UNOmatic

Authors: Thomas Kang, David Peng, Jason Stentz
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—UNOmatic is a system capable of tracking the entire UNO game state to automatically deal cards, track illegal moves, and perform scoring each round. It aims to eliminate as much pressure as possible for players, providing them with an experience to focus on the social aspects of playing card games. It is also able to assist dealers during tournaments to create smoother and faster game play.**

***Index Terms*—automatic card dealer, CNN architecture, machine learning, UNO cards classification, web sockets**

1 INTRODUCTION

Ever since its inception in 1971, UNO has remained one of the most popular playing card games in the world as it surpasses 151 million copies sold worldwide[1]. Recently, to celebrate its 50th anniversary, there was a professional tournament held, demonstrating its popularity from a household environment all the way to full-scale competition. Due to the game's vast popularity, there are many conflicting rule sets. One such discrepancy is the ability, or inability, to stack +2 and +4 cards. In a casual setting, rule disputes can lead to arguments and negatively impact the user experience. In a tournament setting, it is absolutely imperative to enforce a standard set of rules followed by all participants.

Alongside the problem of vastly different interpretations of the rules, there is also a lengthy scoring process at the end of each UNO round. Once someone runs out of cards, therefore winning the game, the remaining cards of the losing players must be meticulously tallied. This task can be very tedious, especially given a player's card count is potentially unbounded. Alongside being time-consuming, the process is a pressure point for scoring mistakes, if not done carefully.

UNO serves as a popular means of fostering social interaction, making it imperative that all participants feel included and comfortable playing. One of the primary barriers to entry for newcomers is the uncertainty surrounding the game's rules. Teaching someone the game can consume valuable playing time, particularly if additional players wish to join later on.

To address these problems, we propose UNOmatic, a fully automatic UNO game controller, equipped with card dealing, full state tracking & validation, round scoring, and a website. Using two cameras connected to a Raspberry Pi and an onboard card classification model, the device is able to track all players' hands, the discard pile, and the remaining cards in the draw pile without any trouble for the user. Maintaining this state information allows for automatic control flow, rule enforcement, scoring, and a website-based game display. The system as a whole takes the burden of managing the game flow off of the user in an elegant way that further enhances the player experience.

2 USE-CASE REQUIREMENTS

We have identified the following use case requirements:

1. Size: The machine should be usable on most surfaces on which people would play UNO. This includes a reasonably sized table as well as on the floor. Thus, we require that the machine be usable on at least a 3 ft by 3 ft table.
2. Longevity: The machine should be able to be used for the duration of a competitive UNO tournament, which according to past UNO championships, is around 7 games[2].
3. Automatic gameplay: The machine should move to each player and deal cards seamlessly with minimal hiccups. Thus we require less than 10 errors in any of the subsystems when going through the deck once.
4. Real-time website updates: Spectators and players want real-time updates on the state of the game. If an error occurs, users should be able to correct it from the website quickly.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Fig. 1 is a high level block diagram of our system. The Raspberry Pi acts as the main controller of the whole system, issuing commands to the various attached devices. Players press buttons to signal they have placed cards or ended their turn, which triggers the Pi to take pictures from the cameras, classify them using the CNN Card Classifier, and issue motor commands to the Arduino. The Pi also handles website communication, sending out updates, and receiving corrections from users in the case of errors. This website can be hosted either on the Pi or an external device.

The software inside the Pi has been broken into three main subsystems: the controller, the UNO state, and the displayer(s). These subsystems act asynchronously from one another and communicate through message passing like nodes in a distributed system. Each of them waits on a

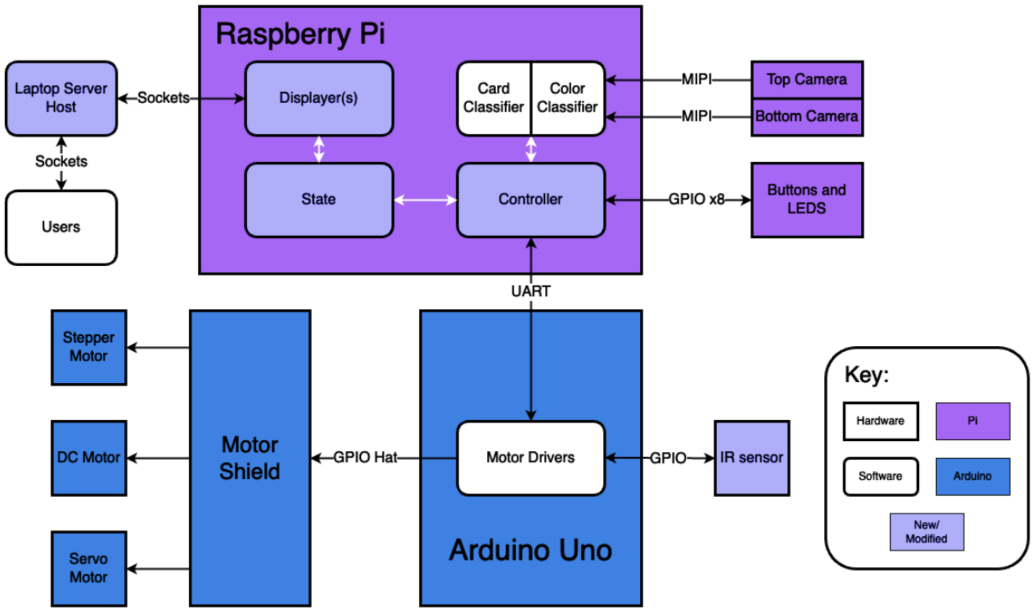


Figure 1: Top Level Diagram

thread-safe queue of requests, able to respond and act on any request at any time. The controller is responsible for handling IO. The UNO state is responsible for responding to the actions of users within the rules of UNO gameplay. The displayer(s) are responsible for making updates to the game state viewable to any players or spectators. These independent, asynchronous nodes allow for easy handling of interruptions, such as calling UNO or making corrections to the state.

The Arduino Uno handles all motor control and communicates with the Raspberry Pi over a UART interface, receiving text commands and communicating back when it is done. The motors are controlled using the Adafruit Motor Shield V2 which handles power and low-level signals. Also, we have added an IR sensor to confirm a card has been successfully dealt. This takes the burden off the user to confirm they have received a card.

On the software side, one of the main engineering principles we enforced was separation of concerns. It was a major emphasis in our design that all of the logic handling UNO control flow would remain within the state subsystem. The implementer of the controller interface does not need to understand the full details of UNO. Similarly, the displayer implementer is not required to understand the full mechanics of UNO. This separation allowed us to easily track and fix bugs, since none of the logic from any one subsystem bled into another subsystem.

One of the core technologies we relied upon was thread-safe asynchronous queues to pass messages between the software components of our system. These queues proved invaluable to us as a method of allowing our code to respond to asynchronous events at any time while still communicating between the various threads safely. They also became a point of serialization for all of the events that happened and made logging and debugging the system trivial, as we could observe the events entering and exiting the queues.

In order to get the best final design for the card dealer, we went through multiple iterations. Each time we had a failure, we devised a hypothesis for the failure and changed one variable at a time to confirm it. After multiple iterations of this scientific experimentation, we arrived at a solution that satisfied our requirements.

Before we purchased the motors for the chassis rotation, we wanted to guarantee that the stepper motor would meet our design requirements. In this process, we calculated the minimum torque required to rotate the chassis. This incorporated gear slip, force transfer, and friction, all using mathematical equations. We made sure to add a margin of error to be conservative in our calculations.

4 DESIGN REQUIREMENTS

1. Size: Dimensions should be less than 1 ft by 1 ft. This stems from our size requirement which requires the machine to fit comfortably in the center of a small square table, approximately 3 ft in width. This will give ample space for players to rest their cards and interact with the machine comfortably, even when sitting on the ground close to each other.

2. Battery Life: Our batteries should last more than 1 hour. This stems from our longevity requirement of being able to play 7 games. On average, games last 5-10 minutes so this meets this requirement.

3. Motor Power Draw: Due to the current limitations of the Adafruit Motor Shield V2, we are limited to drawing 1.2 Amps at 12V to each motor.

4. Rotation weight: We estimate the weight of our machine to be around 7-8 pounds. Thus, the motor

should be able to rotate all of this weight.

5. Rotation speed: In order to meet our longevity requirement, the machine should rotate quickly from player to player in order to allow moves to be played at a normal pace. Thus, we require rotation to the next player to take < 3 seconds.

6. Rotation accuracy: To meet our automatic gameplay requirement, the rotation to each player should point at them within $\pm 10°$. This will ensure that gameplay does not have to pause to reorient the machine or pick up cards that have been dealt in the wrong place.

7. Classification: To meet our automatic gameplay requirement, the card classifier should have an accuracy $> 95\%$ and have a latency of < 1 second. With 108 cards in a deck, this will allow for minimal disruptions of the game as well as not slowing down the game waiting for classification as this can be interleaved with the physical motor movements.

8. Card dealing: To meet our automatic gameplay requirement, dealing one card should take < 1 second. It should also deal only one card at a time at least 95% of the time. This will allow card dealing to be a quick and seamless process with minimal interruptions.

9. Website: To fulfill the real-time website updates requirement, we require the end-to-end latency of updating the website be < 2 seconds.

5 DESIGN TRADE STUDIES

5.1 Hardware Design

5.1.1 Central Rotation

We chose an internal gear and a spur gear system to rotate the system and small caster wheels to support the system. This system provides very accurate rotation based on our gear ratio. It is a lot more stable and uses less force than using the stepper motor's axle acting as the central axle and moving the top platform. The caster wheels with low rotational friction prevent the rotating spur gear from touching the bottom platform and creating more friction. We have explored using a lazy-susan bearing which provides low-friction rotation of a platform without us having to manually align the wheels. We needed it to be at least an inch in height due to the shaft length, 20 centimeter internal diameter to accommodate the internal gear system, and 25 centimeter external diameter for the top chassis to stay compact. Since most consumer-level lazy-susan bearings are used for low-profile tables, we were unable to find a product that fits our specific dimensions but also costs less than the caster wheels.

5.1.2 Dealer

The cards in our system will be dealt from the bottom of the deck. While it is mechanically a lot easier to deal from the top, having to take a picture of each card makes dealing from the bottom a better option. In case we want to deal from the top, we could have built an IR sensor circuit to detect card movement and take a picture as it leaves. However, the card has to leave at a reasonable speed, leaving us with a very small amount of time to capture the card's symbol and color, and low resolution could lead to poor classification accuracy. With the card being dealt from the bottom, the camera has a lot larger window to take a picture of the card before being dealt to players, and it will always be the same cutout. However, we do have to make sure we have enough friction on the roller and pressure on top of the cards to make sure we only deal one card at a time.

5.1.3 Motors

Our system requires motors at 2 places: chassis rotation and card dealing. As mentioned in 5.1.1, chassis rotation uses an internal gear-spur gear system where the spur gear is connected to a motor and rotates. Card dealing also takes 2 motions: extruding the card forward toward the exit of the dealer and ejecting the card at a higher speed toward the players. For these 3 motions, we chose a stepper motor, a continuous servo, and a DC motor respectively. To control these 3 motors, we have purchased an Adafruit motor shield. It provides libraries for all 3 types of motors, and we don't need to build additional circuitry to control them or distribute power. This, however, puts a 12V, 1.2A limit per motor.

For card extrusion, we needed fine control or we would push out multiple cards. For this, we have selected a continuous servo motor that gives us precise control, is compact and light, and also turns continuously so that it can keep extruding continuously without having to return back to its position before making another extrusion.

For card ejection, we chose a DC motor with high RPM to rotate the rollers quickly. Neither a stepper nor a servo would have been able to give $200 \sim 300$ RPM.

For the chassis rotation, we need a precise and powerful motor to move the entire chassis which will weigh about 4 kilograms. A stepper motor or servo motor would have been appropriate, but steppers have high torque and precise control over rotation. Since the upper body of the design sits on wheels and rotates horizontally via gears, we have calculated the force needed to rotate our system with the following design specifications:

- The gears we use have a $20°$ pressure angle, which is a widely used metric for standard gear systems.
- The contact point of the gears from the center of the spur gear is approximately 2.2cm.
- Based on [3], we will be using 0.04 as our friction coefficient(μ_r). This is equivalent to Nylon wheels on

concrete, which is most likely a lot higher than our actual value, but to simplify calculations and be conservative with our values, we have chosen 0.04 specifically.

- Based on [4], the amount of force needed to start a motion is about 2.5x of sustaining the motion.
- Approximately 70% efficiency in gear force transmission.

With this and the calculations below, we chose a 26Ncm Nema17 stepper motor. It not only provides us with a generous amount of torque in case we add more parts than our expectation but also only draws 0.8A at 12V, which fits our criteria.

Calculations:

Rotation force:

$$\begin{aligned} F_r &= W \times \mu_r \\ &= (4kg \times 9.81m/s^2) \times 0.04 \\ &= 1.570N \end{aligned} \tag{1}$$

Static rotational force:

$$\begin{aligned} F_{\text{static r}} &= F_r \times 2.5 \\ &= 1.57N \times 2.5 \\ &= 3.924N \end{aligned} \tag{2}$$

Gear's force:

$$\begin{aligned} F_{\text{gear}} &= \cos(20^\circ) \times \tau_{\text{motor}}/2.2\text{cm} \times 70\% \\ &= 0.29897\text{cm} \times \tau_{\text{motor}} \end{aligned} \tag{3}$$

Minimum motor torque:

$$\begin{aligned} \tau_{\text{motor}} &= F_{\text{static r}}/0.29897\text{cm} \\ &= 3.924\text{N}/0.29897\text{cm} \\ &= 13.125\text{Ncm} \end{aligned} \tag{4}$$

5.1.4 Power Source

Our entire design runs off of 2x 12V 3000 mAh batteries and a 9V 600 mAh battery. Using this, we would like to achieve 1+ hour battery life. The 12V batteries will power the stepper motor, DC motor, and the RPI 5, while the 9V battery powers the Arduino Uno board itself and the servo. We have chosen these specific 12V batteries because of their ability to supply 12V 3A, which is 36 watts. The RPI 5 can be powered by a regular 15W battery bank, but it will go on low-power mode that limits the amount of current going to the peripherals to 600mA instead of 1.6A with the correct power supply. This poses a problem as we use 2 Pi cameras as well as many GPIO pins for communication. Thus, to supply the RPI 5 with the correct input voltage and amperage, we have bought a 36Wh battery with a variable buck converter that steps it down to 5V with up to 5A output. With our battery capacity, even in the worst case of the RPI 5 running full throttle at all times, we are still able to get 1+ hour of battery life. For the motors, the stepper and the DC combined used about 1.1 A at 12V while moving. Considering the battery capacity, we should be able to get well over 2 hours of battery on the motors. Finally, the Arduino usually draws about 50 mA from the board itself and 160 mA from the servo. Thus, with our 9V 600mAh capacity, we get about 3 hours of battery life, which is sufficient.

5.2 Microcontrollers

5.2.1 Raspberry Pi 5

For our main controller, we are using a Raspberry Pi 5. We need to be able to handle two camera streams, and only the Raspberry Pi 5 supports two simultaneous camera streams. Also, the Raspberry Pi 5 has around 2x the performance of the Raspberry Pi 4 which will greatly speed up machine learning inference and overall system responsiveness. We also considered using an Nvidia Jetson Nano, however, it has a much larger volume and weight due to its large heatsink.

5.2.2 Arduino Uno

While the Raspberry Pi is a very good platform, it was not meant to handle real-time IO. For this purpose, the Arduino Uno is a much better option. It has native 5V GPIO, real-time environment, and access to the powerful Adafruit Motor Shield V2. As mentioned earlier, this motor shield allows us to control up to 1 stepper, 2 DC motors, and 2 servos all at the same time. To communicate between the two microcontrollers, we are using a UART interface over USB. We chose this over other serial interfaces such as I2C because UART allows for multi-master functionality and can use the USB port which is convenient. Multi-master allows the Arduino to signal to the Pi when the Arduino is done moving the motors.

5.3 Software

5.3.1 Card Classification Model

For the model architecture, we have considered three main options. The first is a standard convolutional network. If this model is able to achieve a high accuracy, it would be ideal given our computationally limited environment, but it may not be complicated enough for classifying with 15 labels.

We also considered using a vision transformer architecture, trained on our dataset from scratch. This will require more compute time, but it should theoretically be better at generalizing to unseen examples and learning long-term dependencies across the image.

Finally, we considered using a pre-trained transformer, such as BERT, which is good at turning images into a lower-dimensional representation. We would then fine-tune the model to fit our specific use case. This will make our model more complex but may increase accuracy.

We have decided to stick with a convolutional neural network since it is the most lightweight approach that maintains the ability to learn spatial relationships.

5.3.2 Color Classification

After classifying the type of card, the system must then determine the color of the card. There are two main ways of solving this problem – naive computer vision techniques and more machine learning techniques. The naive computer vision approach would be to independently apply red, yellow, blue, and green color filters to the image and compute average intensities. The filter that produces the highest average intensity reveals the color of the card. This method would result in very speedy color classification, but it would require specific tuning for the top and bottom cameras to be resilient to lighting changes. Alternatively, the color can be classified by another convolutional network. This method is slower, but it is capable of learning to be resilient to lighting conditions if given enough data. For our purposes, we decided on using more machine learning to classify the card colors.

5.3.3 Software Control Model

Throughout the design and implementation of our software systems, we wrestled with two main philosophies for handling the software control flow – synchronous and asynchronous. In the synchronous model, the UNO state requests information from the controller through function calls, blocking until the controller responds. These function calls allow for a streamlined, linear control flow that is very simple to debug. The issue is, that these function calls are blocking, waiting for user input. So, while waiting for a player to play a card, the system is unable to listen to state correction requests or other asynchronous actions from different sources. On the other hand, an asynchronous model has a more complicated control flow, as each node acts independently and concurrently. Asynchronous code is prone to complicated bugs and produces a control flow that is difficult to reason about. Despite these challenges, we chose an asynchronous model for our system, to be able to handle any action at any time.

5.3.4 Website

The main discussion for the website surrounded which web architecture to use. We considered Django and Flask since they are both common Python backend frameworks. We decided on Flask since it is lighter-weight.

For communicating the state between devices, we considered using websockets and Asynchronous Javascript and XML (AJAX). We ended up using websockets due to its low latency and overhead.

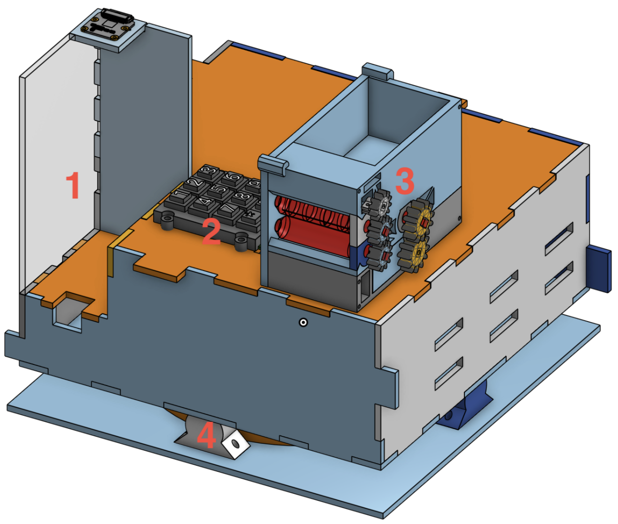


Figure 2: Entire chassis design in CAD
Label 1: Card discard pile
Label 2: Keypad for user input
Label 3: Card dealer
Label 4: Chassis rotation system

6 SYSTEM IMPLEMENTATION

6.1 Hardware

6.1.1 Full body Integration

The outer chassis was built using laser-cut 1/8-inch pressed boards purchased from Techspark with gears and the card dealer being 3D printed. The bottom plate's dimension is 26cm x 26cm while the upper rotating chassis is 25cm x 25cm x 10cm, length x width x height, with additional height on the discard pile, label 1 in Fig.2. The discard pile has a camera on the top to take a picture of the card that was just played, and the extra height makes sure there is enough distance between a full stack and the camera for it to focus. The small cutout on the front makes it easier to pull the cards out. Also, the left transparent wall is acrylic so that the person on the left of the current player can see the card in play easily. Label 2 is a number pad where the users will input different information: reset the game, the color of the card after a wild card, draw card, call bluff, and others. Hidden behind the keypad, we have a small red LED that lights up whenever an illegal move is detected. This is directly connected to the GPIO pin of the RPi. Label 3 is our card dealer that ejects cards towards the player automatically. Finally, label 4 is part of our chassis rotation system using an internal gear and a spur gear connected to a stepper motor. Labels 3 and 4 are explained in more detail in the next sections.

6.1.2 Chassis Rotation System

As mentioned in section 5.1.1, we are using 4 caster wheels to support the weight of the chassis, and a spur gear rotating within an internal gear to rotate the chassis.

The shapes in Fig. 3 labeled as 1 are the wheels, while label 2 shows the spur gear connected to the stepper motor and the internal gear.

6.1.3 Card Dealer

To make it easy to keep track of the cards being dealt, we designed the card dealer so that it deals from the bottom of the deck. Fig. 4's label 1 is the cutout for the corner of the card. A camera from the bottom takes a picture of each card before it gets dealt to the player. Label 2 is the roller with rubber bands that extrudes the card towards the exit of the dealer so that the rollers in label 3 can catch the card and move the card out at a faster speed. Label 2 rollers are geared with a continuous servo for more precision and force. Label 3 rollers are geared with a DC motor for higher speed. The approximate connections between the gears are shown in section 6.1.1's label 3, which shows the other side of the dealer design. Throughout many trials, we have changed the shape and size of the card hole. We settled with a card hole that is slightly smaller than the thickness of two cards and has a slight curve/slope as you can see near label 3's left side. Also, the front portion of the dealer can now be removed from the rest of the body. This helped us go through iterations faster and more efficiently with less filament and time used for 3D printing each card hole design. Finally, we have added an IR sensor that attaches right above the label 3 rollers and is housed in label 4. It provides feedback on whether the card has been successfully dealt or not, eliminating the need for human confirmation.

6.2 Motor Control

All three motors will be controlled using the Arduino Uno. We are using the Adafruit Motor Shield V2 to handle all of the extra circuitry and low-level control needed for the motors. The Motor Shield attached on top of the Arduino and is powered off of the 12V 3000mAh battery. Adafruit also provides a library for controlling the motors which are attached to the shield. The Arduino receives commands from the Raspberry Pi over UART. Latency and bandwidth are not huge considerations as the commands are only a couple of bytes. Then, depending on if the command is to deal a card or rotate the chassis, the Uno will enable the appropriate motors and send a done command back to the Pi. We offload motor control over to the Arduino because it is much better at supporting real-time IO control than the Raspberry Pi. The Arduino itself also draws very little power, around 50 mA.

6.3 Card Classification System

One of the key functions of the device is the ability to seamlessly classify cards from an image for rule enforcing, game state validation, and execute the correct control flow for the played card.

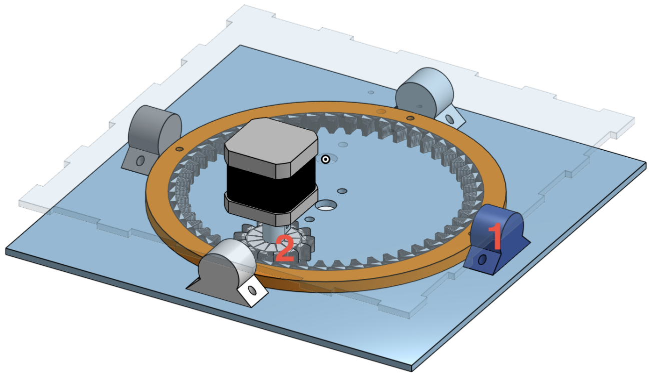


Figure 3: Chassis rotation system in CAD
Label 1: Caster wheels to support the weight and rotation
Label 2: Spur gear that rotates

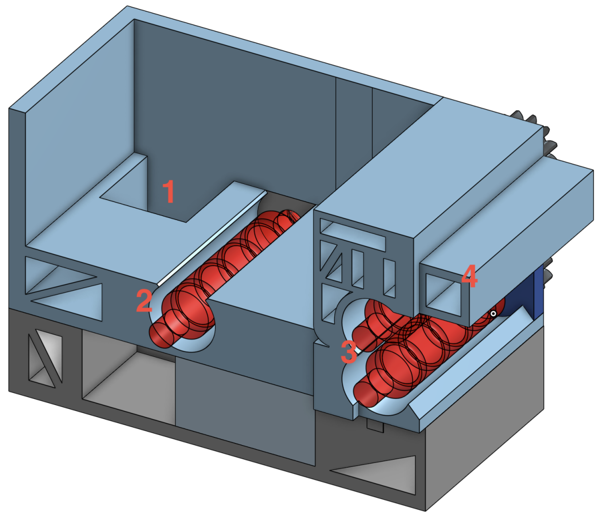


Figure 4: Card Dealer Design in CAD
Label 1: Hole for the camera to capture the card's corner
Label 2: Roller that extrudes the card forward
Label 3: Rollers that deal the cards to players
Label 4: IR sensor housing

Pictured in Fig. 5 is the ResNet model architecture for image classification. Our device employs a fine-tuned version of the ResNet18 model. The ResNet18 model has been pretrained to output a 1024-length vector representative of the semantic meaning of its input image. For our task, there are 15 output classes (0, 1, 2, ..., skip, reverse, ...), instead of 1024. So, we added two linear layers onto the head of ResNet18, reducing the model output to 15-length one-hot encoded vector. This new architecture was then fine-tuned on our own generated data.

We created two separate datasets. One of the datasets was for the camera facing the discard pile, and the other dataset was for the camera facing the bottom of the draw pile. Since these cameras see different lighting and angle conditions, it proved most beneficial to sample images for these cameras separately. For each camera, we took

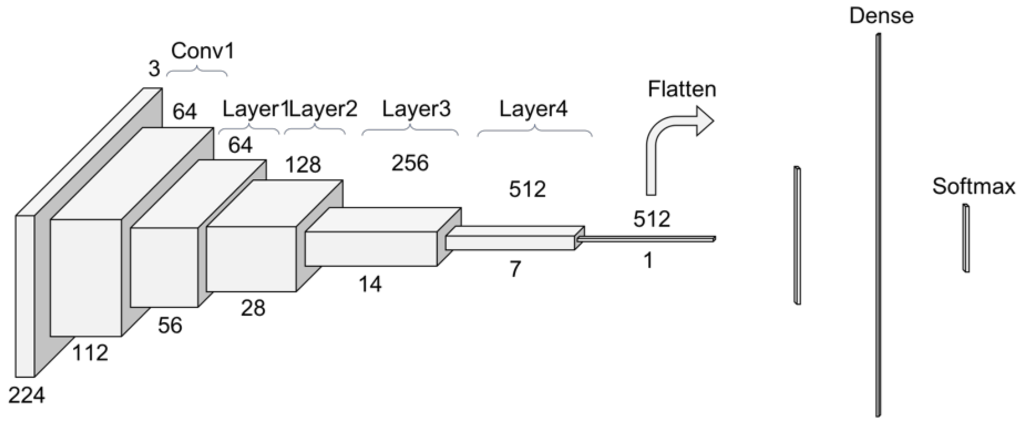


Figure 5: ResNet Architecture for Image Classification

around 200 photos, hand labeling both the color and type of the pictured card. After hand-classifying these images, we synthetically created 10 images per card by repeatedly sampling noise to vary angle, zoom, saturation, brightness, contrast, and more. When first training the model, we initially split into test, train, and validation sets *after* creating more synthetic data. Since these derived images were closely related to their real images, it resulted in the test and validation accuracy being misleadingly high. It was significantly more meaningful and effective to split the real images into test, train, and validation sets *before* synthesizing new data.

Our final design ended with two card classification models – one fine-tuned on real and synthetic data from the top camera and one fine-tuned on real and synthetic data from the bottom camera. Initially, we trained a single model on the datasets for the top and bottom cameras. Although this model performed very well on test and validation sets, it performed slightly worse in the field than the separated models.

After the card type is identified, the color may also need to be extracted, specifically for every card other than the Wild and +4 card. Initially, we used color filters hand-tuned to finding yellow, blue, green, and red in images. Although applying these filters led to incredibly quick color classification, it was very difficult to find the correct threshold values for blue and green. This basic algorithm had difficulty differentiating between blue and green, especially when lighting conditions changed.

To fix this, we went with a machine learning approach. Although this approach makes color classification much slower, it allows color classification to be better tuned to the varying lighting conditions that our system must withstand. This color classification model was trained on both the top and bottom camera datasets, again using a pre-trained ResNet18 as its base.

6.4 Software Model

In our design report, we laid out a mainly sequential execution with blocking IO. However, this did not scale to the full feature set we wanted to implement. We wanted to be able to wait on multiple IO sources at once and also at arbitrary times. Thus, it made sense to rework the architecture of our software to an asynchronous execution model pinned around async queues which sent messages between the three main software subsystems; the State, Controller, and Displayer.

6.4.1 State

The State handles all of the UNO game logic. It receives actions and updates from both the Controller and Displayer and applies the appropriate transformations on the players' hands. It will then emit new actions for the Controller and Displayer to do. For example, if the card played was illegal, the State would emit a new event to retry the turn as well as signal that the card was illegal. In another case, if a skip card was played, the State would emit two events to advance the machine to the next player instead of the usual one.

6.4.2 Controller

The Controller handles all of the interfacing with IO. It receives actions from the State such as GetUserInput or DealCard which it translates into the appropriate hardware actions, whether it be polling certain buttons on the keypad or sending motor commands over UART to the Arduino. It then responds with the user input or card classification back to the State.

6.4.3 Displayer

The Displayers acts as the interface between the UNO process and the web server. It sends state updates over web socket and receives Reset and CorrectState requests from the web server. It will then forward these requests to the State for processing.

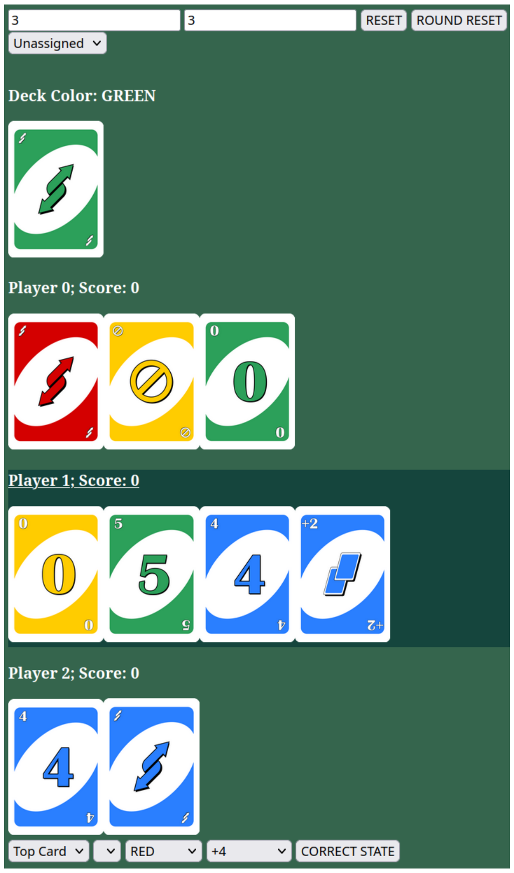


Figure 6: Example Spectator View

6.5 Website

With a competitive UNO tournament happening recently, it is very important to have an interface for spectating the game, validating the game state, and making any necessary corrections from failed classification or dealing.

The website is a separate process and can be hosted on either the Raspberry Pi or a third party machine like a laptop. The website uses Python's Flask as its backend. The Controller and clients communicate changes to the state with the web server through web sockets.

The website frontend shows the current cards each person has with different roles to show only the cards for the player you choose. The user is also able to correct state errors by replacing an incorrect card with the correct card. A snapshot of the final website can be seen in Figure 6.

7 TEST & VALIDATION

A compilation of all of our test results can be found in Figure 7.

7.1 Tests for Size

This validates the size use case and design requirements. We measured the width and height of the base of the platform using a ruler. It is 26 cm x 26 cm which is less than our target of 1 ft x 1 ft. The height of the entire system is also within 1 ft at the highest point which is where the top camera sits.

7.2 Tests for Rotation

This helps validate the automatic gameplay use case requirement. It also validates all of the rotation design requirements. Here we tested how quickly and accurately the platform can rotate from one player to the next player. By passing these tests we also implicitly verified the requirement for weight as we rotated the platform successfully. To do this, we first marked the 90° intervals representing a 4-player game. We also marked ±10° areas around each player as the valid zone. Then we ran many rotations and timed them with a stopwatch. Each rotation took about 1.90 s - 2.00 s, which is within our initial requirement of 3 seconds. Our initial methodology of releasing the stepper motor caused the motor to jerk and cause a huge error. Letting the stepper motor hold its position results in higher power usage and more heat but gains accuracy in exchange. We decided to go with holding the motors since the battery life with the motors held is still significantly above our requirement. In terms of accuracy results, when the chassis moves in the same direction as before, it has less than 5° error, within our 10° target. However, when the direction is reversed, the stepper motor stalls and skips steps, resulting in up to 15° error, slightly higher than our design requirement. However, since it is fairly easy to correct this during the game by lifting up the chassis, we still validate the automatic gameplay use case requirement.

7.3 Tests for Battery Life

This validates the longevity use case requirement and the battery life design requirement. To test this we played normal UNO games on loop while recording the time. We played for 5 battery cycles where a battery cycle involves draining the battery from 100% to 0%. We then recorded the average battery life which was 93 minutes, 50% longer than our 1 hour target.

7.4 Tests for Card Classification

This test helps validate the automatic gameplay use case requirement. This will also validate the card classification design requirement. We were interested in measuring both the accuracy and latency of card classification. After collecting our own data and synthesizing new data, we ended with around ten thousand images in our dataset for the bottom camera and twenty thousand images for the top camera. After splitting these sets into train, test, and validation datasets, using the 80%, 10%, 10% rule, we observed 100% accuracy on testing and validation sets for both models after 2 epochs. In the field, we experienced less than 1 incorrect classification per 500 images. This result far exceeds our requirement of $> 95\%$ accuracy. As for color, the naive CV implementation saw a 93% accuracy, whereas the ML model reached 100% accuracy.

Our final classification time for both type and color ended up being around 300 ms. This time would have been cut in half if we chose the naive CV implementation, but we would have lost significant accuracy. We have easily cleared our requirement of 1000 ms allocated for classification.

7.5 Tests for Card Dealing

This helps validate the automatic gameplay use case requirement. This also validates the card dealing design requirement. In this test, we test both the latency and accuracy of dealing a card. To do so, we continuously dealt 1000 cards. We timed the total time to deal the cards and calculated the average time to deal one card. We measured a dealing latency of 1.5 seconds per card which is higher than our design requirement of 1.0 seconds per card. This is because the motors need more time to ensure that the card is fully dealt every time. If we spin the motors for a shorter period of time, the amount of dealing errors increases drastically. However, this is not too big of a deal as aside from the dealing phase, card dealing represents a small portion of total game time and is masked by the latency of other actions such as machine rotation. Thus, we still validate the automatic gameplay use case requirement. We also counted the number of errors that occurred during dealing. Our accuracy was 97.5% which is higher than our 95% target.

7.6 Website Latency

This helps validate the real-time website update use case and design requirement. For this test, we measured the round trip latency of updating the website and receiving a response from the website confirming the update. This time was logged on the Raspberry Pi. To measure the one way latency, we divided the round trip latency by two. Using the logs, we found the website latency to be 200-250 ms, which is far less than our 2000 ms requirement.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Fig. 8 on page 13. Most of our initial tasks were parallelizable. While integration was mostly dedicated towards the end of the semester and individual tasks, Thomas and David were able to do gradual integration between the hardware and embedded software on the way, reducing the amount of time used for full system integration. The software driving the system took longer than expected to finalize. While we thought we had our final control flow after around week three, we decided to make the switch to async much later on. This caused delays in starting the website, which bled into our final week. Although we did finish and test the website, it would have been better to have more slack time in the final parts of our schedule.

8.2 Team Member Responsibilities

David worked on the embedded code and worked with Jason on implementing the software on the Raspberry Pi as well as the website. Thomas managed all the hardware components. This includes developing CAD models for all hardware components, laser cutting the outer chassis, and prototyping different designs for the card dealer. Also, he assembled the final chassis. Jason drove the software design and implementation for the project. This task includes designing and implementing the control flow, providing controller interfaces for David to implement, collecting data for classification, and training a home-grown UNO card classifier.

8.3 Bill of Materials and Budget

The BOM is shown in Table 1 on page 12. Many parts we have purchased only come in larger quantities than what we need. Thus, the budget to build our project is realistically lower than what is listed. We have purchased additional batteries in case they die during demo day as well.

8.4 Risk Management

8.4.1 Hardware

The most significant risk in hardware was getting the dealer to only deal one card at a time more than 95% of the time, and the rotation of the chassis being accurate throughout a whole game of UNO. For the dealer, since we are dealing from the bottom, we had to make sure the bottom roller has enough friction to grab a card but also make sure the card above it is not caught together. Based on some experimentation, we figured out that the amount of pressure from the top and the opening size for the card to leave were the most important parts. We purchased 3D print filament to use with our 3D printer and improved the accuracy of the card dealer over many design iterations until we finally got a design that achieved our target accuracy. We also changed the design of the dealer to make it quicker to iterate on, since we only had to replace a small part of the dealer. We also added an IR sensor to detect when a card is not dealt successfully. This means that even if an error occurs, we do not corrupt the state of the machine, and we can have a human intervene to fix the dealer safely.

To mitigate the risk around chassis rotation error, we lock the stepper motor even when it is not in use. This improves our rotation accuracy at the expense of power as we draw the full stall current. It is also relatively easy to fix any deviations in the rotation as we can manually turn the machine to face the correct player.

8.4.2 Software

One of the biggest sources of risk in the software subsystem was the card classification algorithm. A low accuracy may have actually downgraded the UNO experience, rather than upgrade it. Throughout the dataset collection and training process, we maintained a medium-sized list of potential ideas for improving accuracy. One of these ideas was adding more preprocessing to the pipeline. As it stands, the images are sent directly into the classification model(s) with no preprocessing. If there had been difficulties with achieving high enough accuracies, we were already exploring contour algorithms to cut the card symbols out of the image for much more straight-forward classification. Additionally, we had plans to upgrade our model to a transformer-based model, which have recently proven very effective in image classification contexts. Continually generating new ideas for worst-case scenarios while developing our main algorithm helped to mitigate the risk of any failures or underperformances of our primary ideas.

Another source of risk was the switch to an asynchronous model. To mitigate risk, we kept this change on a separate branch from the main branch, where our functional synchronous model lived. The main point of the asynchronous model was to be able to process request at any time, even while waiting for user input. If the asynchronous approach had failed, we had backup plans to support asynchronous state corrections at the end of each turn. This would have slightly harmed the experience, but it would have been functional. Having a mostly-implemented backup system on a separate branch made it much easier and safer to take risks with a new software model.

9 ETHICAL ISSUES

If our product were to be widely adopted in use for UNO tournaments, there are a few security concerns associated with the device. Firstly, if someone were able to hijack the system, they could intercept the score calculation, and a round winner would get fewer points than they deserved. This would require that someone manually count the cards to double-check validity, but of course, this defeats the purpose of counting the cards in the first place. Also, a person hijacking the communication between the game state and the website could trick the system into thinking they are a moderator. Moderators are able to see all of the opponents' cards, instead of just their own. They could then, in theory, pass this extra information along to someone they are helping in the tournament. Most of the UNO tournaments have stakes involved, making this a larger issue. To add to the security concerns, an adversary could hijack the cameras of machines in households. While the cameras are not placed to give out a lot of information except the cards, there still exists a concern about privacy.

Another possible issue is the reliance on machine learning. While we have mitigation methodologies and ways to fix classification failures, this could still cause disagreements among players. Also, it not only decreases players' trust in the machine to accurately track the game, but it could also lead to confusion on whether someone has cheated or the machine has made a mistake. These collectively could reduce the quality of the experience from a family game night all the way to a tournament with high stakes.

10 RELATED WORK

Previously in capstone, team *PokerCam* created a full poker game tracker. They built a custom cardshoe that uses a camera and machine learning to identify everyone's hand. We have realized that UNO, which is one of the most popular card games with a competitive scene, doesn't have a similar system. Also, there are many automatic card dealers that deal cards for people, but in UNO, the direction of play constantly changes, making it hard to use traditional automatic card dealers. Merging these two ideas, we wanted to build a machine that has an UNO game tracker with automatic card dealing. This will assist dealers in tournaments and promote a more enjoyable experience when playing casually. Watching videos online about automatic card dealing and shuffling provided us a general idea of how we could deal cards from the bottom[5] and rotate the chassis[6].

11 SUMMARY

Overall, we successfully created a novel, engaging, and innovative way to play the game of UNO. From casual family gatherings to intense tournament settings, UNOmatic provides seamless control-flow handling for all levels of players. Our system exceeded the majority of our design requirements. This includes an incredibly accurate card classifier, a reliable card dealer, an engaging multi-purpose website, and perfect control-flow and state management.

Some places for improvement were the device rotation accuracy and the card dealing latency. Although these did not quite meet the design requirements, they still are within our user requirements. The rotation accuracy problem seems to stem from the low motor quality and motor wear over time. This can be fixed by implementing sensors that provide a feedback mechanism to ensure the rotation is accurate. Also, while the dealing latency is still higher than our proposed requirement, it doesn't impact the user experience negatively since the dealing phase is a short, one-time cost at the beginning of the game. This is a harder problem to solve since it involves improving the dealer design or finding the right material with higher friction that requires less time to grab and extrude the card.

11.1 Future work

If we were to add features to the hardware side, we could add another camera/sensor that would face the players. This would then be used in conjunction with machine learning to find the next player as the machine rotates. Thus, the players could sit in any configuration instead of being forced to sit in a circle. We could also add a microphone and speech processing model to listen for UNO instead of having players press buttons. This would mimic normal UNO game play better.

As an extension to the software portion of the project, it would be nice to implement some kind of validation for the moderator view on the website. Currently, any viewer can switch to the moderator view, which exposes all players' cards. In the future, it would be necessary for tournament settings to include some kind of password-protected authentication to enter moderator mode.

11.2 Lessons Learned

One of the most important things we learned was the importance of design iteration. On both the hardware and software side, we went through many versions and designs before we settled on our final product. 3D printing proved invaluable in quickly producing many different card dealers with slightly different characteristics. On the software side, we did an almost full system rewrite to accommodate the asynchronous execution model. However, we were able to reuse a lot of code from the previous iteration which helped a lot in migrating quickly. The modular design of our software from the beginning made it easier to do even a dramatic architecture change like the one we did. Another lesson we learned is the difficulty of making trade-offs. Several times we came to a crossroads which forced us to sacrifice in one area to meet the needs in another. Moving to the async execution model was a large engineering effort cost, but allowed us to implement many of the features that ended up in our final design. Also, for chassis rotation, we had to sacrifice battery life in order to gain rotation accuracy, which was an important decision to make.

Glossary of Acronyms

- mAh - milli Amp hours
- RPI – Raspberry Pi
- Wh - Watt hours
- IO - Input Output

References

[1] Wikipedia. *Uno (card game).* 2022. URL: `https://en.wikipedia.org/wiki/Uno_(card_game)`.

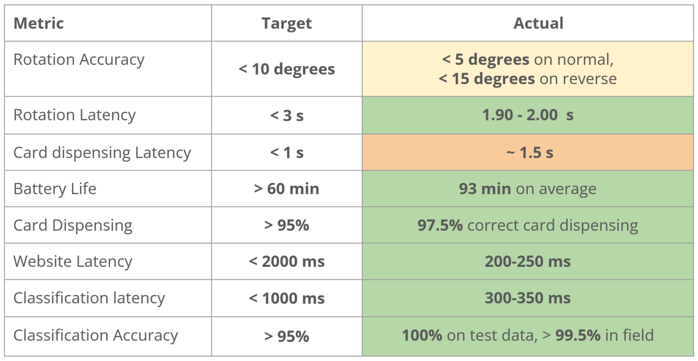
[2] VeeFriends. *Announcing the Official UNO™ Tournament at VeeCon 2023.* 2023. URL: `https://blog.veefriends.com/announcing-the-official-uno-%EF%B8%8F-tournament-at-veecon-2023-bcc19ec9e6db`.

[3] Bulldogcastors. *Castor wheels Roll Resistance.* https://www.bulldogcastors.co.uk/blog/castor-wheels-roll-resistance/. 2/29/2024. 2016.

[4] Dave Lippert and Jeff Spektor. *Calculating proper rolling resistance: A safer move for material handling.* https://www.plantengineering.com/articles/calculating-proper-rolling-resistance-a-safer-move-for-material-handling/.

[5] 3DprintedLife. *Rigged Card Sorting Machine - ALWAYS Get The Hand You Want!* 2020. URL: `https://www.youtube.com/watch?v=eMTXyl7tPEk&ab_channel=3DprintedLife`.

[6] Mr Innovative. *Diy Arduino based card dealing machine.* 2021. URL: `https://www.youtube.com/watch?v=dx9-wwSQbUE&t=176s&ab_channel=MrInnovative`.

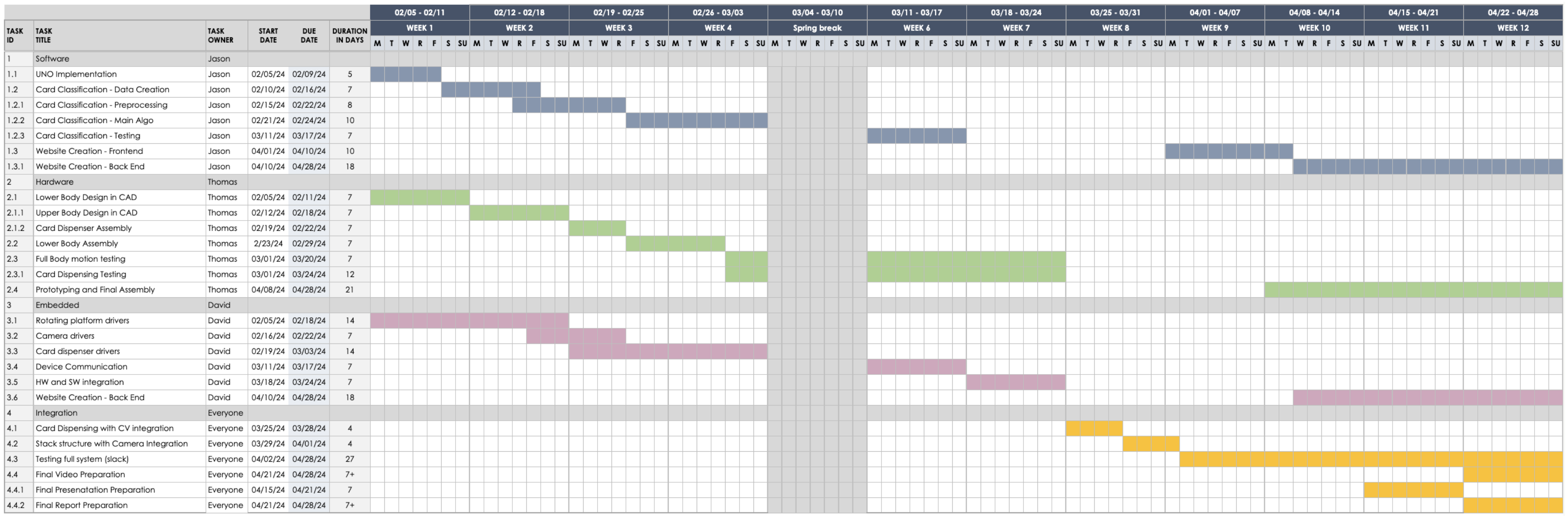


Metric	Target	Actual
Rotation Accuracy	< 10 degrees	**< 5 degrees** on normal, **< 15 degrees** on reverse
Rotation Latency	< 3 s	**1.90 - 2.00 s**
Card dispensing Latency	< 1 s	**~ 1.5 s**
Battery Life	> 60 min	**93 min** on average
Card Dispensing	> 95%	**97.5%** correct card dispensing
Website Latency	< 2000 ms	**200-250 ms**
Classification latency	< 1000 ms	**300-350 ms**
Classification Accuracy	> 95%	**100%** on test data, > **99.5%** in field

Figure 7: Compiled Tests Results

Table 1: Bill of materials

Description	Manufacturer	Quantity	Cost	Total
Small breadboards	Amazon	1	$10.00	$10.00
Stepper Motor 1	Amazon	1	$9.00	$9.00
Stepper Motor 1 (replacement)	Amazon	1	$9.00	$9.00
Stepper Motor 2	Amazon	1	$18.00	$18.00
Stepper Motor 3	Amazon	1	$13.00	$13.00
Continuous Servo	Amazon	1	$25.00	$25.00
Voltage Converter	Amazon	1	$20.00	$20.00
9V Rechargeable Batteries	Amazon	1	$23.00	$23.00
RPI screw terminal block	Amazon	1	$20.00	$20.00
Battery 12v 3000mAh	Amazon	2	$26.00	$52.00
Battery 12v 3000mAh (extra)	Amazon	2	$26.00	$52.00
9V Battery to DC adapter	Amazon	1	$6.00	$6.00
12V DC Motor	Amazon	1	$9.00	$9.00
3D Printer Filament	Amazon	1	$19.00	$19.00
Rubber bands	Amazon	1	$7.50	$7.50
Wheels	Amazon	1	$8.00	$8.00
Screws	Amazon	1	$17.00	$17.00
18 AWG Wire	Amazon	1	$12.00	$12.00
Bearing	Amazon	1	$9.00	$9.00
Axle	Amazon	1	$6.50	$6.50
White LED backlight 1	Amazon	1	$7.00	$7.00
White LED backlight 2	Amazon	1	$7.50	$7.50
Adafruit Motor Shield	Digikey	2	$20.00	$40.00
RPI wide angle camera	Digikey	1	$35.00	$35.00
Wooden boards	Tech Spark	6	$3.00	$18.00
Camera adapter - 200mm	Digikey	2	$1.00	$2.00
Camera adapter - 300mm	Digikey	2	$2.00	$4.00
Raspberry Pi 5	Inventory	1	$0.00	$0.00
RPI wide angle camera	Inventory	1	$0.00	$0.00
Arduino Uno	Thomas Kang	1	$0.00	$0.00
Grand Total				$458.50



TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION IN DAYS
1	Software	Jason			
1.1	UNO Implementation	Jason	02/05/24	02/09/24	5
1.2	Card Classification - Data Creation	Jason	02/10/24	02/16/24	7
1.2.1	Card Classification - Preprocessing	Jason	02/15/24	02/22/24	8
1.2.2	Card Classification - Main Algo	Jason	02/21/24	02/24/24	10
1.2.3	Card Classification - Testing	Jason	03/11/24	03/17/24	7
1.3	Website Creation - Frontend	Jason	04/01/24	04/10/24	10
1.3.1	Website Creation - Back End	Jason	04/10/24	04/28/24	18
2	Hardware	Thomas			
2.1	Lower Body Design in CAD	Thomas	02/05/24	02/11/24	7
2.1.1	Upper Body Design in CAD	Thomas	02/12/24	02/18/24	7
2.1.2	Card Dispenser Assembly	Thomas	02/19/24	02/22/24	7
2.2	Lower Body Assembly	Thomas	2/23/24	02/29/24	7
2.3	Full Body motion testing	Thomas	03/01/24	03/20/24	7
2.3.1	Card Dispensing Testing	Thomas	03/01/24	03/24/24	12
2.4	Prototyping and Final Assembly	Thomas	04/08/24	04/28/24	21
3	Embedded	David			
3.1	Rotating platform drivers	David	02/05/24	02/18/24	14
3.2	Camera drivers	David	02/16/24	02/22/24	7
3.3	Card dispenser drivers	David	02/19/24	03/03/24	14
3.4	Device Communication	David	03/11/24	03/17/24	7
3.5	HW and SW integration	David	03/18/24	03/24/24	7
3.6	Website Creation - Back End	David	04/10/24	04/28/24	18
4	Integration	Everyone			
4.1	Card Dispensing with CV integration	Everyone	03/25/24	03/28/24	4
4.2	Stack structure with Camera Integration	Everyone	03/29/24	04/01/24	4
4.3	Testing full system (slack)	Everyone	04/02/24	04/28/24	27
4.4	Final Video Preparation	Everyone	04/21/24	04/28/24	7+
4.4.1	Final Presenatation Preparation	Everyone	04/15/24	04/21/24	7
4.4.2	Final Report Preparation	Everyone	04/21/24	04/28/24	7+

Figure 8: Gantt Chart