames from the channel, decodes and renders it with OpenCV.
- Since the Pi is fitted high above the table, there requires a method to start and kill the processes on Pi remotely. This is achieved through ssh with public key bindings between the Pi and the laptop. The laptop sends commands to the Pi in bash scripts which asks for ssh access. With the public key, the Pi would not ask for the user password every time a bash script is running.

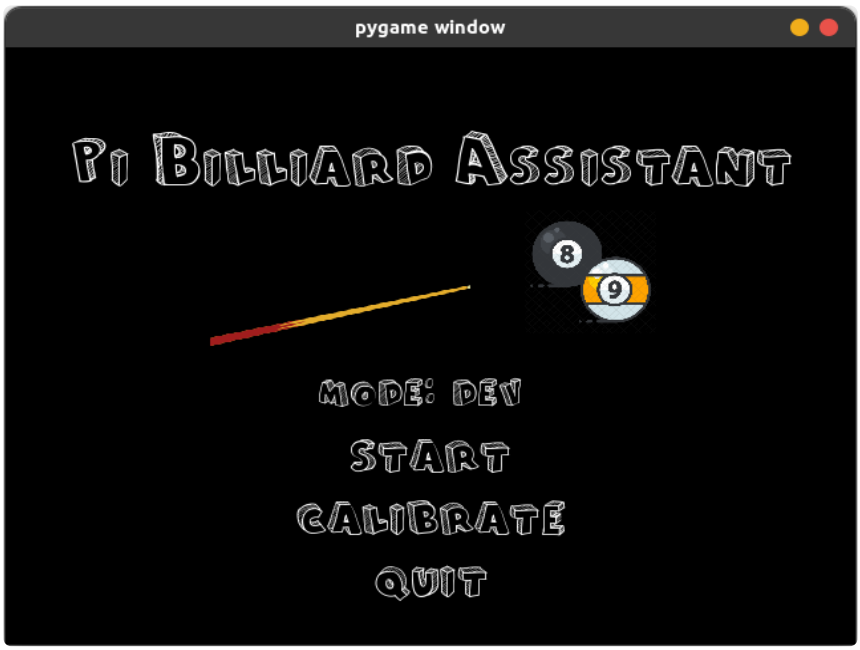
Graphical User Interface

The GUI is built with Pygame that enables running the program in different modes and tweaking parameters for different tables.

The program can be run in user mode and developer mode. In user mode, only the predicted

trajectories are displayed, while in developer mode, the table boundaries are also displayed in combination with command line outputs of workflow.

The program also supports calibration for table color and stick sensitivity. The user could choose from a blue or green table, which are two most commonly used table colors. They could also tweak the sensitivity parameter of cue stick detection. If the program prints unreasonable trajectories, tuning the value down could help decrease false line recognitions. If the program is not recognizing the cue stick, tuning the value up would help the system identify it.



Graphical User Interface

Dev Notes and Issues

Projector Issues

At the beginning of the project, the team"s objective was to implement a projector and print the estimated trajectory on the pool table. The team set requirements of the projector including portability, resolution, and cost. The team chose a mini projector that could be powered by a 5 volts, 2.5 amps input. In the first testing, the projector was able to project Linux desktop on the wall. After completing the image processing, the team attempted to project the trajectory result on the pool table. However, due to strong light conditions, the projector did not have sufficient power to display a clear image to the pool table. The image was dim and the trajectory was undetectable by the human eyes. So, the team decided to change the objective to show the trajectory on the laptop which ensures the trajectory can be identified clearly. After discussion, the team settle the projector as a stretch feature that will be added and validated in the future prototype

Shape and Pink Detection Issues

During the first two week, the team was exploring the optimal strategy for the cue and cue ball detection. The team implemented two different detections for cue ball and cue. Hough circle detection was used for balls. After testing, the algorithm identified the player's hand as a circle for the most of the time and did not recognize some cue ball as a circle. The rate of misclassification and misidentification would cause troublesome issues in later image processing. The team chose to discard the algorithm for accuracy concern.

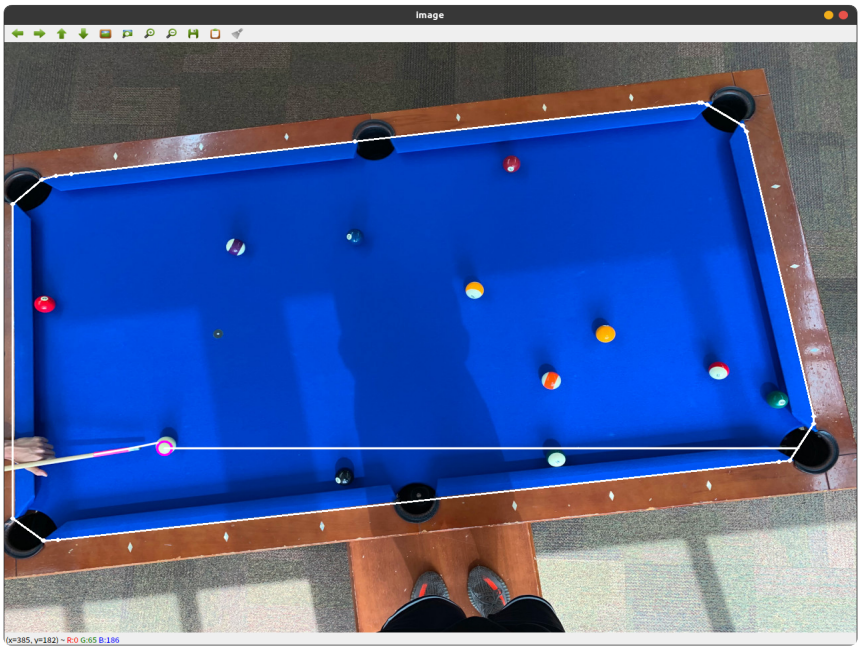
The cue detection was an arduous issue. In the first algorithm, the team applied OpenCV line detection. It generated multiple lines for one frame in all the testing cases, which could not be used

undoubtedly. The team wrapped pink duct tape around the tip of the cue. However, the pink recognition was not consistent enough. The pink color detection was hard coded multiple times and there was no optimal result in the end.

Cue Ball Detection

Initially, the team intended to also detect the cue ball. With the location and radius information of the cue ball, the trajectory can be more specified based on the player"s hitting angles. However, two issues forced the team to give up the plan.

- For faster computational speed, the Pi is using a resolution of 480p. Under this resolution, the radius of the ball is around 5-6 pixels. Furthermore, the OpenCV hough circle function only recognizes radius in integers. However, computing complex angular collision requires much more accuracy than the function could provide.
- The Pi"s computing power is rather limited. Detecting the cue ball costs approximately the same computing power with cue stick detection. The team could achieve twice detection speed with cue ball detection deleted, which is a significant performance improvement



Cue Ball Detection Sample

OpenCV in C++

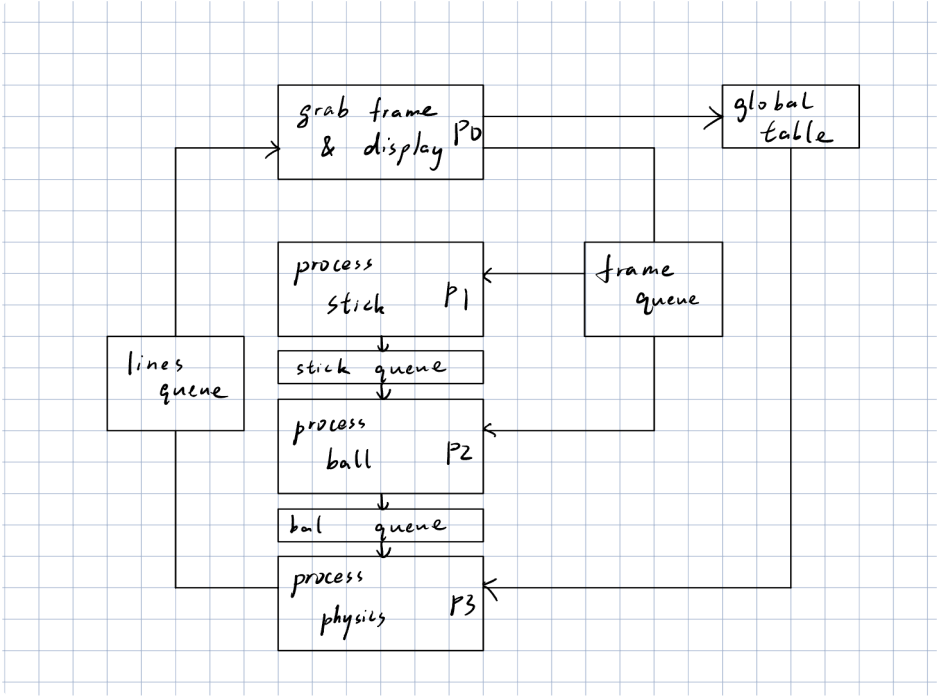
The team tried to use OpenCV in C++ to improve system performance. However, the compiling process of OpenCV is too complex and prone to errors. The team did not make a successful installation on an Ubuntu laptop because unsolvable errors keep popping up. There is reason to believe that installation on Raspberry Pi could be even more time costly.

Complex Multiprocessing Schematic

The team experimented with two kinds of multiprocessing logic and only one of them proved successful.

The first logic was a chain-like workflow. The processor goes one after each other to do frame grabbing, stick detection, physical computation and line rendering. However, it brought about two issues. First, it is hard to communicate between different processes in python multiprocessing. The processes are hard to synchronize and do not produce a stable workflow in queues. Second, python multiprocessing does not have a user-friendly way of debugging. Using a system with multiple queues is very likely to encounter quitting issues where the script does not return to the terminal after termination.

The other multiprocessing logic is to make use of parallelism, where 3 cores act as working processes and one core acts as the main process to handle the workflow. This logic works well and is documented in the part before.



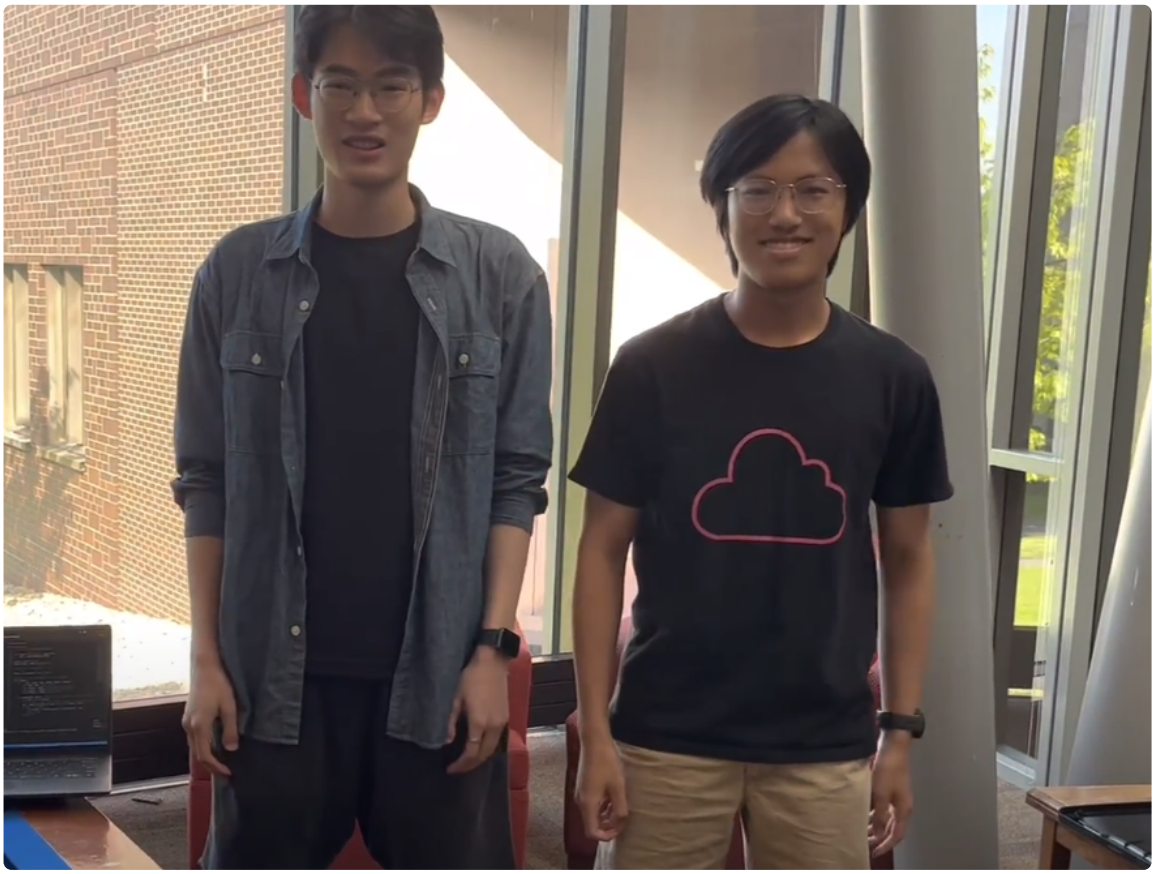
Old Schematic

Result and Future Work

The team is satisfied with the outcome of this project. It has fast real time response and is quite useful in making some shots with wall reflection. The project also comes with a low cost, requiring only a Raspberry Pi board and a Rpi camera. It has the potential to be a budget competitor among the current billiard aiding systems on the market.

However, due to the limited development time, the project is only a rough demo and could be improved in different aspects. First, the detection accuracy can be even better if more tests can be made to optimize the parameters. Second, the project can be exported to exe or moved to mobile phone platforms like android or ios to make it even more versatile.

Work Distribution



Project group picture

Yixin Zang

yz2874@cornell.edu

Built hardware components and evaluated the system

Zim Gong

zg284@cornell.edu

Designed and implemented the software of the system

Parts List

- Raspberry Pi 4 Model B - 2 GB RAM $45.00
- Raspberry Pi Camera Module 3 Standard - 12MP Autofocus $25.00
- Buttons, Resistors and Wires - Provided in lab

Total: $70.00

References

OpenCV-Python Tutorials (https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html)
multiprocessing — Process-based parallelism (https://docs.python.org/3/library/multiprocessing.html)
Getting started with the Camera Module (https://projects.raspberrypi.org/en/projects/getting-started-with-picamera)
Reflection Physics Reference
(http://www.sunshine2k.de/articles/coding/vectorreflection/vectorreflection.html#:~:text=Reflection%20in%20a%20nutshell%3A&text=The%20formula%20w%20%3D%20v%2(
R-Pi GPIO Document (https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/)

Code Appendix

Only the two main functions are appended here. For the entire project, please refer to the following github link.

ECE5725-Pi-Billiard-Assistant (”https://github.com/ZimG386/ECE5725-Pi-Billiard-Assistant)
raspi_main.py

```
#
# W_yz2874_zg284 5/9/2023 Raspberry Pi Main Script
#
# Description:
# This script is the main script for the raspberry pi. It will grab frames from the camera,
# detect the pool table, and send the table to the queue. It will also receive the lines
# from the queue and render them on the frame.
# The script uses multiprocessing to accelerate the process.
# The script also uses zmq to communicate with the laptop to send frames.
#


import argparse
import base64
import copy
import cv2 as cv
from datetime import datetime
import json
from multiprocessing import Process, Queue, Value
import numpy as np
from scipy.spatial import ConvexHull
import time
import zmq


parser = argparse.ArgumentParser(description='Main script for pool game detection on raspberry pi.')
parser.add_argument('--mode', type=str, help='user/developer mode', default='user')
args = parser.parse_args()


# Define a class for balls and the move function
class Object:
def __init__(self, pos, speed, direct, radius):
    self.pos = pos # Ball position
    self.speed = speed # Ball speed, not used yet
    self.direct = direct # Ball direction vector
    self.radius = radius # Ball radius


def move(self, hull): # Move the ball based on its direction vector
    self.pos += self.direct
    # If the ball is out of the table, change its direction, not yet implemented
    res = point_in_hull(self.pos + 2 * self.direct, hull)
    if res is not None:
        self.direct = collide_hull(self.direct[0:2], res)
        return False
    return True


# Check if a point is inside the convex hull
# Return the normal vector of the plane that the point is outside the hull
# Return None if the point is inside
def point_in_hull(point, hull, tolerance=1e-12):
    res = 0
    for eq in hull.equations:
        res = np.dot(eq[:-1], point) + eq[-1]
        if res >= tolerance:
            return eq[0:2]
    return None


# Calculate the new direction vector after collision
def collide_hull(direct, eq):
res = direct-2*np.dot(direct, eq)*eq
return  res


# Simulate the cue stick behavior, hits the cue ball and let the cue ball move
# Return the lines that the cue stick has traveled
def simulate_stick(object1, num_iter, hull, lines, flag):
while flag > 0:
    iter = 0
    ini_pos = copy.deepcopy(object1.pos)
    while iter <= num_iter:
        res = object1.move(hull)
        if res == False:
            flag -= 1
            lines.append([ini_pos[0], ini_pos[1], object1.pos[0], object1.pos[1]])
            break
return lines
```

```
# Find the pool table contour
# Return the cv.convexHull object of the table
# Return None if no table is found
def find_table(frame_hsv, color):
# hsv color range for blue pool table
lower_blue = np.array([color[0],120,120])
upper_blue = np.array([color[1],255,255])
# Mask out everything but the pool table (blue)
mask = cv.inRange(frame_hsv, lower_blue, upper_blue)
# cv.imshow("cropped table", mask)
# Find the pool table contour
contours = []
contours, _ = cv.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_NONE)
# Find the largest contour as the border of the table
if contours != []:
    table = max(contours, key = cv.contourArea)
    return cv.convexHull(table)
else:
    return None


# Master process, grab frames from camera and show frames with rendered lines
def grab_frame_display(run_flag, frame_queue, line_queue, table_queue, dev, color):
IP = '10.48.155.12'
context = zmq.Context()
footage_socket = context.socket(zmq.PAIR)
footage_socket.connect('tcp://' + IP + ':5555')
start_datetime = datetime.now() # Initialize start time
last_receive_time = 0 # Initialize last receive time
initial = True # Flag for first frame
cap = cv.VideoCapture(0) # Test mode, load video
cap.set(cv.CAP_PROP_FRAME_WIDTH, RES_X)
cap.set(cv.CAP_PROP_FRAME_HEIGHT, RES_Y)
cap.set(cv.CAP_PROP_FPS, 30) # Set frame rate, testing
if not cap.isOpened():
    print("Cannot open camera")
    exit()


while run_flag.value:
    ret, frame = cap.read() # Capture frame-by-frame
    # if frame is read correctly, ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        cap.release()
        cv.destroyAllWindows()
        run_flag.value = 0
        break
    # Convert to hsv color space
    frame_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    # Find the pool table if it is the first frame
    if initial:
        table = find_table(frame_hsv, color)
        if table is not None:
            hull = ConvexHull(table[:,0,:]) # Convert to ConvexHull object
            table_queue.put(hull) # Send table to queue
            table_queue.put(hull) # Send table to queue
            table_queue.put(hull) # Send table to queue
            print('P0 Put table, queue size: ', table_queue.qsize())
            initial = False
        else:
            print('P0 No table found')

    # Check if time since last send to queue exceeds 30ms
    curr_datetime = datetime.now()
    delta_time = curr_datetime-start_datetime
    delta_time_ms = delta_time.total_seconds()*1000
    if delta_time_ms > 10 and frame_queue.qsize() < 2 and table is not None: # If past time and queue is not full, send to queue
        start_datetime = curr_datetime # Update start time
        new_mask = np.zeros_like(frame)
        img_new = cv.drawContours(new_mask, [table], -1, (255, 255, 255), -1)
        cropped = cv.bitwise_and(frame_hsv, img_new)
        frame_queue.put(cropped) # Send frame to queue
        print('P0 Put frame, queue size: ', frame_queue.qsize())


    if not line_queue.empty(): # Receive lines from queue
        last_receive_time = time.time()
        lines = line_queue.get()
        print('P0 Get line, queue size: ', line_queue.qsize())
    if dev and (table is not None):
        cv.drawContours(frame, [table], -1, (255, 255, 255), 2) # Draw table contour
    if time.time() - last_receive_time < 1:
        for i in lines: # Draw lines to OpenCV frame
```

```
            print('P0 Trajectories, ', lines)
            cv.line(frame, (int(i[0]), int(i[1])), (int(i[2]), int(i[3])), (0, 255, 255), 2, cv.LINE_AA)
    _, buffer = cv.imencode('.jpg', frame)
    jpg_as_text = base64.b64encode(buffer)
    footage_socket.send(jpg_as_text)
    # cv.imshow('frame', frame) # Display the resulting frame


    if cv.waitKey(1) == ord('q'): # Press q to quit
        cap.release() # Release camera
        cv.destroyAllWindows() # Close all windows
        run_flag.value = 0
        print("Set run_flag 0, start quiting sequence")
        break

frame_queue.put(None) # Send None to queue to signal other processes to quit
print("Quiting P0")
print('P0 frame queue empty: ', frame_queue.empty())
print('P0 line queue empty: ', line_queue.empty())


# Process 1 detects the stick, it is exactly the same as Process 2
def process_stick_1(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity):
table = None
while run_flag.value:
    if (table is None) and (not table_queue.empty()):
        table = table_queue.get()
    if not frame_queue.empty():
        frame_hsv = frame_queue.get() # Get frame from queue
        print('P1 Get frame, queue size: ', frame_queue.qsize())
        # color range for pick stick
        lower = np.array([0, 0, sensitivity])
        upper = np.array([255, 255-sensitivity, 255])


        mask = cv.inRange(frame_hsv, lower, upper)
        # Detect lines for stick
        lines = None
        start_length = 320


        cue = np.array([0, 0, 0, 0])
        flag = True
        while not cue.any() or start_length > 15:
            cue = np.array([0, 0, 0, 0])
            start_length = start_length / 2


            if start_length < 40:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 40, None, minLineLength=15, maxLineGap=20)
                flag = False
            else:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 120, None, minLineLength=start_length, maxLineGap=20)


            if lines is not None:
                n = 0
                for i in range(0, len(lines)):
                    l = lines[i][0]
                    cond1 = l[1] < 160 and l[3] < 160
                    cond2 = l[1] > 410 and l[3] > 410
                    # cond3 = l[0] > 0 and l[0] < 160 and l[2] > 0 and l[2] < 170
                    if not (cond1 or cond2):
                        cue += l
                        n += 1
                        cv.line(frame, (l[0], l[1]), (l[2], l[3]), (0,0,0), 2, cv.LINE_AA)
                        break
            if flag == False:
                break

        if cue.any():
            cue = cue / n
            cue = cue.astype(int)
            center = np.array([320, 240])
            d0 = np.linalg.norm(cue[0:2] - center)
            d1 = np.linalg.norm(cue[2:4] - center)
            if d0 < d1:
                cue[0], cue[2] = cue[2], cue[0]
                cue[1], cue[3] = cue[3], cue[1]
                print('P1 Cue coordinates: ', cue) # Print to verify queue contents

                cue = np.array(cue, dtype=np.half)
                stick_euclid = np.linalg.norm(cue[2:4]-cue[0:2])/15
                vec = np.array((cue[2:4]-cue[0:2])/stick_euclid, dtype=np.half)
```

```
                obj_stick = Object(cue[2:4], 3, vec, 5)
                lines = []
                lines = simulate_stick(obj_stick, 100, table, lines, 2)
                print('P1 Trajectories:', lines)
                line_queue.put(lines)
                print('P1 Put line, queue size: ', line_queue.qsize())

    else:
        time.sleep(0.03)
print("Quiting P1")
print('P1 frame queue empty: ', frame_queue.empty())
print('P1 stick queue empty: ', line_queue.empty())


# Process 2 detects the stick
def process_stick_2(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity):
table = None
while run_flag.value:
    if (table is None) and (not table_queue.empty()):
        table = table_queue.get()
    if not frame_queue.empty():
        frame_hsv = frame_queue.get() # Get frame from queue
        print('P2 Get frame, queue size: ', frame_queue.qsize())
        # color range for pick stick
        lower = np.array([0, 0, sensitivity])
        upper = np.array([255, 255-sensitivity, 255])


        mask = cv.inRange(frame_hsv, lower, upper)
        # Detect lines for stick
        lines = None
        start_length = 320


        cue = np.array([0, 0, 0, 0])
        flag = True
        while not cue.any() or start_length > 15:
            cue = np.array([0, 0, 0, 0])
            start_length = start_length / 2


            if start_length < 40:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 40, None, minLineLength=15, maxLineGap=20)
                flag = False
            else:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 120, None, minLineLength=start_length, maxLineGap=20)


            if lines is not None:
                n = 0
                for i in range(0, len(lines)):
                    l = lines[i][0]
                    cond1 = l[1] < 160 and l[3] < 160
                    cond2 = l[1] > 410 and l[3] > 410
                    # cond3 = l[0] > 0 and l[0] < 160 and l[2] > 0 and l[2] < 170
                    if not (cond1 or cond2):
                        cue += l
                        n += 1
                        cv.line(frame, (l[0], l[1]), (l[2], l[3]), (0,0,0), 2, cv.LINE_AA)
                        break
            if flag == False:
                break

        if cue.any():
            cue = cue / n
            cue = cue.astype(int)
            center = np.array([320, 240])
            d0 = np.linalg.norm(cue[0:2] - center)
            d1 = np.linalg.norm(cue[2:4] - center)
            if d0 < d1:
                cue[0], cue[2] = cue[2], cue[0]
                cue[1], cue[3] = cue[3], cue[1]
                print('P2 Cue coordinates: ', cue) # Print to verify queue contents

                cue = np.array(cue, dtype=np.half)
                stick_euclid = np.linalg.norm(cue[2:4]-cue[0:2])/15
                vec = np.array((cue[2:4]-cue[0:2])/stick_euclid, dtype=np.half)
                obj_stick = Object(cue[2:4], 3, vec, 5)
                lines = []
                lines = simulate_stick(obj_stick, 100, table, lines, 2)
                print('P2 Trajectories: ', lines)
                line_queue.put(lines)
                print('P2 Put line, queue size: ', line_queue.qsize())
```

```
    else:
        time.sleep(0.03)
print("Quiting P2")
print('P2 frame queue empty: ', frame_queue.empty())
print('P2 stick queue empty: ', line_queue.empty())


# Process 3 computes Physics
def process_physics(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity):
table = None
while run_flag.value:
    if (table is None) and (not table_queue.empty()):
        table = table_queue.get()
    if not frame_queue.empty():
        frame_hsv = frame_queue.get() # Get frame from queue
        print('P3 Get frame, queue size: ', frame_queue.qsize())
        # color range for pick stick
        lower = np.array([0, 0, sensitivity])
        upper = np.array([255, 255-sensitivity, 255])


        mask = cv.inRange(frame_hsv, lower, upper)
        # Detect lines for stick
        lines = None
        start_length = 320


        cue = np.array([0, 0, 0, 0])
        flag = True
        while not cue.any() or start_length > 15:
            cue = np.array([0, 0, 0, 0])
            start_length = start_length / 2


            if start_length < 40:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 40, None, minLineLength=15, maxLineGap=20)
                flag = False
            else:
                lines = cv.HoughLinesP(mask, 1, np.pi/180, 120, None, minLineLength=start_length, maxLineGap=20)


            if lines is not None:
                n = 0
                for i in range(0, len(lines)):
                    l = lines[i][0]
                    cond1 = l[1] < 160 and l[3] < 160
                    cond2 = l[1] > 410 and l[3] > 410
                    # cond3 = l[0] > 0 and l[0] < 160 and l[2] > 0 and l[2] < 170
                    if not (cond1 or cond2):
                        cue += l
                        n += 1
                        cv.line(frame, (l[0], l[1]), (l[2], l[3]), (0,0,0), 2, cv.LINE_AA)
                        break
            if flag == False:
                break

        if cue.any():
            cue = cue / n
            cue = cue.astype(int)
            center = np.array([320, 240])
            d0 = np.linalg.norm(cue[0:2] - center)
            d1 = np.linalg.norm(cue[2:4] - center)
            if d0 < d1:
                cue[0], cue[2] = cue[2], cue[0]
                cue[1], cue[3] = cue[3], cue[1]
                print('P3 Cue coordinates: ', cue) # Print to verify queue contents

                cue = np.array(cue, dtype=np.half)
                stick_euclid = np.linalg.norm(cue[2:4]-cue[0:2])/15
                vec = np.array((cue[2:4]-cue[0:2])/stick_euclid, dtype=np.half)
                obj_stick = Object(cue[2:4], 3, vec, 5)
                lines = []
                lines = simulate_stick(obj_stick, 100, table, lines, 2)
                print('P3 Trajectories: ', lines)
                line_queue.put(lines)
                print('P3 Put line, queue size: ', line_queue.qsize())

    else:
        time.sleep(0.03)
print("Quiting P3")
print('P3 frame queue empty: ', frame_queue.empty())
print('P3 stick queue empty: ', line_queue.empty())
```

```
RES_X = 640
RES_Y = 480
CENTER_X = RES_X/2
CENTER_Y = RES_Y/2


#Global Run Flag
frame = 0


if __name__ == '__main__':
if args.mode == 'dev':
    dev = True
else:
    dev = False
with open('/home/pi/Project/cali.json') as json_file:
    cali = json.load(json_file)

color = cali['color']
sensitivity = cali['sensitivity']

run_flag = Value('i', 1)
# run_flag controls all processes
# initialize queues for inter-process communication
frame_queue = Queue()
stick_queue = Queue()
line_queue = Queue()
table_queue = Queue()
# initialize processes
p0 = Process(target=grab_frame_display, args=(run_flag, frame_queue, line_queue, table_queue, dev, color))
p1 = Process(target=process_stick_1, args=(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity))
p2 = Process(target=process_stick_2, args=(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity))
p3 = Process(target=process_physics, args=(run_flag, frame_queue, table_queue, line_queue, dev, sensitivity))
# start processes
p0.start()
p1.start()
p2.start()
p3.start()
# wait for processes to finish
p0.join()
p1.join()
p2.join()
p3.join()
```

laptop_main.py

```
#
# W_yz2874_zg284 5/10/2023 Laptop Main Script
#
# Description:
# This script is the main script for the laptop. It will display the UI and
# send the commands to the Raspberry Pi.
#


import base64
import cv2 as cv
import json
import numpy as np
import os
import pygame
from pygame.locals import * # for event MOUSE variables
import zmq


# Initialize pygame and set general parameters
pygame.init() # MUST occur AFTER os enviroment variable calls
pygame.mouse.set_visible(True) # Set mouse visibility
RES_X = 640 # Set screen parametersns +
RES_Y = 480
WHITE = 255, 255, 255 # Set colors
BLACK = 0, 0, 0
screen = pygame.display.set_mode((RES_X, RES_Y)) # Set screen size


# Define the fonts
font_s = pygame.font.Font('From Cartoon Blocks.ttf', 40)
font_m = pygame.font.Font('From Cartoon Blocks.ttf', 50) # Font size 50
font_l = pygame.font.Font('From Cartoon Blocks.ttf', 60)


# Define the buttons
buttons_list = {'start':(RES_X/2, 325), 'calibrate':(RES_X/2, 375), 'quit':(RES_X/2, 425)} # Button dictionary
buttons_cali_t = {'blue':(450, 175), 'green':(450, 275), 'done':(RES_X/2, 425)}
buttons_cali_s = {'increase +':(450, 175), 'decrease -':(450, 275), 'done':(RES_X/2, 425)}


# Graphics
ball_img = pygame.transform.scale(pygame.image.load('balls.png'), (100, 100))
ball_rect = ball_img.get_rect()
ball_rect = ball_rect.move((390, 130))


stick_img = pygame.transform.scale(pygame.image.load('stick.png'), (200, 50))
stick_rect = stick_img.get_rect()
stick_rect = stick_rect.move((150, 190))


code_run = True
ui_run = True
calibrate = False
rpi_run = False
dev = False


while code_run:
screen.fill(BLACK) # Erase the work space
# Define the header and display it on the screen
header_text = font_l.render("Pi Billiard Assistant", True, WHITE)
header_rect = header_text.get_rect(center=(RES_X/2, 90))
screen.blit(header_text, header_rect)
screen.blit(ball_img, ball_rect)
screen.blit(stick_img, stick_rect)
# Initialize the button and display it on the screen
button_rects = {}
for text, text_pos in buttons_list.items():
    text_surface = font_m.render(text, True, WHITE)
    rect = text_surface.get_rect(center=text_pos)
    screen.blit(text_surface, rect)
    button_rects[text] = rect # save rect for 'my-text' button
text_surface = font_s.render('mode: user', True, WHITE)
dev_rect = text_surface.get_rect(center=(RES_X/2, 275))
screen.blit(text_surface, dev_rect)
pygame.display.flip()


while ui_run:
    for event in pygame.event.get(): # for detecting an event for touch screen...
```

```
        if (event.type == MOUSEBUTTONUP):
            pos = pygame.mouse.get_pos()
            if dev_rect.collidepoint(pos):
                dev = not dev
                screen.fill(BLACK) # Erase the work space
                screen.blit(header_text, header_rect)
                screen.blit(ball_img, ball_rect)
                screen.blit(stick_img, stick_rect)
                for text, text_pos in buttons_list.items():
                    text_surface = font_m.render(text, True, WHITE)
                    rect = text_surface.get_rect(center=text_pos)
                    screen.blit(text_surface, rect)
                if dev:
                    text_surface = font_s.render('mode: dev', True, WHITE)
                else:
                    text_surface = font_s.render('mode: user', True, WHITE)
                screen.blit(text_surface, dev_rect)
                pygame.display.flip()
            for (text, rect) in button_rects.items(): # for saved button rects...
                if (rect.collidepoint(pos)): # if collide with mouse click...
                    if (text == 'start'): # indicate correct button press
                        ui_run = False
                        rpi_run = True
                        pygame.quit()
                    elif (text == 'calibrate'): # indicate correct button press
                        ui_run = False
                        calibrate = True
                    elif (text == 'quit'): # indicate correct button press
                        ui_run = False
                        code_run = False


if rpi_run:
    print('Starting Pi Billiard Assistant...')
    if dev:
        os.system('bash ./startpi_dev.sh')
    else:
        os.system('bash ./startpi.sh')
    context = zmq.Context()
    footage_socket = context.socket(zmq.PAIR)
    footage_socket.bind('tcp://*:5555')


    while True:
        # print('Receiving...')
        frame = footage_socket.recv_string()
        img = base64.b64decode(frame)
        npimg = np.fromstring(img, dtype=np.uint8)
        source = cv.imdecode(npimg, 1)
        cv.imshow('Stream', source)
        if cv.waitKey(1) == ord('q'): # Press q to quit
            cv.destroyAllWindows() # Close all windows
            print("Connection closed")
            ui_run = True
            os.system('bash ./killpi.sh')
            break

if calibrate:
    color = [100, 120]
    sens = 175
    cali_stage = 0
    screen.fill(BLACK) # Erase the work space
    header_text = font_l.render("Calibrate Table", True, WHITE)
    header_rect = header_text.get_rect(center=(RES_X/2, 60))
    screen.blit(header_text, header_rect)
    cali_t_rects = {}
    for text, text_pos in buttons_cali_t.items():
        text_surface = font_s.render(text, True, WHITE)
        rect = text_surface.get_rect(center=text_pos)
        screen.blit(text_surface, rect)
        cali_t_rects[text] = rect # save rect for 'my-text' button
    text_surface = font_s.render('table color', True, WHITE)
    rect = text_surface.get_rect(center=(200, 150))
    screen.blit(text_surface, rect)
    text_surface = font_m.render('BLUE', True, WHITE)
    rect = text_surface.get_rect(center=(200, 225))
    screen.blit(text_surface, rect)
    pygame.display.flip()
    while cali_stage == 0:
        for event in pygame.event.get(): # for detecting an event for touch screen...
            if (event.type == MOUSEBUTTONUP):
                pos = pygame.mouse.get_pos()
                for (text, rect) in cali_t_rects.items(): # for saved button rects...
                    if (rect.collidepoint(pos)): # if collide with mouse click...
```

```
                        if (text == 'blue'): # indicate correct button press
                            color = [100, 120]
                            screen.fill(BLACK) # Erase the work space
                            for text, text_pos in buttons_cali_t.items():
                                text_surface = font_s.render(text, True, WHITE)
                                rect = text_surface.get_rect(center=text_pos)
                                screen.blit(text_surface, rect)
                            screen.blit(header_text, header_rect)
                            text_surface = font_m.render('BLUE', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 225))
                            screen.blit(text_surface, rect)
                            text_surface = font_s.render('table color', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 150))
                            screen.blit(text_surface, rect)
                            pygame.display.flip()
                        elif (text == 'green'): # indicate correct button press
                            color = [50, 70]
                            screen.fill(BLACK) # Erase the work space
                            for text, text_pos in buttons_cali_t.items():
                                text_surface = font_s.render(text, True, WHITE)
                                rect = text_surface.get_rect(center=text_pos)
                                screen.blit(text_surface, rect)
                            screen.blit(header_text, header_rect)
                            text_surface = font_m.render('GREEN', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 225))
                            screen.blit(text_surface, rect)
                            text_surface = font_s.render('table color', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 150))
                            screen.blit(text_surface, rect)
                            pygame.display.flip()
                        elif (text == 'done'): # indicate correct button press
                            cali_stage = 1
                            break
    screen.fill(BLACK) # Erase the work space
    header_text = font_l.render("Calibrate Stick", True, WHITE)
    header_rect = header_text.get_rect(center=(RES_X/2, 60))
    screen.blit(header_text, header_rect)
    cali_s_rects = {}
    for text, text_pos in buttons_cali_s.items():
        text_surface = font_s.render(text, True, WHITE)
        rect = text_surface.get_rect(center=text_pos)
        screen.blit(text_surface, rect)
        cali_s_rects[text] = rect # save rect for 'my-text' button
    text_surface = font_m.render(str(sens), True, WHITE)
    rect = text_surface.get_rect(center=(200, 225))
    screen.blit(text_surface, rect)
    text_surface = font_s.render('sensitivity', True, WHITE)
    rect = text_surface.get_rect(center=(200, 150))
    screen.blit(text_surface, rect)
    pygame.display.flip()
    while cali_stage == 1:
        for event in pygame.event.get(): # for detecting an event for touch screen...
            if (event.type == MOUSEBUTTONUP):
                pos = pygame.mouse.get_pos()
                for (text, rect) in cali_s_rects.items(): # for saved button rects...
                    if (rect.collidepoint(pos)): # if collide with mouse click...
                        if (text == 'increase +'): # indicate correct button press
                            sens += 5
                            screen.fill(BLACK) # Erase the work space
                            text_surface = font_m.render(str(sens), True, WHITE)
                            rect = text_surface.get_rect(center=(200, 225))
                            screen.blit(text_surface, rect)
                            for text, text_pos in buttons_cali_s.items():
                                text_surface = font_s.render(text, True, WHITE)
                                rect = text_surface.get_rect(center=text_pos)
                                screen.blit(text_surface, rect)
                            screen.blit(header_text, header_rect)
                            text_surface = font_s.render('sensitivity', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 150))
                            screen.blit(text_surface, rect)
                            pygame.display.flip()
                        elif (text == 'decrease -'): # indicate correct button press
                            sens -= 5
                            screen.fill(BLACK) # Erase the work space
                            text_surface = font_m.render(str(sens), True, WHITE)
                            rect = text_surface.get_rect(center=(200, 225))
                            screen.blit(text_surface, rect)
                            for text, text_pos in buttons_cali_s.items():
                                text_surface = font_s.render(text, True, WHITE)
                                rect = text_surface.get_rect(center=text_pos)
                                screen.blit(text_surface, rect)
                            screen.blit(header_text, header_rect)
                            text_surface = font_s.render('sensitivity', True, WHITE)
                            rect = text_surface.get_rect(center=(200, 150))
```

```
                        screen.blit(text_surface, rect)
                        pygame.display.flip()
                    elif (text == 'done'): # indicate correct button press
                        cali_stage = 0
                        break
print('Calibrate Complete!')
dictionary = {
    'color': color,
    'sensitivity': sens
}


json_object = json.dumps(dictionary, indent = 4)


with open("cali.json", "w") as outfile:
    outfile.write(json_object)

os.system('bash ./send_cali.sh')


ui_run = True
calibrate = False
```