Team E8: Smart Poker Table

Authors: **Brandon Hung**: Carnegie Mellon University;

Zongpeng Yu: Electrical and Computer Engineering, Carnegie Mellon University;

Patrick Kollman: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—The current standard for live poker uses the outdated method of human estimation to track valuable information about the game state, such as the overall pot size and bet sizes. This practice often leads to slower playing times and shifts a player's focus from the crucial decision making processes involved in playing towards bookkeeping, which ultimately takes away from the playing experience. We propose a system to track and display important game elements for real-life poker players. Motivated by online poker, our system will track individual bet sizes, the overall pot size, and player order through a device controlled by a Raspberry Pi. The device will include a computer vision system to scan stacks, an actuation system to direct the camera to the correct location, and a real-time display to show this information to the players. The game state will be controlled by a dealer through an intuitive UI. Our goal is to provide a simple device that a dealer can use to provide an online poker-like experience to in-person games, improving overall gameplay quality.**

***Index Terms*—Computer Vision, Mechatronics, Microcontrollers, Motor Control, Python, Servo, Systems, User Interface**

I. Introduction

The rules of poker state that all table information should be available to players at all times. Table information is crucial to competitive poker players for making informed decisions during a game. For example, knowing the stack size of other players is very important when choosing to either call, raise, or fold. If an opponent has more money than themself, they may want to rethink about raising the bet size. If the opponent has less money than themself, it may be an opportune time to step on the pedal and raise. In a real life poker match, if a player wanted to know the size of another player's stack, they would need to ask the casino dealer to physically count that player's stack chip by chip. This method is becoming outdated given the efficiency of online poker games.

Online poker games have revolutionized the poker environment by constantly displaying all table information directly to the players. In an online poker game, players can always view the pot size, player stack sizes, bet sizes, and whose turn it is to act during a round. This is the way poker is meant to be played, with all table information readily available. The goal of the Smart Poker Table is to display the same information provided in an online poker game to real life poker players. With a maximal update time of 10 seconds, the Smart Poker Table will successfully imitate the online poker environment for real life casino poker players.

II. Design Requirements

Our goal with this project is to create an environment that creates a smooth play experience like one found in online poker games; as a result, the most important requirement is providing the players and dealer with real-time, accurate information. Since there is no objective measurement of real-time speed as defined in a poker game, we have chosen a maximal update time of 10 seconds (allotting for turning, image capturing and computer vision processing, and updating the display). In person, a round of poker with a full table takes anywhere from 30 seconds to 10 minutes. Averaging round lengths gives us 5 minutes and 15 seconds; with a full table of eight, our device will require 8 * 4 * 10 = 320 seconds per round, or about 5 minutes and 20 seconds a round in the worst case scenario. An update speed bounded by 10 seconds will allow our device to keep pace with standard live poker games in the worst case.

Our metric for the accuracy of estimating the pot size will be within +/- 10% of the actual pot size. This number may seem quite low, but it is sufficient in the context of the problem. At the end of the game, the winner's payout is based on the physical chips on the table --- which is unrelated to our estimated pot value. Instead, the main purpose of the pot estimation is to help players make better betting choices; a rough idea of the pot size works well for this purpose, which is why 90-110% estimates are just as useful 100% accurate ones.

Our device is intended to be easy to use and require only 5 minutes to learn for the average poker dealer. This represents a way to metric the ease of use since end users ultimately want something which works as out of the box as possible. While this is admittedly an arbitrary value, it provides a good guideline to follow on the eventual simplification/streamlining of the design. As for the computer vision, we would like the image capture and processing pipeline to occur within 3 seconds. This will provide us ample time to meet our maximal update time listed in the first paragraph. Turning the camera to the correct position using a servo will need to occur within 5 seconds to fit alongside our computer vision processing time and maximal update time requirements.

Here, we'll address the more physical requirements of the system. The camera will be positioned at 10 inches off the surface of the table, which allows for average-sized stacks to be placed underneath it. A height of 10 inches and a shortest distance to stacks of 20 inches means the camera will need an FOV of at least 53.12 degrees based on our current design (arctan(10/20)*2 = 53.12). The servo will need to turn within a 4.735 degrees of our reference stack reading position. This translates to a worst case scenario (when the stacks are 48 inches away) of +/- 4 inches of deviation from the horizontal line of our stack reading position (arctan(4/48) = 4.735

degrees). Anything beyond this point will result in the camera pointing in the wrong direction and being unable to read the stack, which will result in a completely incorrect bet size. Finally, our camera will need a minimum resolution of 63x460 pixels. This is derived from our window size of 48x48 square inches and a chip area of 1.53x0.209 square inches, with the formula $axisResolution = 2 \, x \, FOV \, / \, minSize$. Plugging in the values 2*(48/1.53) and 2 * (48 /0.209) gives us a minimum bound of 62.74 and 459.33 pixels in the x and y directions, respectively.

Finally, we want our internal game state to perfectly match the current state of the game in real life. These two must be exactly the same or else synchronization issues will occur in the software.

III. Architecture and/or Principle of Operation

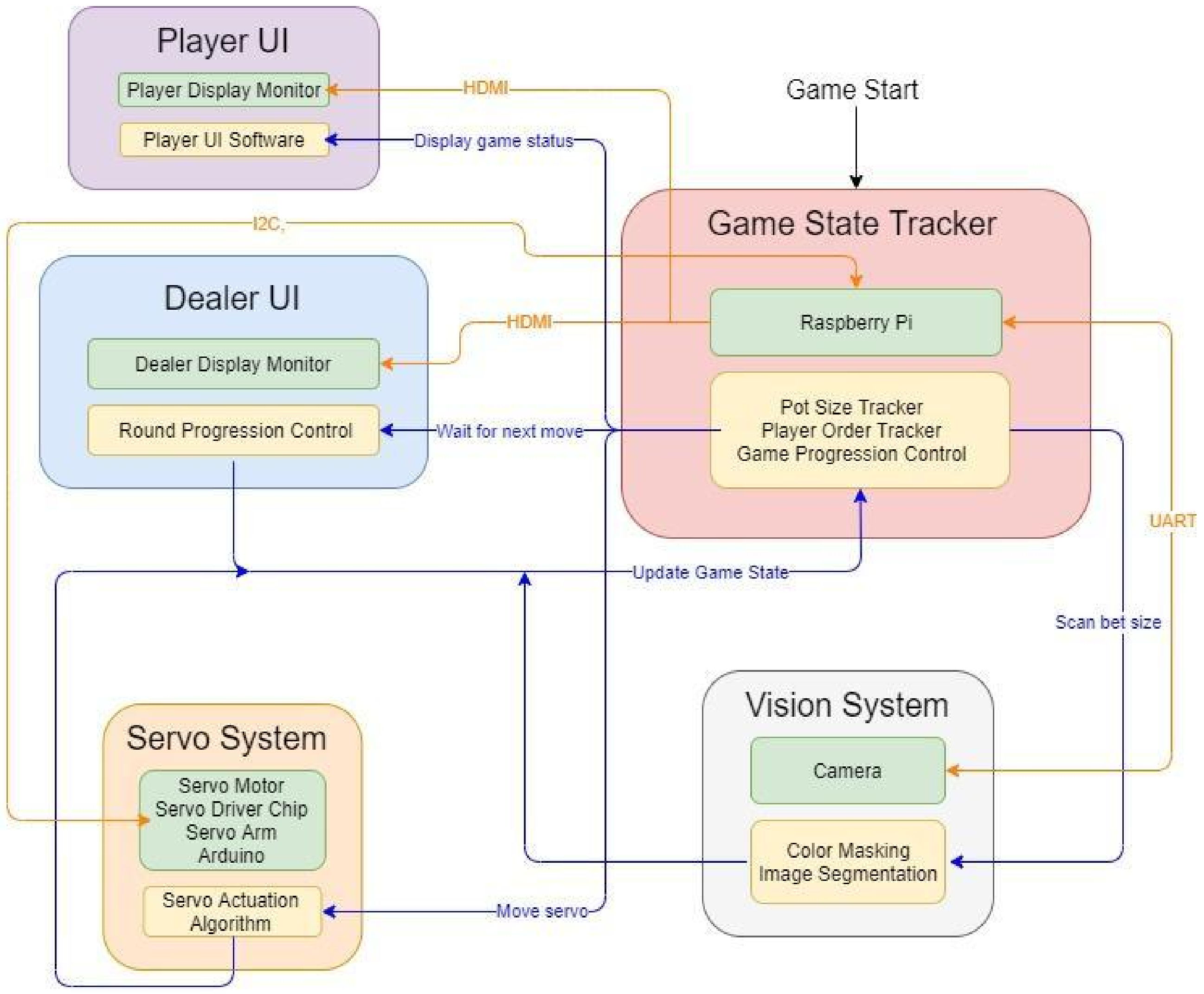


Fig. 1 Overall System Architecture

Broadly speaking, we can divide our project into five main subsystems: the dealer UI, the player UI, the game state tracker, the servo system, and the vision system. Since our design presentation, we have decided that the vision and servo systems were disconnected enough to warrant their own subsystems and separated them from our original hardware subsystem. In the diagram above, each system has hardware component(s) in its green box and software component(s) within its yellow box. The orange arrows show the connections between different hardware components, and the blue arrows represent the conditions and order in which software functions are called. In addition to the block diagram above, we have provided two more block diagrams at the end of the page. The Software Block Diagram details specifically the interplay between the dealer UI and the game state tracker, while the Communications Block Diagram describes in detail the protocols used to communicate between different devices/drivers.

When a game is started, the game state tracker is initialized. The game state tracker (or GST, for short) then initializes the Player UI, the Dealer UI, the Vision System, and the Servo System. Control is then handed to the Dealer UI, which waits for the dealer's input to start a round or to configure the system. Once the dealer starts a round, the "Play Round" loop in the Software Block Diagram begins. During this loop, the Dealer UI collects information regarding players folding and/or betting and transmits it to the GST, which then hands over control to the appropriate subsystem. If a player folds, the GST removes them from the order tracking and proceeds to the Dealer UI. If a player raises or calls, the GST will first proceed to the Servo System. Once the Servo System has moved to the appropriate position, control is returned to the GST which in turn calls the Vision System. After the Vision System collects the stack size information, it transmits the information back to the GST to update the state and repeat the cycle. For a pseudocode explanation of this process, please refer to figure 10 under the Subsystem: Game State Tracker section.

The hardware architecture consists of a Raspberry Pi, a camera, a dealer display monitor, a servo motor / servo driver, and a player display monitor. The Raspberry Pi serves as the main controller of the system, to which the peripherals are connected. To send and receive image data, the Pi communicates with the camera over UART. To actuate the servo with positional feedback, the Pi uses I2C to communicate with the servo driver which in turn uses PWM to drive the servo itself. Both monitors contain different windows which are updated over HDMI as the game progresses. Most of the design is abstracted behind libraries, allowing us to focus more on the algorithms than connecting the subsystems.

IV. Design Trade Studies

A. *Stack Scanning: Computer Vision vs RFID*

One of the biggest design decisions we made as part of our design process was choosing to go with computer vision instead of RFID for the stack height reading. The most prominent reason for this change is feasibility. While we have little actual data on the feasibility of scanning a stack of chips using RFID, feedback from the professors generally suggested it was a nigh impossible task to achieve. While there are a few examples of RFID being used to scan items quickly (RFID Blog, 2020), our problem is fundamentally harder due occlusion from RFID tags. This is less of a problem when scanning items such as the clothes mentioned in the above article, but can prove to be a strong issue when dealing with stacked playing chips. Even the most sophisticated leaders in RFID technology can struggle with scanning thin stacks of more than 25 chips (RFID Journal, 2013). In contrast, computer vision can scan chips using fairly simple methods if properly structured. In the end, we decided to go with a tradeoff of a method that is more likely to provide a guarantee of being able to scan stack heights to some accuracy over experimenting with a relatively unknown technology potentially unable to fulfill our requirements.

B. *Servo Subsystem: Continuous vs Regular*

Another design decision we made was choosing between a standard 180 degree servo and a continuous rotation servo. A 180 degree servo tends to be fairly positionally accurate and easy to use; the ability to simply write an angle to the motor is built into the driver chip we initially wanted to use. Meanwhile, a continuous rotation servo provides 360 degrees of rotation but is more complex in terms of control. Additionally, its feedback is not readable by a Raspberry Pi so using this component requires introducing an Arduino to communicate between the motor and the Pi. Our device placement on the poker table (see Fig. 11, Overall CAD Model) also depends on what motor we choose, as the range of rotation limits how far we can place the device from the dealer. In the end, we chose to go with continuous servo. Despite introducing more complexity, the continuous servo allows us to place the device in the middle of the table and close to the players --- thereby reducing the difficulty of scanning stacks. We believe this will translate into an increased accuracy, making the continuous servo well worth the extra complexity involved with implementation.

C. *Hardware Components Consolidated vs Distributed*

Deciding between a placement of components in which parts were distributed versus a placement that consolidated the parts in one area actually proved to be quite difficult. Distributed components allow us to hide the less visually appealing electrical components. This would create a prettier setup which is less distracting from a standard poker environment. However, this method creates complex wiring schemes that increase both the chance of failure (i.e a wire getting cut) and the difficulty of setup and troubleshooting (since devices must be individually repositioned and inspected). On the other hand, consolidating the various components as much as possible allows us to shorten wires and more quickly identify any hardware issues that crop up during gameplay --- but doing so involves more mechanical design and construction. Our choice to go with consolidated components is based on the decision to invest more in design, as we believe that a good design preemptively mitigates many risks associated with possible system failure later on.

V. System Description

A. *Subsystem: Computer Vision*

The computer vision subsystem includes a USB camera, an algorithm which takes a picture and scans the stack size in the captured photo, and an algorithm to calibrate the colors according to the current lighting conditions. The pseudocode for the stack size scanning algorithm is given in the code block below:

```
photo = capture_photo(camera)
bet_size = 0
for color in chip_colors:
    value = get_value(color)
    masked = mask_photo(photo, color)
    masked = open_and_close(masked)
    stk_size = get_bounding_box(masked)
    n_chips = stk_size.height/chip_height
    bet_size = bet_size + value * num_chips
```

Fig. 2 Stack Scanning Algorithm Pseudocode

First, our camera will need to capture the photo. It will then loop through the colors of each chip and segment the image by color to obtain just the chips which are the color associated with a value. For each of these color-masked images, the computer vision will perform morphological operations to make the image more robust and then find the size of the maximal bounding box. It then references the stored value of chip height and calculates from there the expected number of chips. The height of the stack is then multiplied by the value of each chip to get a value for that stack. The total size of the bet is calculated by summing the value of each stack.

The algorithm used to capture photos leverages existing software in OpenCV:

```
import cv2
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
```

Fig 3. Photo Capture Algorithm

In the snippet above, *frame* is a variable containing an image of what the webcam was looking at when the picture was taken.

The mechanical assembly for the camera and camera mount will look like the following:

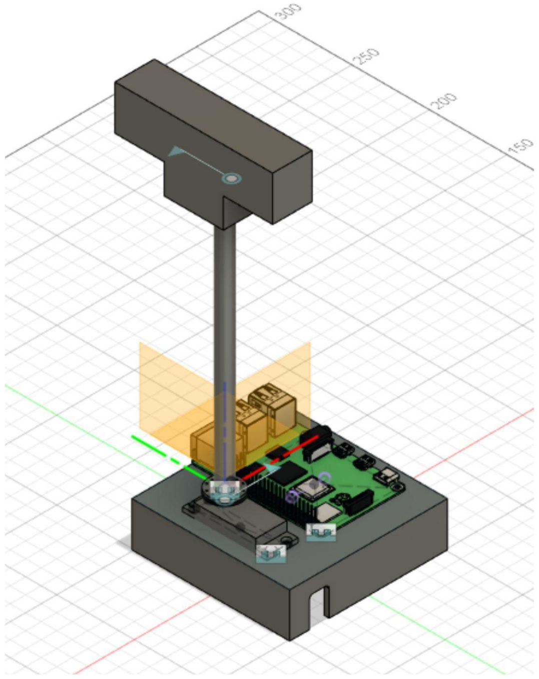


Fig. 4 Camera Mount Assembly

where the camera is mounted to a 3D printed rod with a ¼”-20 threaded end section. The other end of the rod is mounted to the servo horn, which allows the camera to pivot with the servo. The servo, Arduino, and Raspberry Pi will in turn be arranged on the box-like mounting assembly to keep the components close to one another.

B. *Subsystem: Servo*

The servo subsystem includes a 360 continuous rotation servo with feedback from adafruit, 6 volt power supply, arduino uno, and raspberry pi.

To start, we connect 360 servo to arduino. The reason why we didn’t connect servo directly to Raspberry Pi is because the Pi lacks the capability to read PWM from a GPIO. However, the Arduino could do that because it has PWM pins designed to receive feedback, so not only are we able to get the 360 continuous servo rotating with different RPMs, we could also set the angle where we want the servo to rotate to. The library we are using to actually control the servo angle is the Parallax-FeedBack-360-Servo-Control-Library-4-Arduino.

In order to make servo rotate to the correct position, the Arduino will need to receive a signal from the Raspberry Pi containing the angle to turn to. Once received, the Arduino will then begin spinning the servo. After it has reached the target position, the Arduino will transmit a confirmation signal to the Raspberry Pi, allowing the GST to call the Vision Subsystem. For the communication protocol, we will use USB, and we will import the python serial library on the Raspberry Pi and use the builtin serial function for Arduino.

```
// Servo control example (with +/-4 degrees
error) each 1 second.
void move_servo_example(){
    servo.rotate(270, 4);
    delay(1000);
    servo.rotate(-180, 4);
    delay(1000);
}

// Pi to Arduino Serial (python)
import serial
ser = serial.Serial('/dev/ttyUSB0', 9600)
ser.write(b'3')

//Arduino receives from Pi
void loop(){
  if(Serial.available()){
    r = r * (Serial.read() - '0');
    Serial.println(r);
  }
}

//Arduino to Pi Serial
void loop(){
  Serial.println("Sending stuff to Pi");
  delay(2000);
}
// Pi Receives from Arduino (python)
import serial
```

```
ser = serial.Serial('/dev/ttyUSB0', 9600)
while 1:
    if(ser.in_waiting >0):
        data_from_arduino = ser.readline()
```

Fig 5. Code Snippet for Servo Subsystem

The complete servo subsystem diagram is in Fig. 15 under Section *Servo Subsystem Diagram.*

C. *Subsystem: Dealer UI*

The Dealer UI has two functions: system configuration and game state updates. The Dealer UI is meant to be used by non-engineers so it is designed to be very simple and user friendly. It consists of simple controls to calibrate the system and control the game state.

System configuration can occur in between any round, but usually happens at the very beginning of a game. With system configuration, the dealer can control chip denominations, chip colors, the number of players, the location of these players at the table, the stack sizes of these players, and controls when the round begins. These updates are then relayed to the GST.

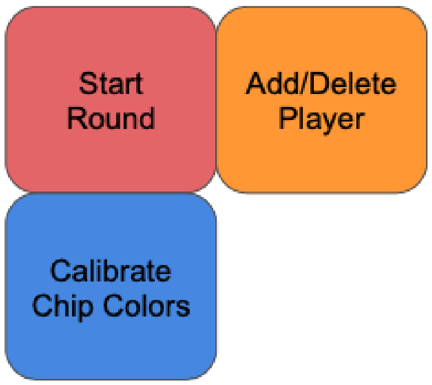


Fig. 7 System Configuration Interface

Game state updates occur after system configuration and when a round has begun. The dealer has at most 4 options when the UI has entered the game state update mode: Check, Fold, Raise, and Call. These are the 4 actions a player can take on any turn, though it may not always be possible to check. The action chosen is then relayed to the GST. The GST will then properly update the game state, scan the player's stack if needed, and rotate the servo. In this manner, the GST is polling the Dealer UI during the round for player actions. When an action is inputted into the Dealer UI, control returns to the GST which takes the appropriate response based on the input.

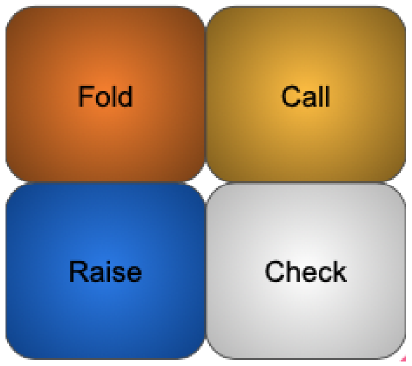


Fig. 8 Game State Update Interface

D. *Subsystem: Player UI*

The Player UI is a graphical user interface for the players at the table. The purpose of the Player UI is to display all relevant table information to the players during the game. The player UI communicates with the GST and retrieves all the necessary information from its modules. This information includes the pot size, player stacks sizes, bet sizes, the amount of players at the table, and which player's turn it is to act. The Player UI is designed to imitate an online poker game and to display the exact same information. An example of an online poker game is depicted in Figure 9, which we envision our Player UI to mimic. We will connect a monitor to the Raspberry Pi via an HDMI cable to display this graphical interface.

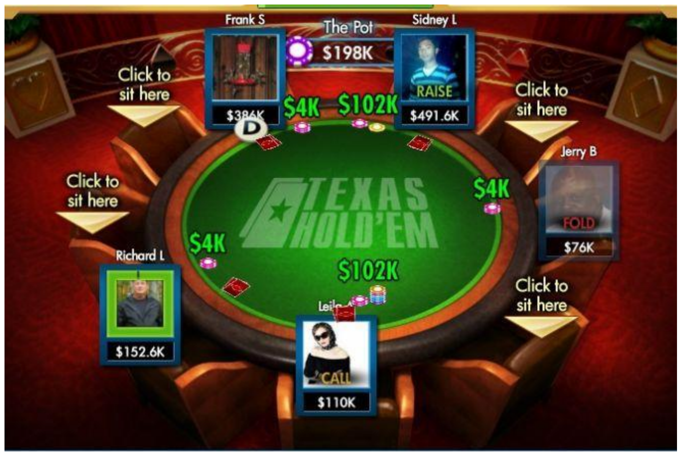


Fig. 9 Online Poker, https://pokerlivenews.com/free-online-poker-games/

VI. *Subsystem: Game State Tracker*

The Game State Tracker is the nucleus of our project. The role of the GST is to first and foremost to track the game state of the match, and relay this information to the Player UI. It's also responsible for the transfer control between the Dealer UI, Servo System, and Vision System.

Once initialization has occurred and a round has begun, the order in which the GST transfers power between subsystems proceeds as follows. First, the GST polls the Dealer UI for a player action. Given this action, the GST will update its internal game state then possibly transfer control to the Vision subsystem to update stack sizes, but then it will always need to transfer control to Servo subsystem to rotate to the next player. Then, the cycle repeats and GST will poll the Dealer UI for the next player's action. This flow within the software is depicted in Fig. 10 at the end of the document, and a pseudocode algorithm is given below:

```
GST.init()
dealerUI.init()
playerUI.init()
servo.init()
camera.init()

while !(exit):
    next_move = dealerUI.poll()
    GST.round_start = false
    switch next_move
        case add_or_remove
            dealerUI.add_remove_players()
            playerUI.update()
        case start_round
            Gst.round_start = true
        case calibrate_chip_colors
            cam.calibration_routine()

    while (GST.round_start):
        player = player_order.get_next()
        if player == null:
            GST.next_phase()
            continue

        input = dealerUI.poll()

        switch input
            case fold
                GST.remove(player)
            case bet
                servo.move(player)
                bet_size = cam.scan_stack()
                GST.pot_size += bet_size

        GST.update_player_order()
        dealerUI.render()
        playerUI.update()

    GST.calculate_payout()
    update_player_UI()
```

Fig. 10 Abstracted game state tracker pseudocode

VII. Project Management

A. *Schedule*

We organized our schedule into 3 phases: Brainstorming/Designing, Development, and Finalization. Right now we are nearing the end of the Brainstorming/Designing phase. The remaining tasks in this phase is to finalize our design choices in this document, and wait on the shipment of outstandings part orders.

The next phase will be the Development phase. Once each team member receives the parts they need to begin working, this phase will ramp up and include the bulk of our work throughout the semester. The Development phase entails setting up our Raspberry Pi environment, and developing each subsystem of our project within this Raspberry Pi environment. At the end of the development phase, we left time to thoroughly test each subsystem before the Finalization phase.

The Finalization phase could also be called the Integration phase. In this phase, our team will integrate each subsystem together and develop a final product. Our team will mainly be working virtually so we need to be very conscious about how we integrate our subsystems and develop interconnections. We believe this task isn't the least bit trivial, so we left a good portion of time at the end of semester to make sure this phase goes smoothly.

A full diagram of our schedule is depicted below in Figure 12, under the *Schedule* section.

B. *Team Member Responsibilities*

Each team member has more or less taken ownership of the subsystem they specialize in. Brandon has most background with Computer Vision, so he has taken the lead on developing the algorithm for scanning stacks. Zongpeng has the most background on hardware, so he has taken the lead on developing the Servo subsystem and writing the firmware for the Servo. Patrick has the most background on software development, so he has taken the lead on writing the Dealer User Interface and the Player User Interface..

The overarching subsystem that the entire team will be working on together is the Game State Tracker. The GST is the central part of the project which transfers control from subsystem to subsystem, so it is necessary that each team member contributes significantly to its architecture and development.

Lastly, the entire team is responsible for the overall system construction. This entails putting all the subsystems together and developing a product that will be user ready and presentable to the stakeholders/professors. The construction of the overall system will require knowledge of individual subsystem interconnections, so it is necessary that each team member contributes significantly to its production.

C. *Budget*

Our Bill of Materials can be found in Figure 13.

D. Risk Management

Component	Risk
Servo Motor	Motor provides enough torque; power supply; servo doesn't burn; servo burned
Computer Vision	Lighting changes, image noise
Software	Bugs leading to undefined behavior
Hardware	Successful hardware integration

For servo, to mitigate the risk, we could purchase additional hardware. For instance, if the servo burns out we could purchase another one; if the power supply is too small to provide enough torque, we could purchase a higher wattage power supply. These risk mitigation plans will work because we currently only used ⅓ of our budget, giving us room to buy replacements.

On the computer vision side, there are several ways we can mitigate inaccuracies in scanning. One way is to implement image pre-processing techniques such as blurring or closing to reduce the amount of noise. Another way is through the calibration routine. By having it accessible at the beginning of every round, the dealer has the ability to counter changes in lighting or chip color. While these methods aren't a guaranteed way to fix a catastrophic failure, we believe they are suitable enough to mitigate the smaller risks associated with the CV.

Unfortunately, there isn't much we can do to mitigate undefined behavior in the software other than rigorous testing. The best way we can deal with a crash is to crash gracefully and ensure the program restarts quickly. That being said, rigorous testing should allow us to catch and eliminate enough of the bugs to where this won't be an issue, especially given how little access an end user has to the underlying software subsystems.

One approach to hardware risk mitigation is simply to have multiple components ready for replacement; this way, if something fails we can quickly replace it. Combining this with a thorough electrical schematic and reading of each data sheet should allow us to wire up the system without likely risks of failure.

References

[1] RFID Blog. (2020, June 11). UNIQLO announced the introduction of RFID tags in 3000 stores worldwide during the year. rfidcard.com. https://rfidcard.com/uniqlo-announced-the-introduction-of-rfid-tags-in-3000-stores-worldwide-during-the-year/

[2] RFID Journal. (2013, March 10). Can I Scan Multiple RFID Tags Simultaneously When They Are Kept in Alignment? rfidjournal.com. https://www.rfidjournal.com/question/can-i-scan-multiple-rfid-tags-simultaneously-when-they-are-kept-in-alignment

[3] Poker Live News. https://pokerlivenews.com/free-online-poker-games/

Software Block Diagram

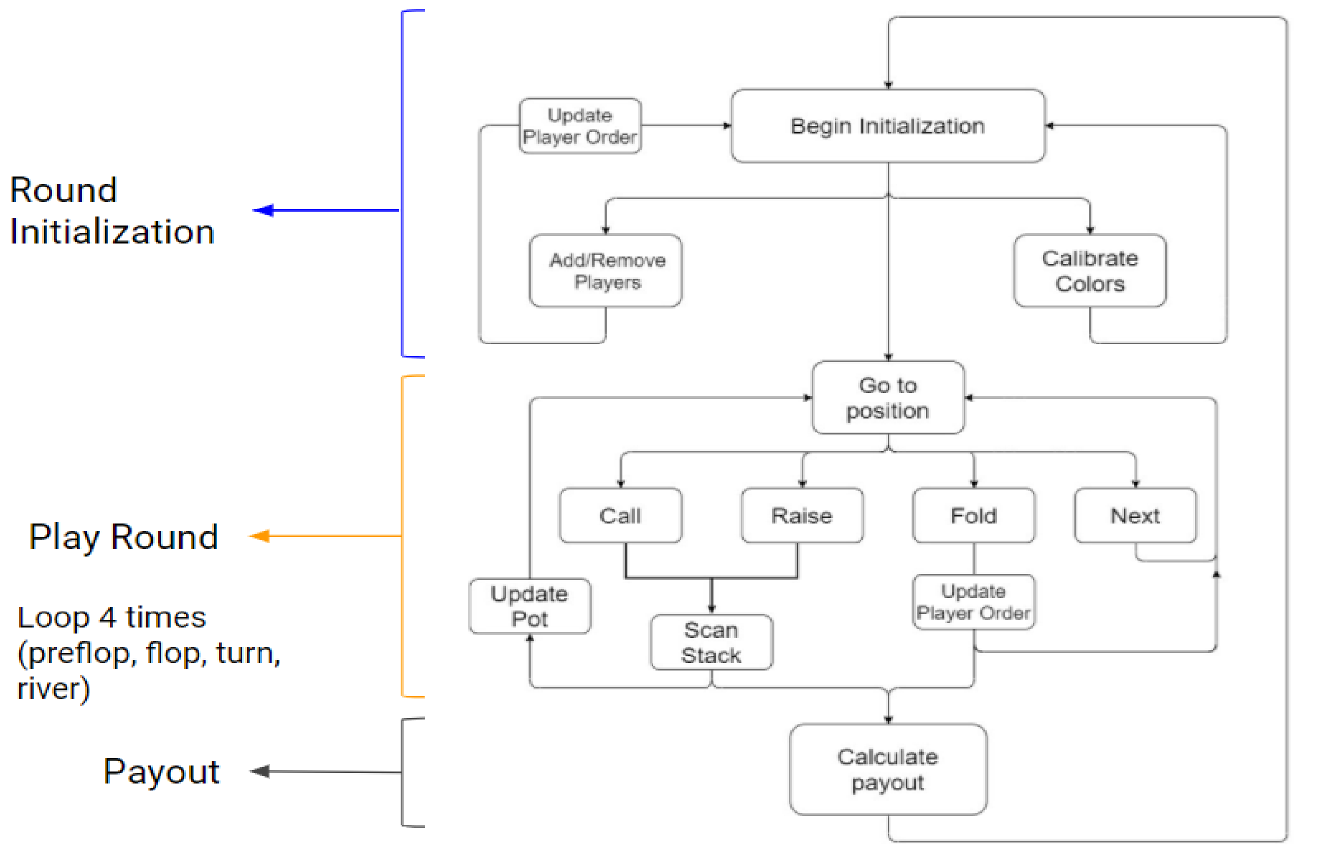


Fig. 10 Software Block Diagram

Overall CAD Model

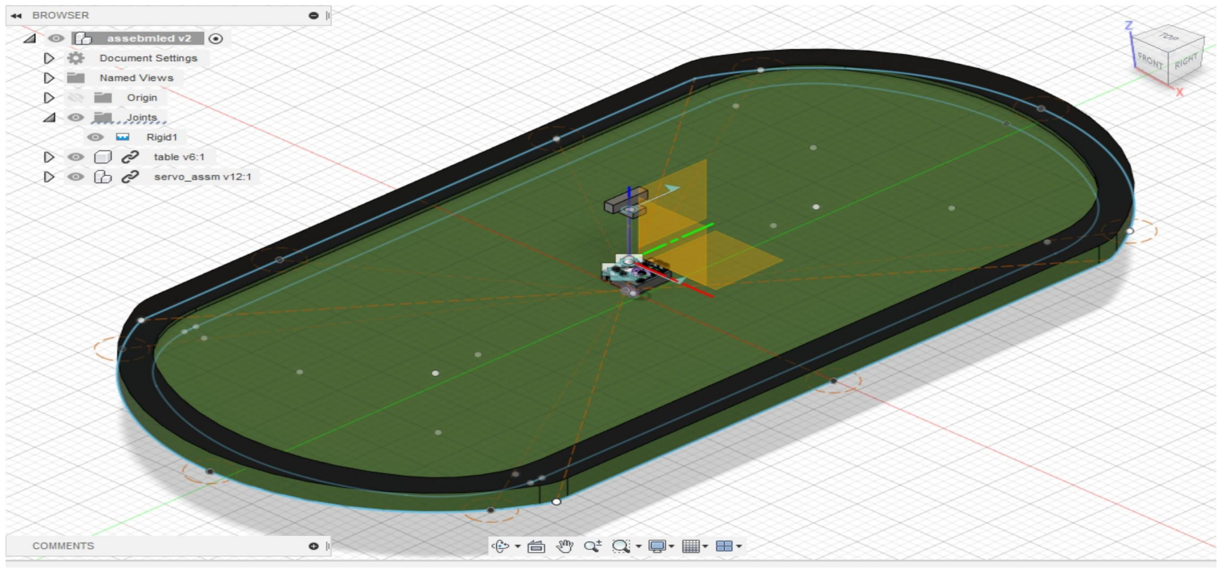


Fig 11: 3D model of assembly on poker table

Communications Block Diagram

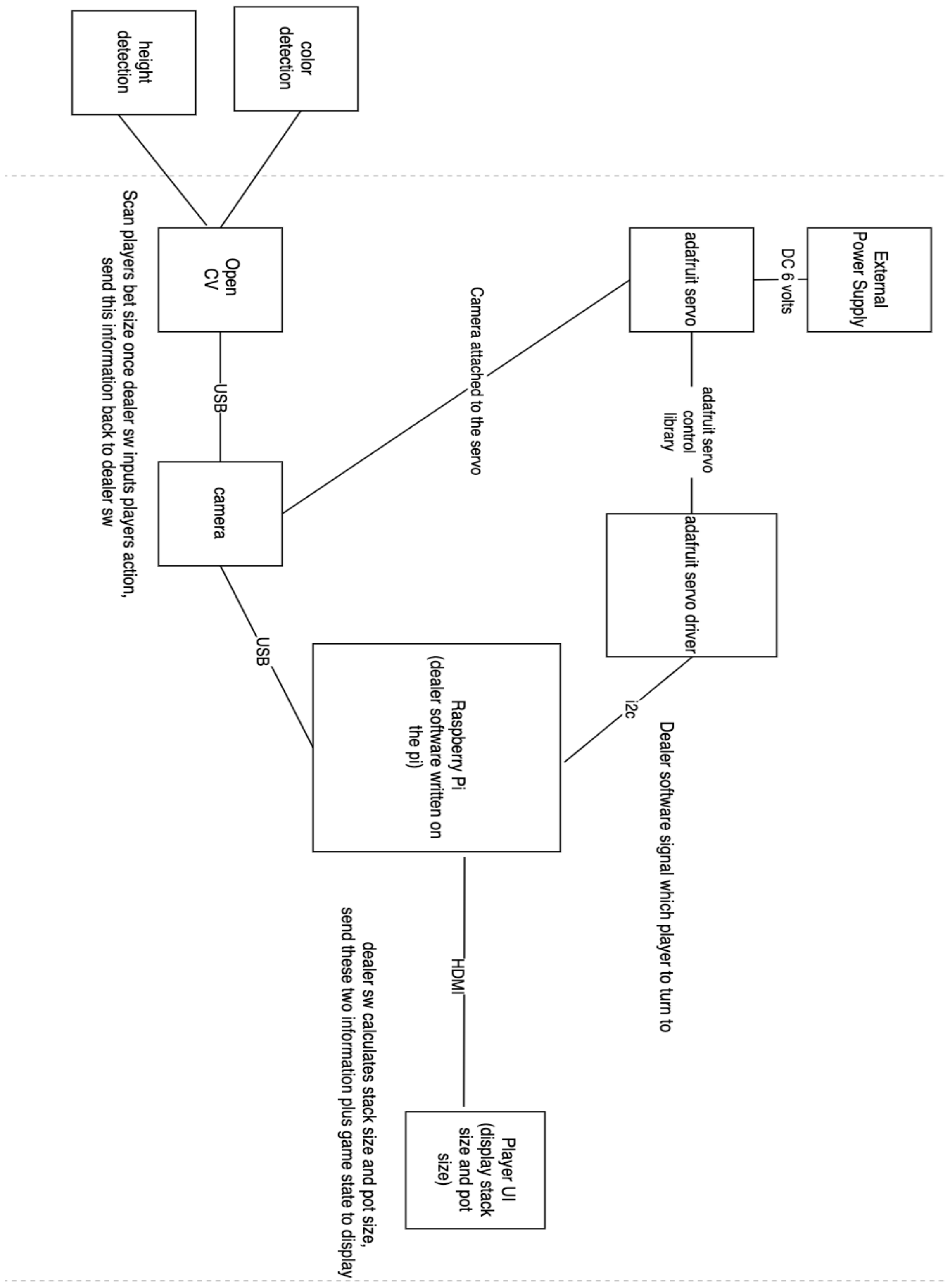
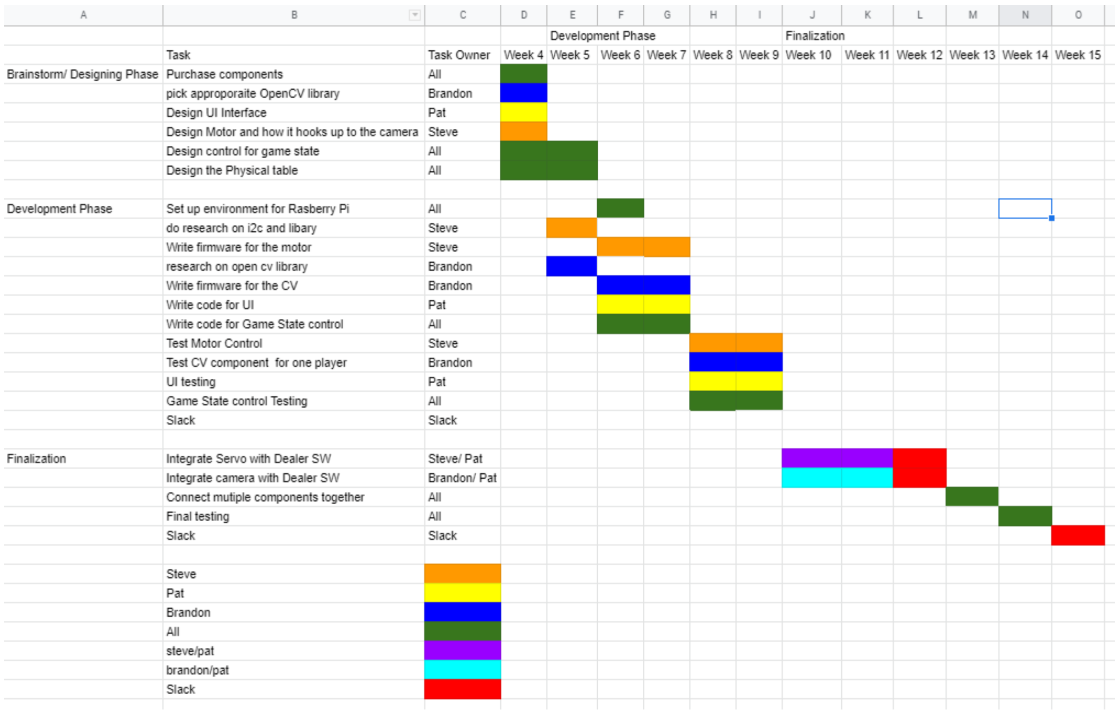

height detection
color detection
Open CV
USB
camera
USB
Scan players bet size once dealer sw inputs players action, send this information back to dealer sw
Camera attached to the servo
adafruit servo
DC 6 volts
External Power Supply
adafruit servo control library
adafruit servo driver
i2c
Raspberry Pi (dealer software written on the pi)
Dealer software signal which player to turn to
HDMI
Player UI (display stack size and pot size)
dealer sw calculates stack size and pot size, send these two information plus game state to display

Fig. 12 Communications Block Diagram

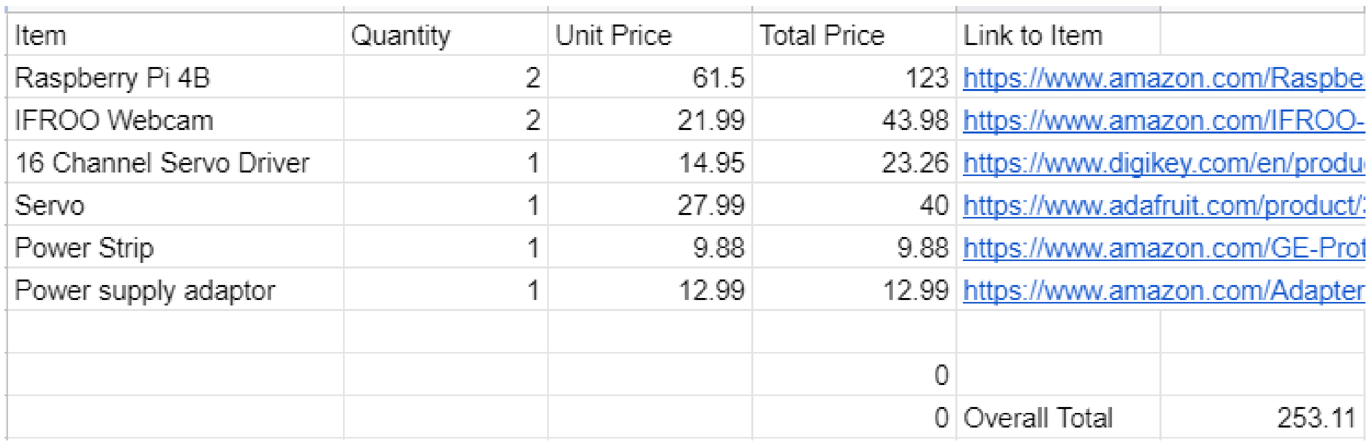
Schedule



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
				Development Phase					Finalization					
	Task	Task Owner	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15
Brainstorm/ Designing Phase	Purchase components	All	🟩											
	pick approporaite OpenCV library	Brandon	🟦											
	Design UI Interface	Pat	🟨											
	Design Motor and how it hooks up to the camera	Steve	🟧											
	Design control for game state	All	🟩	🟩										
	Design the Physical table	All	🟩	🟩										
Development Phase	Set up environment for Rasberry Pi	All			🟩									
	do research on i2c and libary	Steve		🟧										
	Write firmware for the motor	Steve			🟧	🟧								
	research on open cv library	Brandon		🟦										
	Write firmware for the CV	Brandon			🟦	🟦								
	Write code for UI	Pat			🟨	🟨								
	Write code for Game State control	All			🟩	🟩								
	Test Motor Control	Steve					🟧	🟧						
	Test CV component for one player	Brandon					🟦	🟦						
	UI testing	Pat					🟨	🟨						
	Game State control Testing	All					🟩	🟩						
	Slack	Slack												
Finalization	Integrate Servo with Dealer SW	Steve/ Pat							🟪	🟪	🟥			
	Integrate camera with Dealer SW	Brandon/ Pat							▨	▨	🟥			
	Connect mutiple components together	All										🟩		
	Final testing	All											🟩	
	Slack	Slack												🟥
	Steve	🟧												
	Pat	🟨												
	Brandon	🟦												
	All	🟩												
	steve/pat	🟪												
	brandon/pat	▨												
	Slack	🟥												

Fig. 13 Schedule

Budget



Item	Quantity	Unit Price	Total Price	Link to Item	
Raspberry Pi 4B	2	61.5	123	https://www.amazon.com/Raspbe	
IFROO Webcam	2	21.99	43.98	https://www.amazon.com/IFROO-	
16 Channel Servo Driver	1	14.95	23.26	https://www.digikey.com/en/produ	
Servo	1	27.99	40	https://www.adafruit.com/product/	
Power Strip	1	9.88	9.88	https://www.amazon.com/GE-Prot	
Power supply adaptor	1	12.99	12.99	https://www.amazon.com/Adapter	
			0		
			0	Overall Total	253.11

Fig. 14 Bill of Materials

Servo Subsystem diagram

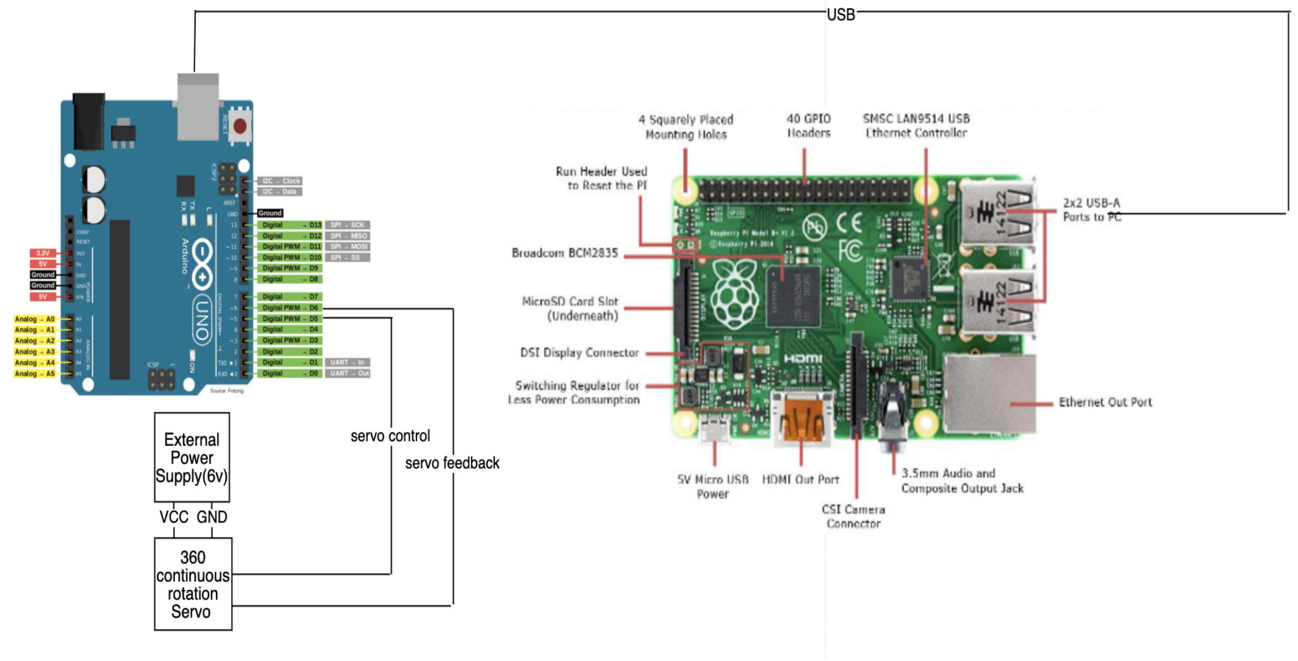

USB
External Power Supply(6v)
VCC GND
360 continuous rotation Servo
servo control
servo feedback
4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
2x2 USB-A Ports to PC
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
3.5mm Audio and Composite Output Jack
CSI Camera Connector

Fig. 15 Servo Subsystem Diagram

Electrical Schematic

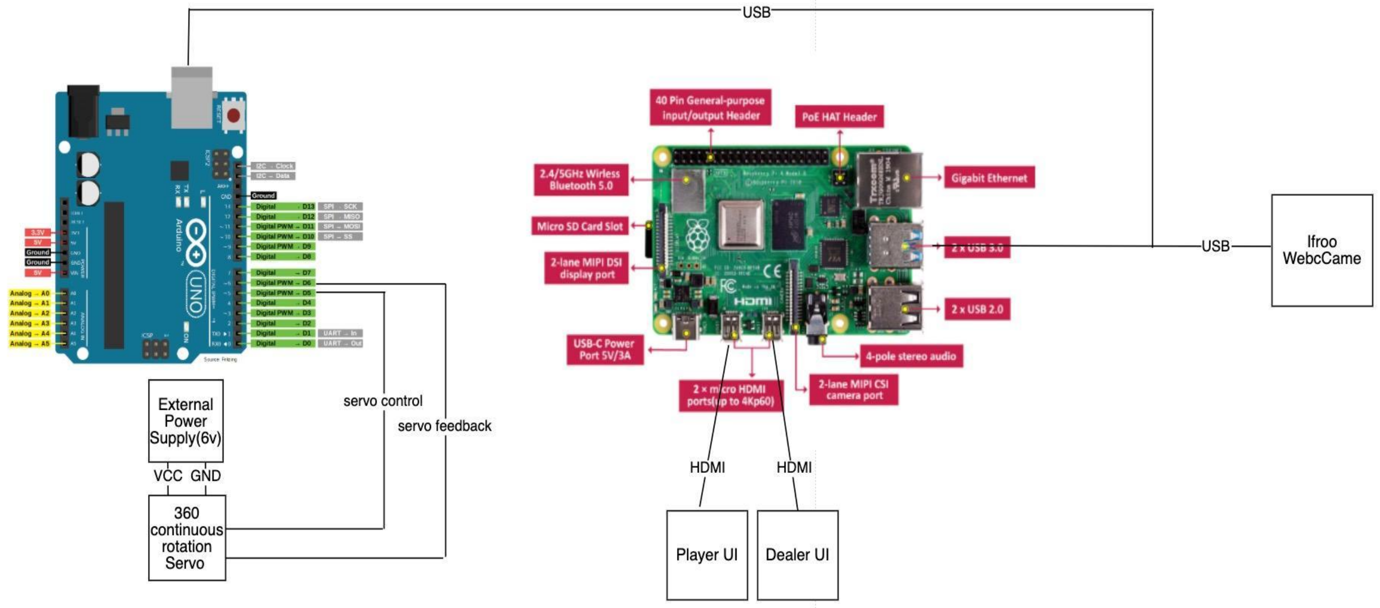

USB
External Power Supply(6v)
VCC GND
360 continuous rotation Servo
servo control
servo feedback
40 Pin General-purpose input/output Header
PoE HAT Header
2.4/5GHz Wirless Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2 x USB 3.0
2-lane MIPI DSI display port
2 x USB 2.0
USB-C Power Port 5V/3A
4-pole stereo audio
2 × micro HDMI ports(up to 4Kp60)
2-lane MIPI CSI camera port
HDMI
HDMI
Player UI
Dealer UI
USB
Ifroo WebcCame

Fig 16. Electrical Schematic