Automated Blackjack Dealer

ME102b Group 3
Yiheng Ji, Matthew Thomas, Hyeong Yoon, Lori Huang

I. Opportunity the Dealer Addresses

Our device partially automates a blackjack game by performing the role of the dealer. The device aims to reduce human errors while unburdening the labor of the dealer during the game.

II. High-level strategy

We wanted our automated blackjack dealer to perform 3 tasks: shoot individual cards to different players, read the values of the cards that it shoots, and sense whether a player decides to “hit” or “stand”. To implement card dealing/shooting, we needed to build a machine capable of ejecting cards and rotating in order to shoot cards in different directions.

To read the cards, we planned to use a camera and an Optical Character Recognition model to read the values of each card. Reading the values of each card would allow us to determine whether a player’s hand had busted or not. To sense a “hit” or a “stand”, we used two buttons - one which represented a “hit”, and the other which represented a “stand”.

Ideally, our device would be able to read and eject cards about once every second. The device should also be able to eject cards at least 3 feet in order to reach the players.

III. Photo of Fully Assembled Device

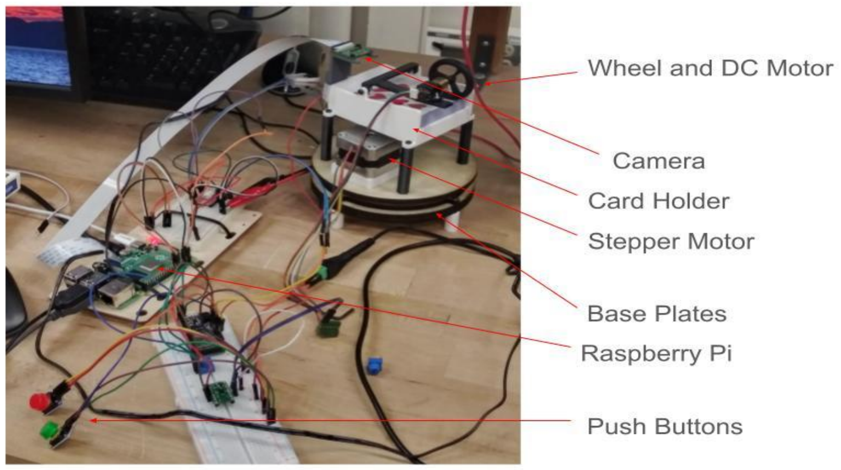


Figure1: Card Dealer Machine with a Camera

IV. Function Critical Decisions

<u>DC Motor Speed</u>

After the cards shoot out, they follow the physics laws as listed below:

$$a = \frac{F}{m}$$

$$F_{drag} = \frac{1}{2} C_d A \rho V$$

Assume the goal is to shoot one card for about 20 cm on the table. The required initial speed of the card can be calculated by plugging the equations above:

$$V^2 - 0^2 = 2 \cdot a \cdot L$$
$$V^2 = 2 \cdot (\frac{\frac{1}{2} \cdot 1.28 \cdot 0.0006m^2 \cdot 1.168kg/m^3 \cdot (\frac{V}{2})}{0.002}) \cdot (0.2)$$
$$V = 2 \cdot 0.112 \cdot 0.2 = 0.044m/s$$

Thus, a DC motor with 400 rpm is sufficient:

$$\omega = \frac{v}{r} = \frac{0.044}{0.015} = 2.9rad/s$$
$$\omega_{rpm} = \omega \frac{60}{2\pi} = 28rpm \ll 400rpm$$

Card Exiting Angle and Force

To offset part of the friction between cards so that only one card is shot out each time, a slop of 10 degrees is attached at the exit of the card holder. Thus, the force and acceleration holding the second card still in the box can be calculated as:

$$F = mg \cdot sin(\theta) = 0.002kg \cdot sin(10) = 3.47 \times 10^4 N$$
$$a = \frac{F}{m} = 0.174m/s^2$$

Stepper motor shaft loads and Gear Calculations

The size of the ring gear was chosen based on the size of our base, we selected this pitch diameter to be 110 mm. Our pinion gear needs to be roughly less than half the radius of our ring gear, so a pitch diameter of 40 mm was selected. Due to the constraints we have on manufacturing (laser plywood and 3d printing in PLA), we opted for a module of 2, with the pinion having 20 teeth and ring gear with 55, with the resulting gear ratio 2.75.

Our base needs to revolve around 1 revolution/sec to meet our dealing specs, using our gear ratio, the necessary stepper rpm is calculated:

$$RPM_{shaft} \cdot T_{pinion} = RPM_{base} \cdot T_{ring}$$
$$RPM_{shaft} = \frac{60 \cdot 55}{20} = 165\ RPM$$

From our motor specs, the corresponding torque output for this speed is 0.24 N m.

$$\tau = F \cdot l = mg\mu \cdot \int_{0.1}^{0.15} r\, dr$$
$$= 0.3kg \cdot 9.8N/m \cdot 0.7 \cdot 0.05$$
$$= 0.10Nm$$

The resultant torque on our stepper motor shaft is within the motor torque limit:

$$\tau_{motor} = GearRatio * \tau = \frac{1}{2.75}\tau$$
$$= 0.037 \ll 0.24NM$$

V. Circuit and State Transition Diagrams

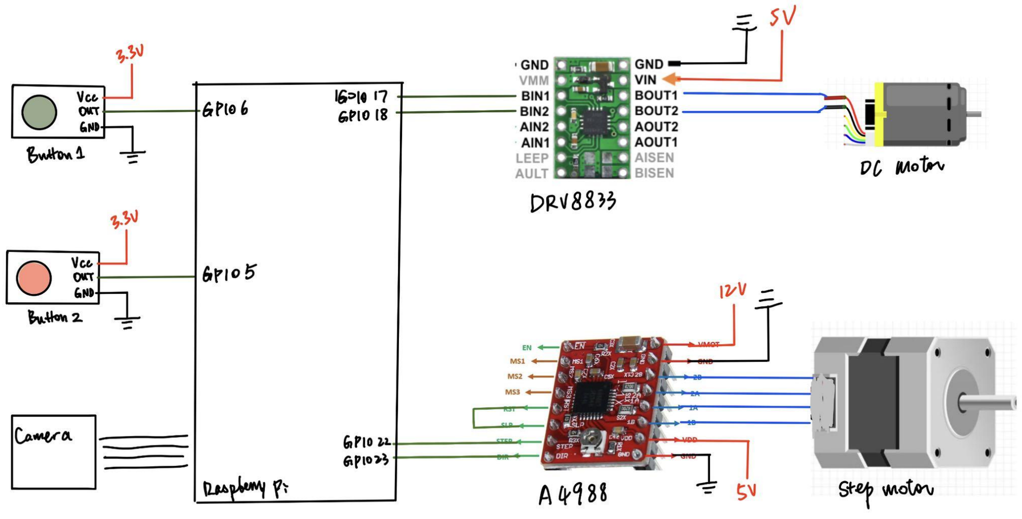


Figure2: Circuit Diagram

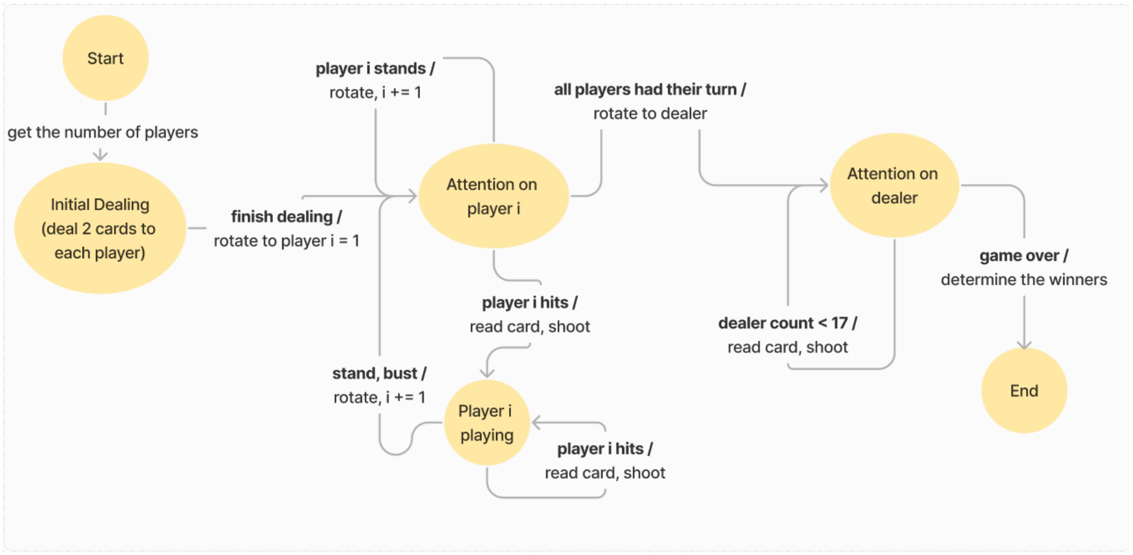

Start
get the number of players
Initial Dealing (deal 2 cards to each player)
finish dealing / rotate to player i = 1
player i stands / rotate, i += 1
Attention on player i
all players had their turn / rotate to dealer
Attention on dealer
game over / determine the winners
End
player i hits / read card, shoot
dealer count < 17 / read card, shoot
stand, bust / rotate, i += 1
Player i playing
player i hits / read card, shoot

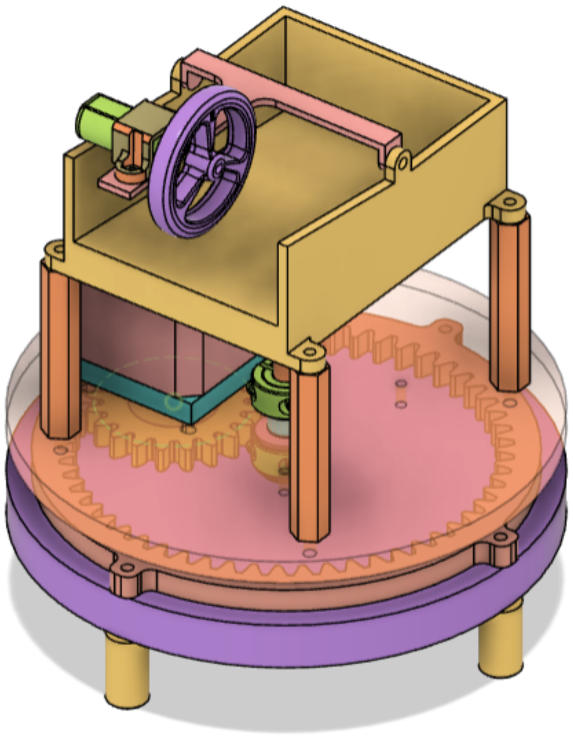
Figure3: State Transition Diagram

VI. Reflections for Future Students of 102B

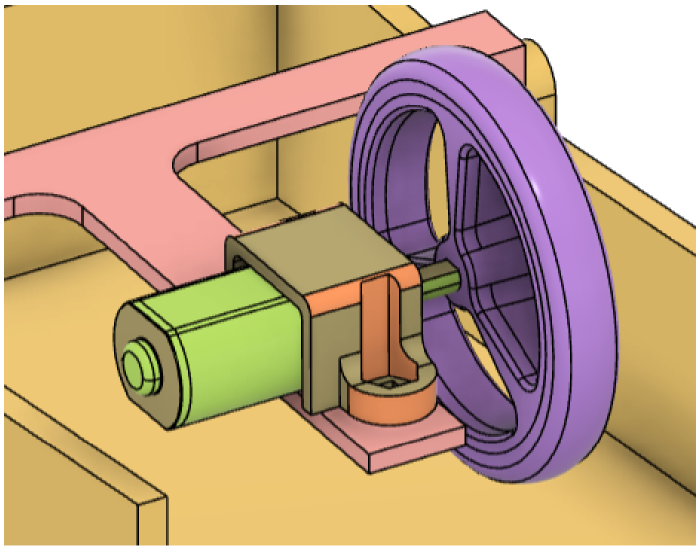
Starting early is the best thing you can do. Often, subsystems that you think will be trivial to make turn out to be a lot more tedious than expected. Instead of developing and building everything at once, make your progress modular so a team can isolate each issue from another. For example, we had our own test logic for shooting, rotating, and reading.

VII. Appendices

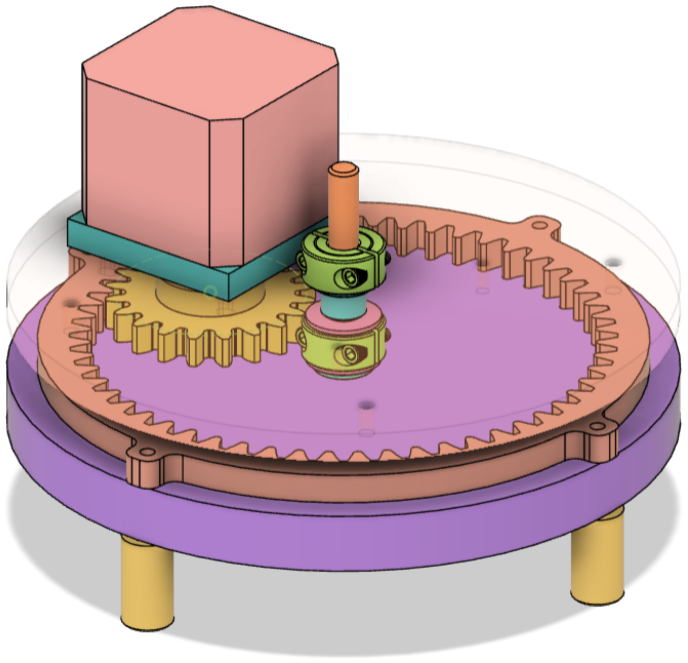
CAD



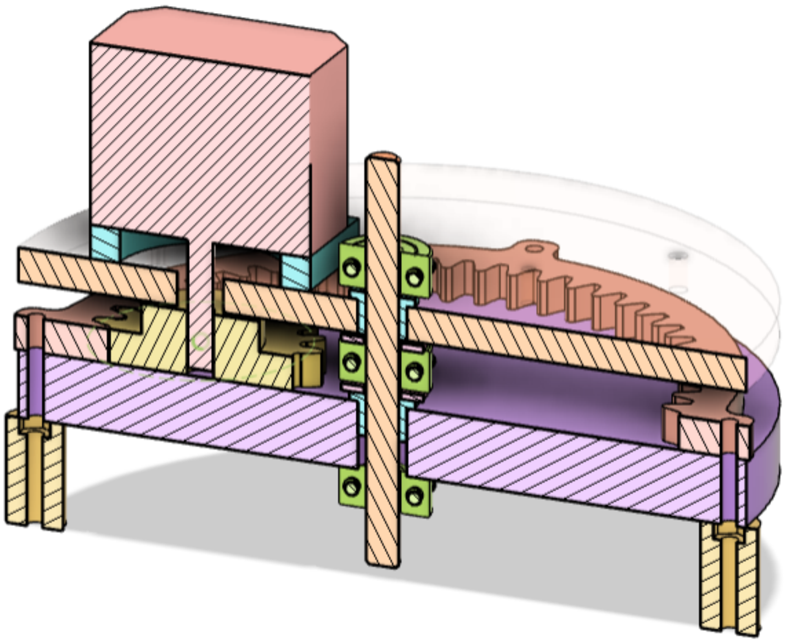
Full Isometric Assembly



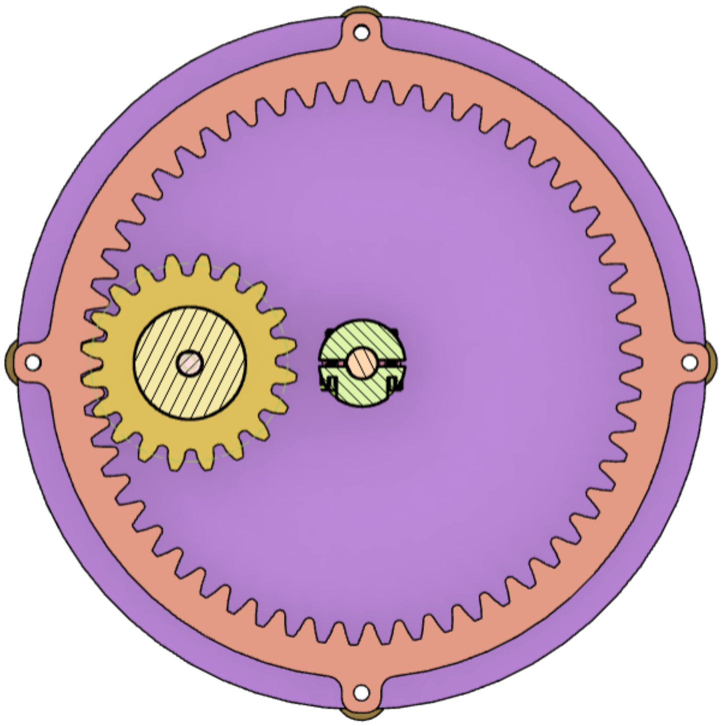
DC Motor and wheel Assembly



Stepper and revolving base subassembly



Shaft transmission crossection view



Ring Gear Top View

BOM

Component	Quantity	Price per	Total	Vendor	Link
Hardware					
1/4" Rotary Shaft	1	4.8	4.8	McMaster	https://www.mcmaster.com/catalog/129/1339/1327K113
M3 SHS screws + washers + nuts (assorted)	1	18.99	18.99	Amazon	https://www.amazon.com/VIGRUE-610PCS-Socket-Machine-Assortment/dp/B09NR8X2LV/ref=sr_1_10?crid=3C5X7LAYNZKNT&keywords=m3%2Bscrews%2Bsocket%2Bhead&qid=1698012121&sprefix=m3%2Bscrews%2Bsocle%2Caps%2C144&sr=8-10&th=1
Gearmotor Bracket Pair	1	2.95	2.95	Pololu	https://www.pololu.com/product/989
Pololu Wheel 40 * 7 mm Pair	1	4.95	4.95	Pololu	https://www.pololu.com/product/1452
1/4" Ligth Duty Sleeve Bearing (flange)	2	0.69	1.38	McMaster	https://www.mcmaster.com/catalog/129/1413/6389K231
1/4" Shaft Collars	3	6.19	18.57	McMaster	https://www.mcmaster.com/catalog/12

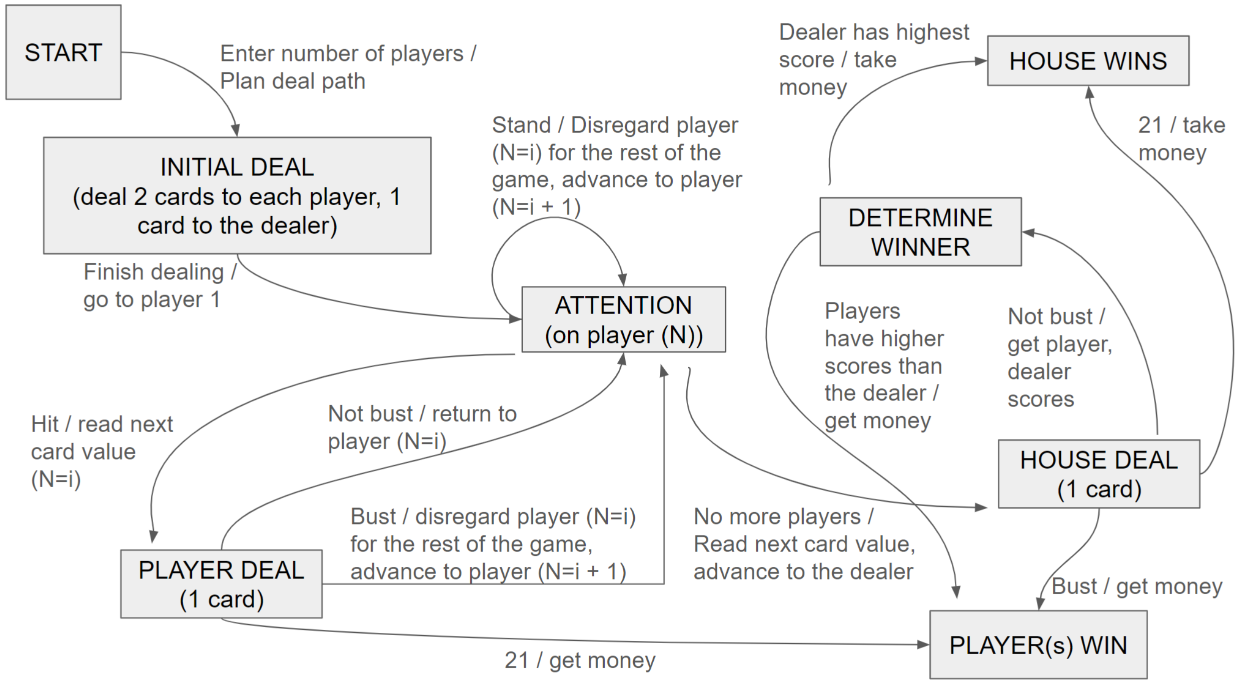
					9/1436/6436K12
¼” Flat Washer	4	0.33	1.32	Ace Hardware	
Material					
PLA	250 g	5	5	Jacobs Hall	
1/8" Plywood	18" x 15"	4.16	4.16	Jacobs Hall	
Electronics					
Micro Metal Gear Motor (75:1)	1	19.95	19.95	Pololu	https://www.pololu.com/product/2366/specs
Nema 17 Stepper Motor	1	14	14	Jacobs Hall	
Raspberry Pi	1	63.61	63.61	Amazon	https://www.amazon.com/Raspberry-Model-2019-Quad-Bluetooth/dp/B07TC2BK1X/ref=sr_1_3?keywords=raspberry+pi&qid=1698033296&sr=8-3&ufe=app_do%3Aamzn1.fos.006c50ae-5d4c-4777-9bc0-4513d670b6bc
Pi Cam	1	9.99	9.99	Amazon	https://www.amazon.com/Arducam-Megapixels-Sensor-OV5647-Raspberry/dp/B012V1HEP4/ref=sr_1_4?keywords=Raspberry+Pi+Camera+Module&qid=1698033326&sr=8-4
A4988 Stepper Motor Driver	1	5.89	5.89	Amazon	https://www.amazon.com/HiLetgo-Stepstick-Stepper-Printer-Compatible/dp/B00LOF1CA2?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&psc=1&smid=A30QSGOJR8LMXA
DRV8835	1	4.95	4.95	Pololu	https://www.pololu.com/product/2135
		Total:	148.4		

Description for figure 2 and 3

In Figure 2, The card dealer machine uses a Raspberry Pi to process the inputs and manage the outputs. The two push button modules with 3 pins are used for analog inputs, which are binary signals. A camera that reads the cards is used to take digital input. On the output side, the 5V DC Motor is driven by driver DRV8833 and the 12V step motor NEMA 17 is driven by A4988.

In Figure 3, the machine will initiate by inquiring about the number of players participating in the game(**state 1**). This input will determine the player's position and initialize parameters, including angles and data structures to keep track of card distribution for each player and the dealer. Once the initial dealing is complete, the machine will rotate to the first player (i=0) and await their input—either a "hit" or "stand" command(**state 2**). In the event of a "stand," the machine will rotate to the next player. Conversely, if the player chooses to "hit," the machine will read and distribute a card to the player, repeating this process until the player either busts or chooses to stand(**state 3**). Once all players have completed their turns, the machine will rotate to the dealer's position. The dealer will then deal cards while the sum of the dealer's cards is under 17(**state 4**). Subsequently, the machine will determine the winners based on the dealt cards, signaling the end of the game(**state 5**).

Logic Flow Chart:


START
Enter number of players / Plan deal path
INITIAL DEAL (deal 2 cards to each player, 1 card to the dealer)
Finish dealing / go to player 1
Stand / Disregard player (N=i) for the rest of the game, advance to player (N=i + 1)
ATTENTION (on player (N))
Hit / read next card value (N=i)
Not bust / return to player (N=i)
Bust / disregard player (N=i) for the rest of the game, advance to player (N=i + 1)
PLAYER DEAL (1 card)
21 / get money
No more players / Read next card value, advance to the dealer
Dealer has highest score / take money
HOUSE WINS
21 / take money
DETERMINE WINNER
Players have higher scores than the dealer / get money
Not bust / get player, dealer scores
HOUSE DEAL (1 card)
Bust / get money
PLAYER(s) WIN

```
import time
import RPi.GPIO as GPIO

# SOME OF THE LOWER GPIO PINS ARE PULLED UP BY DEFAULT (use a higher pin)
hitButton = 6
standButton = 5

def init_buttons():
        GPIO.setup(hitButton, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
        GPIO.setup(standButton, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)



""" Returns true if hit, false if stand """
def poll_button():
        # while GPIO.input(hitButton) == 0 and GPIO.input(standButton):
        #    time.sleep(1)
        hit = 0
        stand = 0

        while True:
                hit = GPIO.input(hitButton)
                stand = GPIO.input(standButton)
                if hit:
                        print("Hit!")
                        time.sleep(1)
                        hit = 0
                        stand = 0
                        return True

                if stand:
                        print("Stand!")
                        hit = 0
                        stand = 0

                        time.sleep(1)
                        return False
```

```
import cv2
import numpy as np
import pytesseract
import time

NUM_SHOTS = 10 #number of images that we get

""" Returns a video capture object """


fail = False

def init_detection():
    # Initialize the webcam
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error opening video")
        exit(-1)

    ret, frame = cap.read()
    height, width, _ = frame.shape
    cv2.namedWindow('Raw Image', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('Raw Image', width, height)

    return cap


""" Returns the card number. Returns [] if unsuccesful """
def detect_card(cap):

    card_images = []
    count = 0
    fail_count = 0

    # for _ in range(NUM_SHOTS):
    while count < 7 and fail_count < 10:
        ret, frame = cap.read()

        if not ret:
            break

        # Format the raw image
        flipped_frame = cv2.flip(frame, -1)

        x,y,h,w = 0,50,240,320
        # frame = flipped_frame[180:350, 240:340] #height and width
        frame = flipped_frame[180:340, 250:360] #height and width

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = cv2.GaussianBlur(frame, (5,5), 0)

        # _, frame = cv2.threshold(frame, 30, 200, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
        _, frame = cv2.threshold(frame, 30, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

        # Show the formatted image
        cv2.imshow('Raw Image', frame)

        # Find the card number
        pytesseract.pytesseract.tesseract_cmd = '/usr/bin/tesseract'  # Replace with
the actual path from 'which tesseract'
        custom_config = r'--oem 3 --psm 7 -c tessedit_char_whitelist=2345678910AJQK'
        text = pytesseract.image_to_string(frame, config=custom_config)

        # print("type is", type(text))
        # print("length of the string is ", len(text))

        #  card_images.append(text)
        if text.strip():
            single_char = text.strip()[0]
            card_images.append(single_char)
            count = count + 1
        else:
            print("Failed to detect")
            print(fail_count)
            # time.sleep(0.5)
            fail_count = fail_count + 1
            # single_char = '*'
            # card_images.append(single_char)

        # DEBUGGING
        # print(f"card detected: {text}")

        # Check for the 'q' key to quit the program
        if cv2.waitKey(1) & 0xFF == ord('q'):
            return '*'

    if fail_count >= 10:
        card = input("Enter card: ")
        card = card.upper()
        card = card[0]
        return card

    # Return the
    ret_val = max(card_images,key=card_images.count)
    if len(ret_val) > 1:
        print('There was a tie in reading card')
        print(ret_val)
        ret_val = ret_val[0]

    # print('sanity check inside detection logic')
    print("camera read ", ret_val)
    # if ret_val.strip():
    #     # try:
    #     #     single_char = text.strip()[0]
    #     #     print("Detected:", single_char)
    #     # except Exception:
    #     #     print("something wrong")
    #     return ""

    # else:
    #     print("No text detected")

    # print("inside the detectio function")
    # print(type(ret_val))
    # # DEBUGGING
    # print(f"ret_val: {ret_val[0], ret_val[1], ret_val[2]}")
    # print("reading",  ret_val)

    return ret_val


def close_detection(cap):
    # Release the capture object and close the OpenCV window
    cap.release()
    cv2.destroyAllWindows()


#ret, frame = cap.read()
    # if ret:
    #     height, width, _ = frame.shape
    #     cv2.namedWindow('Raw Image', cv2.WINDOW_NORMAL)
    #     cv2.resizeWindow('Raw Image', width, height)
    #     print("width ", width)
    #     print("height ", height)
```

```python
import RPi.GPIO as GPIO
import time

DIR = 22
STEP = 23
steps_per_rev = 200

GPIO.setmode(GPIO.BCM)

def init_rotate():
    GPIO.setup(DIR, GPIO.OUT)
    GPIO.setup(STEP, GPIO.OUT)

def spin_clockwise(steps):
    print("Spinning Clockwise...")
    GPIO.output(DIR, GPIO.HIGH)

    for _ in range(steps):
        GPIO.output(STEP, GPIO.HIGH)
        time.sleep(0.001)
        #time.sleep(0.003)
        GPIO.output(STEP, GPIO.LOW)
        time.sleep(0.001)
        #time.sleep(0.003)

    time.sleep(1)

def spin_anticlockwise(steps):
    print("Spinning Anti-Clockwise...")
    GPIO.output(DIR, GPIO.LOW)

    for _ in range(steps):
        GPIO.output(STEP, GPIO.HIGH)
        time.sleep(0.001)
        #time.sleep(0.003)
        GPIO.output(STEP, GPIO.LOW)
        time.sleep(0.001)
        #time.sleep(0.003)

    time.sleep(1)
```

```
import RPi.GPIO as GPIO
import time


def init_shoot():
    global CHANNEL_1
    global CHANNEL_2
    global pwm1
    global pwm2

    # GPIO.setmode(GPIO.BCM)

    CHANNEL_1 = 17
    CHANNEL_2 = 18

    #Set up GPIO
    GPIO.setup(CHANNEL_1, GPIO.OUT)
    GPIO.setup(CHANNEL_2, GPIO.OUT)
    pwm1 = GPIO.PWM(CHANNEL_1, 5000)
    pwm2 = GPIO.PWM(CHANNEL_2, 5000)

def drive_forward():
    GPIO.output(CHANNEL_1, GPIO.HIGH)
    GPIO.output(CHANNEL_2, GPIO.LOW)
   # pwm1.start(80)

def drive_backward():
    GPIO.output(CHANNEL_1, GPIO.LOW)
    GPIO.output(CHANNEL_2, GPIO.HIGH)
    #pwm2.start(80)

def stop_motor():
    GPIO.output(CHANNEL_1, GPIO.LOW)
    GPIO.output(CHANNEL_2, GPIO.LOW)

def shoot():
    drive_forward()
    time.sleep(0.220)
    stop_motor()
    pwm1.stop()

    drive_backward()
    time.sleep(0.5)
    stop_motor()
    pwm2.stop()
```

```
import RPi.GPIO as GPIO
import shoot_logic as sl
import detection_logic as dl
import rotate_logic as rl
import button_logic as bl
import time
import sys

STEP_SIZE = 100

card_vals ={}
card_vals['2'] = 2
card_vals['3'] = 3
card_vals['4'] = 4
card_vals['5'] = 5
card_vals['6'] = 6
card_vals['7'] = 7
card_vals['8'] = 8
card_vals['9'] = 9
card_vals['0'] = 10
card_vals['10'] = 10
card_vals['1'] = 10
card_vals['J'] = 10
card_vals['Q'] = 10
card_vals['K'] = 10
card_vals['A'] = 1

def add_cards(idx):
        sum = 0

        for i in range(len(CARDS[idx])):
                key = CARDS[idx][i]
                val = card_vals[key]
                sum = sum + val

        return sum

SETUP = 1
CHOOSE_NUM_PLAYERS = 2
DEAL = 3
GAME = 4
EXIT = 5

state = SETUP

match state:
        case 1:
                """ SETUP """
                GPIO.setmode(GPIO.BCM)
                # Initialize card detection
                # CAP = dl.init_detection()
                # dl.detect_card(CAP)

                # Initialize card shooting
                # Pins 17 and 18 used for shooting cards
                sl.init_shoot()

                # Pins 5 and 6 are used for buttons
                bl.init_buttons()

                # Pin 22 and 23 used for rotating the base
                rl.init_rotate()

                state = CHOOSE_NUM_PLAYERS

        case 2:
                """ CHOOSE NUM PLAYERS """
                num = input("Number of players: ")
                NUM_PLAYERS = int(num)
                while NUM_PLAYERS < 1 and NUM_PLAYERS > 5:
                        if NUM_PLAYERS < 1:
                                print("Too few players")
                        elif NUM_PLAYERS > 5:
                                print("Too many players")

                        num = input("Number of players: ")
                        NUM_PLAYERS = int(num)

                # dl.close_detection()
                SCORES = [] # Keeps track of the scores of each player; player 0 is
the dealer

                state = DEAL

        case 3:
                """ DEAL CARDS """
                CARDS = [] # index i is the cards that player i has

                # 'player 0' is the dealer

                        # it will return empty list []
                        # it will return an element
                CAP = dl.init_detection()
                card = dl.detect_card(CAP)

                dl.close_detection(CAP)
                print(card)
                cards = [card]
                CARDS.append(cards)
                SCORES.append(add_cards(0))
                sl.shoot()

                # Deal to the players
                for i in range(NUM_PLAYERS):


                        player_cards = []

                        # rotate to player i
                        rl.spin_clockwise(STEP_SIZE)

                        # Deal two cards
                        for _ in range(2):
                                CAP = dl.init_detection()
                                # throw away - gives camera enough time to recognize
the next card
                                # for _ in range(1): #change as necessary
                                #       dl.detect_card(CAP)

                                # detecting the card
                                card = dl.detect_card(CAP)

                                player_cards.append(card)

                                sl.shoot()
                                dl.close_detection(CAP)
                                time.sleep(1)

                        # all the cards ex) [2,3] will be added to CARDS
                        CARDS.append(player_cards)
                        SCORES.append(add_cards(i))


                # Rotate back
                for i in range(NUM_PLAYERS):
                        # rotate to player i
                        rl.spin_anticlockwise(STEP_SIZE)

                # DEBUGGING
                print(f"dealer card: {CARDS[0][0]}")
                for i in range(1, NUM_PLAYERS + 1):
                        print(f"player {i} cards: {CARDS[i][0], CARDS[i][1]}")

                print(f"Scores: {SCORES}")

                state = GAME
```

```
        case 4:
                """ GAME """
                # for players
                for i in range(1, NUM_PLAYERS + 1):
                        # rotate to player i
                        rl.spin_clockwise(STEP_SIZE)
                        done = False
                        player_cards = CARDS[i]

                        while not done:
                                print("hit?")
                                has_hit = bl.poll_button()

                                if has_hit:
                                        # throw away - gives camera enough time to
recognize the next card
                                        # for _ in range(1): #change as necessary
                                        #       dl.detect_card(CAP)
                                        CAP = dl.init_detection()
                                        next_card = dl.detect_card(CAP)
                                        # # detecting the card
                                        # next_card = '*'
                                        # while next_card == '*':
                                        #       next_card = dl.detect_card(CAP)
                                        #       print(next_card)

                                        player_cards.append(next_card)
                                        print("player's cards: ", player_cards)

                                        sl.shoot()
                                        dl.close_detection(CAP)
                                        if add_cards(i) > 21:
                                                done = True
                                                print("Bust!")
                                else:
                                        done = True

                                SCORES[i] = add_cards(i)


                # Rotate back
                for i in range(NUM_PLAYERS):
                        # rotate to player i
                        rl.spin_anticlockwise(STEP_SIZE)


                done = False
                player_cards = CARDS[0]

                while not done:
                        # throw away - gives camera enough time to recognize the 
next card
                        # for _ in range(1): #change as necessary
                        #       dl.detect_card(CAP)
                        CAP = dl.init_detection()
                        # detecting the card
                        # next_card = '*'
                        # while next_card == '*':
                        #       next_card = dl.detect_card(CAP)
                        #       print(next_card)

                        next_card = dl.detect_card(CAP)

                        player_cards.append(next_card)
                        print("dealer's cards: ", player_cards)

                        sl.shoot()
                        dl.close_detection(CAP)

                        if add_cards(0) >= 17:
                                done = True
                                print("Dealer's card is over 17. Stop the game!")

                        SCORES[0] = add_cards(0)

                state = EXIT

        case 5:
                """ EXIT """
                winners = []
                ties = []
                losers = []
                dealer_score = SCORES[0]
                result = {}

                for i in range(1, len(SCORES)):
                        score = SCORES[i]
                        if score > 21:
                                losers.append(i)
                                result[i] = 'Lost'
                        elif score <= 21 and dealer_score <= 21 and score < 
dealer_score:
                                losers.append(i)
                                result[i] = 'Lost'
                        elif dealer_score > 21 and score <= 21:
                                winners.append(i)
                                result[i] = 'Won'
                        elif dealer_score <= 21 and score > dealer_score and score 
<= 21:
                                winners.append(i)
                                result[i] = 'Won'
                        elif score == dealer_score :
                                ties.append(i)
                                result[i] = 'Draw'

                print("Dealer cards")
                print(CARDS[0])
                print("Player cards")
                print(CARDS[1:len(SCORES)])
                print(SCORES)
                for i in range(1, len(SCORES)):
                        print(f"Player {i} {result[i]}")

                #dl.close_detection()
                sys.exit()
```