

SE101 Team 7 Final Report - *BetterBlackjack*

Group Members: Andrew Shen, Bryan Cui, David Li, Edison Du, Qiyue Chen

Date: December 6th, 2023

Minimum Viable Prototype:

1. Be a functional card dealer capable of scanning and recording dealt cards
2. Have a working version of Blackjack, with the human player playing with an AI dealer
3. Accept user input via buttons (hit/stand, changing bets, etc)

What We Achieved:

1. Card dealer scans and correctly identifies Victoria Playing Cards reliably, but may not be as accurate for other decks of cards. The camera works but can be sluggish (5 seconds to deal a card). The card dealing rollers need to be maintained consistently (e.g. re-apply sticky tack for grip) but work well otherwise.
2. A single-player version of Blackjack and Blackjack Dealer was implemented successfully.
3. The buttons and LCD screen for user interaction work consistently and allow for complete gameplay, but the betting feature can be unforgiving (only allowing players to increase their bet) due to the limited buttons available.

Evidence Of Achievements:

1. The card dealer/dispenser dealt and scanned three complete sets of 52 cards with only 1-3 hiccups per deck. The image recognition algorithm successfully determines the value of all playing cards in each set. However, the algorithm is unable to determine if a card does not have a value at all (e.g. jokers, images of the ceiling, etc).
 - a. These images can be found here: [testImages](#). One example of a hiccup is the missing Queen of Clubs between “testImages/cardPhoto/p22.jpg” and “testImages/cardPhoto/p23.jpg”.
2. The code correctly handles this list of special Blackjack cases: [Blackjack Cases](#) (e.g. the first clip in the video demo is case 6, the last clip is case 7). Many games were played with randomly shuffled decks without error in-game logic.
3. The final clip in the video demo shows a complete game loop with the LCD and buttons working.
4. The program runs when the RPi turns on, confirmed since the LCD displays the game. The bash script was particularly frustrating to set up due to the different privileges and Python libraries the user and root have access to.

Technical Details on Achievements:

Raspberry Pi

- Soldered 40 header pins.
- Wired servos, camera, breadboard, LEDs, LCD, buttons to GPIO pins of RPi.
- Created a bash script to run the main game program on boot.

Card Holder

- Designed a modular 3D model in TinkerCAD.
- Printed and assembled the 3D-printed parts.

- Secured white LED beside the camera module for consistent lighting conditions.

Card Dispenser/Roller

- Secured 3D printed rods (wrapped in sticky tack for increased friction) to continuous servos.
- Utilized the “AngularServo” module from “gpiozero” library to program the servos’ direction and speed.
- Fine-tuned servo active times to dispense exactly one card at a time.
- Wrote roller class in the main game for improved readability.
- Wrote a Python script automating the image collection process of cards (including a blank card).

Image Recognition:

- Pillow (Python Imaging Library) is used to perform various image processes to get clean edges on the card’s value (A, 2, ... Q, K).
 - The contrast is increased and the image is converted to grayscale. This is necessary for red cards that are very close to white (the background colour) on the grayscale.
 - Image subtraction: An image of a white card taken beforehand is subtracted from the image to normalize lighting effects (e.g. glare from LED).
- NumPy is used to convert the image to binary (black or white) by carefully choosing a threshold value (all pixels under the threshold become black, all above become white).
- This whole process can be seen here: [process.avi](#)
- A bounding rectangle around the largest shape is extracted, resized, and compared to a set of 13 [templateImages](#) using bitwise-XOR and counting white pixels to find the closest matching shape.
 - A breadth-first search algorithm is applied to “edges” in the image to get the minimal and maximal *x* and *y* coordinates of every “shape”.
 - Edges are pixels that are adjacent to pixels of the opposite colour.
 - A bounding rectangle is created with these minimal and maximal coordinates. The largest rectangle bounding a valid shape (completely enclosed, no holes) is considered.
 - By carefully cropping the image beforehand, this largest shape is consistently the card’s value in all our cases. In the case of “10”, this shape is the “0”.
- Template Images were manually generated with this same algorithm on a handpicked set of cards.
- Wrote a Python class to modularize the process.

Buttons:

- Created two pull-up circuits containing simple push buttons on the breadboard, connecting the RPi’s GPIO and GND pins.
- Wrote button class and function that pauses the program until a button input is registered and returns the ID of the button pressed.
- Registers exactly one button press at a time.

Main Game:

- Prompt the player by displaying text messages on an LCD using the “rpi_lcd” library.
- Updates the player on the state of the game throughout the game (displays both the dealer and player hands).

- Player class keeps track of player and dealer hands as well as the player's balance throughout multiple games.
- Dynamic calculation of "hard" and "soft" hands (i.e. update the value of an ace)
- Takes input on bets, allowing users to increase or confirm their bets.
- Double down option is also included
- Hit/stand phase of the game is skipped if there is a player/dealer blackjack
- Captures photos from the camera using the camera module from "picamera" library.
- Determines if the player wins or loses, adds to/subtracts from player balance accordingly, displays appropriate winning/losing screen

Professional Issues:

Safety

- Users should take care not to place their hands or other belongings into the rollers.
- As with any addiction, gambling can take serious tolls on mental, emotional, and physical health. We recommend users to take breaks between sessions and spend in-game money responsibly.

Privacy

- Our only area of concern is the camera. The console stores exactly one photo at all times – the photo of the card currently being scanned. If the user looks into the console from above whilst playing a game, and when there isn't a deck present, the camera may take a photo of the user. This photo is stored on the device, which is a potential privacy concern if we do not have consent from the user to take their photo.
- All data is stored locally on the Pi, minimizing the risk of personal data leaks

Intellectual Property:

- Rpi_lcd is MIT
- NumPy is BSD
- Pillow is HPND
- Tinkercad is Free Use

Since the license of all 3D party software used (without modification to the original code) in BetterBlackJack are classified as permissive free software, we are free to redistribute our project without restrictions on licensing.

Major Challenges and Research Done:

Card Dispenser/Rollers:

- One of the challenges we encountered was making the card dispenser constantly eject one card at a time. This was a particularly difficult problem because (1) the frictional force between the ejected card and the roller changes as cards are dispensed (less cards weighing down on bottom, normal force changes) and (2) the friction between two cards causes two cards to be dispensed. We solved this problem by raising the roller slightly above the bottom of the card holder, fine-tuning the height of the slit to allow only a single card to be rolled out, and adjusting the friction of the sticky tack.

- The following problem we encountered was that although only a single card was dispensed, the card above it shifted along with the ejected card and the value of the next card was no longer visible to the camera. We solved this challenge by making the inner servo turn the opposite way to push it back to its proper position for the camera to take a photo.
- Gravity caused the sticky tack we wrapped around the rollers to hang down after periods of inactivity, changing its shape. This caused an issue with the rollers unable to make reliable contact with the cards. The issue was fixed by manually reshaping the sticky tack during setup. The card dispenser was also able to calibrate itself to a degree by repeatedly rolling out cards, creating a more even surface.

Camera

- One major issue affecting the consistency of our image recognition algorithm is the lighting. Since our algorithm relies on thresholding the image to black and white, this can be very variable depending on the lighting. To fix this, we added an LED light next to the camera to normalize the lighting.
- When capturing an image, the camera has to wait a couple seconds to adjust to the lighting and focus. However, this process is repeated for every photo, which lengthens the image recognition process. We learned about the camera calibration process, and resolved this problem by switching from using the terminal command “raspistill” to the Python library “picamera”, which offered us the ability to adjust the wait time between image captures. Through trial and error, we found a compromise between the image consistency and wait times.

Image Recognition

- A lot of research and trial and error had to be done to find the algorithm that worked best for us. The initial research and algorithms all used OpenCV. We looked into cascade classifiers (machine learning), feature detection (ORB, SIFT, and FAST), contour matching, and image subtraction before finalizing our algorithm.
- Cascade classifiers identify images with machine learning. We decided this would not be optimal for our purposes, since it would require us to train machine-learning models from scratch (we would have to take or “steal” hundreds of photos). Not only that, but the OpenCV cascade classifier only trains on positive or negative samples, meaning each classifier is only tuned to detect images of one type of “thing”. This means that to identify the value of a card, we would have to train 13 different classifiers, which seemed computationally excessive.
- Feature detection is a fast algorithmic approach that finds “corners” in the image, and compares these corners. However, due to the nature of the algorithm, feature detection can’t differentiate between 6 and 9 because they are geometrically identical, just rotated.
- Contour matching looks at shapes in the image and compares them. It suffers from the same issue as feature detection, in that it can’t differentiate between 6 and 9s at all. Overall, contour matching was much less reliable than feature detection. Simpler shapes like circles had a very close match with all other shapes, so the card values 6, 9, and 10 would come up very often while complex shapes like A were harder to detect.
- Image subtraction (literally subtracting the pixels of a template image from the image) was more reliable since it accounts for the orientation of 6 and 9 to differentiate them. We found that bitwise-XOR was stronger than image subtraction since image subtraction only measures the

pixels that are the same, XOR factors in pixels that are the same as well as pixels that are different. However, direct pixel comparison methods are only effective if the images we take overlay on top of the template images well. Since there is significant shifting between images due to the card roller not being perfect, our final algorithm performs cropping, edge detection, and resizing before applying the bitwise-XOR method.

Software

- The development of software required to run our game was the first time our group used a version control system (Git) to work on a project. Consequently, much of the development time in the infancy of our project was on learning to navigate and manage problems with Git (e.g. merge conflicts, unfinished code).

Unable to Install OpenCV

- Although we did extensive research in OpenCV and our algorithms originally used OpenCV, we couldn't install it onto the Raspberry Pi. Upon conversing with other groups, we discovered that OpenCV is incompatible with our model of RaspberryPi (Zero W). We fixed this by re-implementing the algorithm using Pillow and NumPy, which we were able to install.
- Certain features that were built into OpenCV had to be coded from scratch, such as the contour area finding method, which we had to write a custom breadth-first search and edge detection algorithm to compensate for.

Team Member Contribution:

Andrew

- Designed the 3D model of the card holder.
- Wired hardware components to the Raspberry Pi.
- Wrote classes for rollers and buttons.
- Video editor and screenwriter for project video; small cameo role.

Bryan

- Integrating buttons and LCD into main game logic (the code in main.py)
- Lead actor and talent manager for project video.

David

- Design and implementation of main game logic (the code in main.py)
- Video editor, cinematographer, and cameraman for project video.

Edison

- Researched OpenCV, primarily cascade classifiers, contour matching, and bitwise-XOR.
- Developed the final image recognition algorithm in Pillow and NumPy
- CEO of BetterBlackjack in project video.

Qiyue

- Researched OpenCV, SIFT feature detection algorithm for image recognition
- Soldering header pins on RaspberryPi



- Video editor and lead cinematographer for project video.

References:

Card dispenser design inspired by:

-  Rigged Card Sorting Machine - ALWAYS Get The Hand You Want! .

Image Recognition

- OpenCV documentation
 - [Cascade Classifier Training](#)
 - [OpenCV: Contours : Getting Started](#)
 - [Contour Features](#)
 - [Feature Matching](#)
 - [Arithmetic Operations on Images](#)
- NumPy & Pillow
 - [NumPy reference — NumPy v1.26 Manual](#)
 - [Pillow docs](#)
- Videos
 -  Feature Detection and Matching + Image Classifier Project | OPENCV PYTHON
 -  Playing Card Detection Using OpenCV-Python on the Raspberry Pi 3 + PiCamera