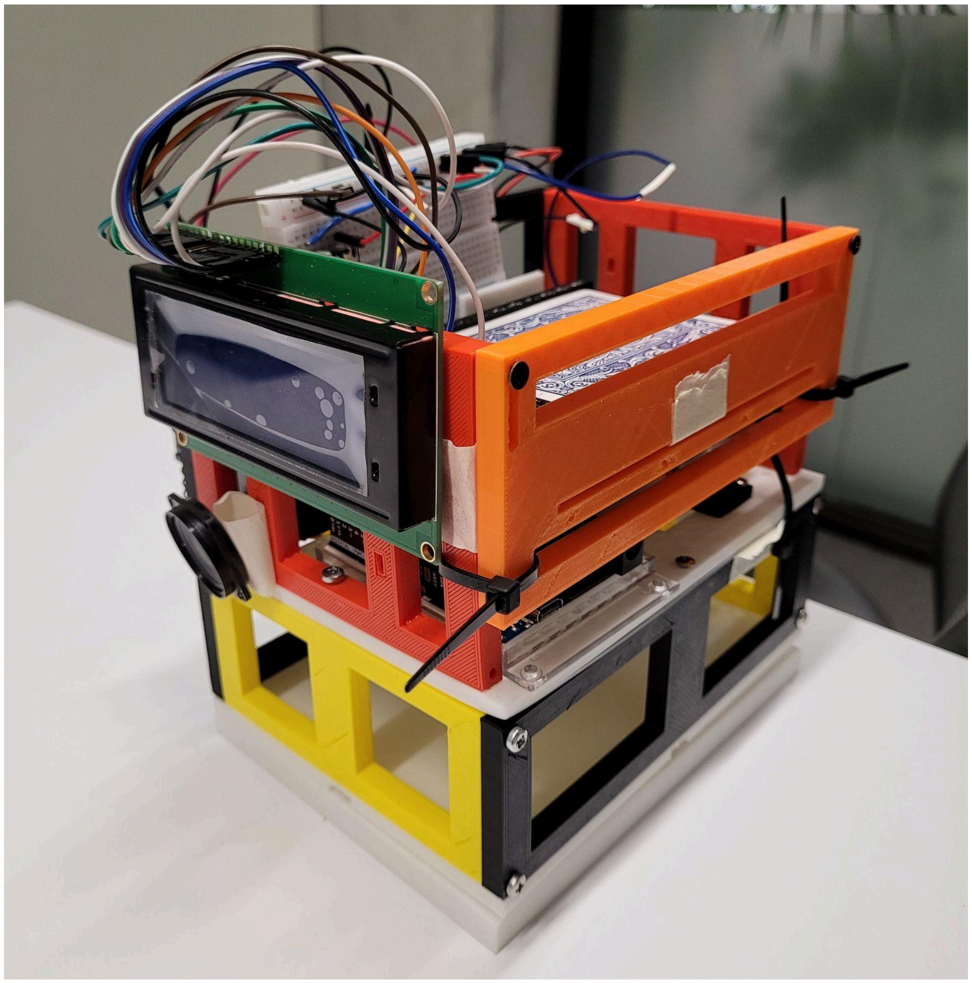
Final Report

MIE438: Microcontrollers and Embedded Microprocessors
Due: April 12, 2024

Team #24	
Team Member	**Student Number**
Henry (Hua Hao) Qi	1005758039
Harry Park	1005674405
Humyra Taifoor	1006080526
Yi Lian	1005709333
Demo Video Link: https://youtu.be/uCzBhp2jN_U	


K.I.R.B.Y - Kards in Real-life Brought to You

1.0 Summary of Project Proposal

To help eliminate the tedious and repetitive process of dealing in card games, an automatic card shuffler and dealer is proposed. The process of dealing out cards can significantly lower the quality of experience of players, in addition to the risk of unfairness during the shuffling and dealing process. To reduce downtime between rounds and ensure fairness, an automatic shuffler and dealer can speed up the process and keep players engaged.

1.1 Initial Plans

The initial goal for this problem is to shuffle and deal cards evenly among players once the user has split the deck into half and inputted the number of players and cards that need to be dealt. Once this core problem has been solved, further modifications can be made by placing the deck into the design without splitting it in half, adding the option to select to deal the cards out evenly or a specific amount, selecting the number of players, and being able to automatically shuffle and deal based on pre-set setting based from a pool of default games.

The preliminary design consisted of using 4 DC motors and 1 servo motor to both shuffle and deal the cards, where 2 DC motors are attached to one roller each that feeds two piles of cards into the centre, 1 DC motor lifts and lowers the platform holding the cards using a lead screw mechanism, a servo motor and a gear system to rotate the base, and a DC motor to feed out the cards. All the motors are connected to an Arduino Uno R4 Wifi, which is connected to a liquid crystal display (LCD) to display the stages of the shuffling and dealing process. C++ was decided to be used as the main language. For the first component of the design, the microcontroller will be programmed to shuffle the cards by varying the speed and timing of the rollers. For the second component of the design, the program moves the platform with cards by controlling the motors and outputs cards, after doing arithmetic calculations depending on the number of players inputted by the user.

1.2 Planned and Unplanned Changes

After receiving initial feedback for the proposal, the team planned to reduce the scope of the project from shuffling and dealing to solely dealing. With this change, only 2 DC motors were required. To add complexity, a planned change was to implement a speaker and LCD to play background music and show the dealing status as the cards were dealt. The team also chose to use Wifi as the communication network between the user interface and the microcontroller.

As the project progressed, the team decided to use a static platform to deal the cards out instead of an adjustable one. This eliminated the need for the servo motor to adjust the platform. The team also removed the additional speed control aspect as it added unnecessary complexity, so only one DC motor required encoders out of the two. As well, the base no longer rotates with a gear system and servo motor, but rather with a DC motor with an encoder to create a more simple mechanical structure. This helped the team save on Arduino pin capacity and weight. As such, the final design only uses 2 DC motors (one with an encoder and one without), one LCD, one speaker, and one Arduino Uno R4 Wifi.

1.3 Challenges and Solutions

Some challenges experienced by the team include wire management, LCD integration, mechanical issues in rotating and dealing cards, and calibration issues. Each component in the design utilizes many wires, especially the DC motor with an encoder and the LCD. Since the design includes a rotating body where only the upper part of the design moves, careful consideration had to be made in organizing the components to prevent the entanglement of wires. A solution for this was to reorganize the wires, use electrical tape and zip ties when necessary to ensure tight connections and use shorter wires where possible. Additionally, one major change to prevent wire entanglement was the base motor placement. Originally, the base motor was placed in the bottom stationary part of the design but would be wired to the microcontroller that rests on the top rotating platform. Since the rotation motion only requires movement of the shaft, this added complexity to wire placement as the wires would have to rotate with the platform while the motor remains stationary. Flipping the base motor and using a housing for the motor to rotate the top platform instead ensured that all wired components rotated together.

Another challenge was that the LCD could not display any letters due to its constant brightness level. The solution for this was to implement a potentiometer to control the brightness of the LCD. Furthermore, the design faced mechanical issues in dealing cards, as the motor would rotate the roller but the smooth finishing on the cards would prevent them from being outputted. To add friction between the motor and the cards, rubber bands were used to wrap around the roller. Additionally, the top of the body was initially not able to rotate separately from the base. The solution to this was to reduce the friction between the base and the body by adding a gap between the upper and lower sections of the body. Lastly, calibrating the motors to ensure that the body rotates at an equal angle each time was quite difficult. This issue was resolved by calibrating the encoder counts per degree rotated after assembling all parts.

2.0 Final Design

The overall design is an embedded system as it contains a processing device, the Arduino Uno R4 with Wifi, elements to receive input, such as information received from the user interface, elements to implement output, such as the motors and the speakers, and bridging elements, such as the motor driver and the ESP32 module on the Arduino Uno R4 to communicate over Wifi [1].

The final design's main functionality is to deal out cards. Cards are placed on top of a platform, where they are dealt out to different players between rotations of the base. The dealing motion is controlled in real-time by a roller attached to a DC motor. While the cards are being dealt, the LCD displays the user configurations and dealing status for visibility. At the end of the dealing process, the Kirby theme song is played using the speakers as an indication. Thus, the final design mainly focuses on the singular task of dealing out cards according to user input, adhering to the general properties of embedded systems where it is a combination of hardware and software designed for a specific function.

To improve user interaction, the microcontroller can also be connected to a phone or computer using Wifi. This is done by hosting a web server accessible by the user, which displays a website where the user can input settings. These user specifications such as speed settings, the number of players, and the option to start/stop or execute an emergency stop are communicated to the Arduino. The Arduino fits into the project as it is the device that can take in the user input, control the motors and the speaker, and communicate the status to the LCD screen.

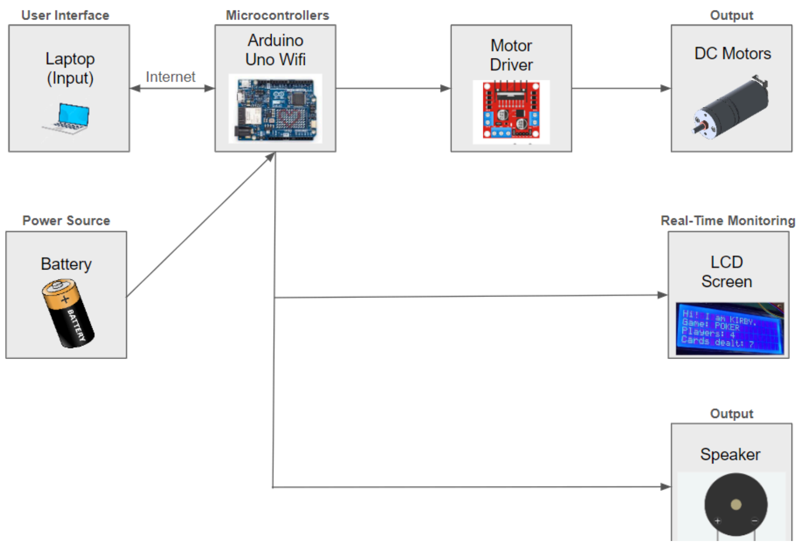
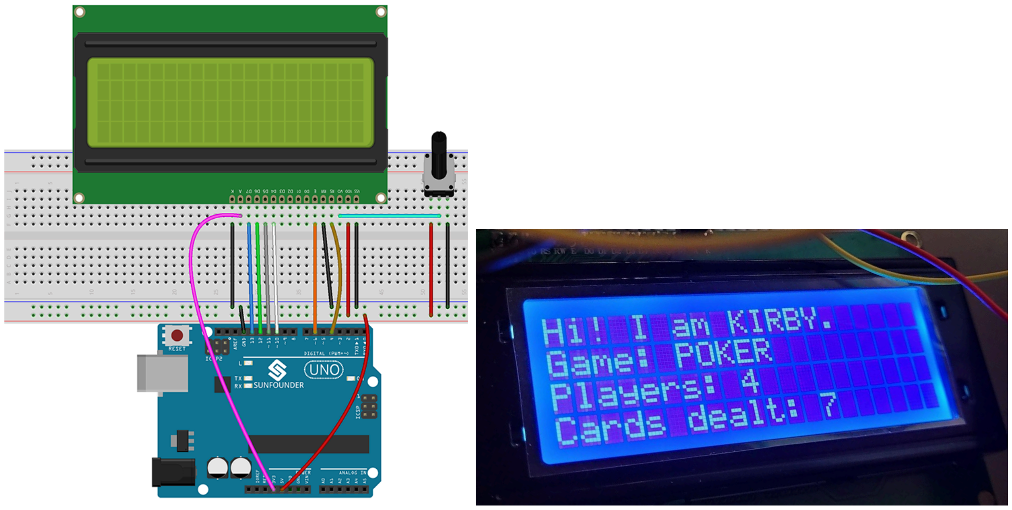


Figure 1: System architecture design.

2.1 Hardware

The chosen LCD model is the SunFounder LCD2004 with the capability of displaying 4 rows of 20 characters. It is powered directly from the battery through the breadboard and the LCD backlight from the 3.3V pin of the Arduino. A potentiometer is used to adjust the contrast ratio of the LCD screen by modifying the resistance to the power source. This LCD was chosen for its large display to allow KIRBY to communicate during its dealing phase. The display includes the current game being played, the number of players, and its progress in the current dealing phase. All information is obtained from user inputs through the microcontroller-hosted website.



```
Hi! I am KIRBY.
Game: POKER
Players: 4
Cards dealt: 7
```

Figure 2: LCD connection to the Arduino Uno R4 Wifi (Left) & LCD demo output (Right).

Two additional 6V DC motors are included, powered by a single L298 motor controller. One DC motor rotates KIRBY's base, using encoder values to keep track of its orientation to deal cards into equal piles for each player. The other DC motor turns on and off to deal out a single card at a time when prompted.

2.2 Software

To integrate the various functionalities of the card dealer, the team used a high-level C++ language to develop a finite state machine [2] that combines the functionalities into different stages of the card-dealing process. Since the webpage has to be called multiple times during the card dealing process, the finite states were maintained by hosting a private webpage with the *webServer()* function.

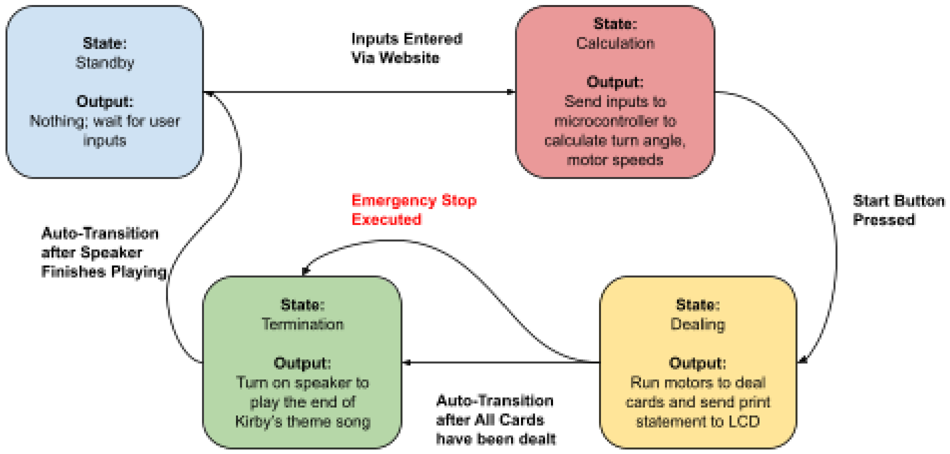

State: Standby
Output: Nothing; wait for user inputs
Inputs Entered Via Website
State: Calculation
Output: Send inputs to microcontroller to calculate turn angle, motor speeds
Emergency Stop Executed
Start Button Pressed
Auto-Transition after Speaker Finishes Playing
State: Termination
Output: Turn on speaker to play the end of Kirby's theme song
Auto-Transition after All Cards have been dealt
State: Dealing
Output: Run motors to deal cards and send print statement to LCD

Figure 3: The four states of KIRBY - Standby, Calculation, Dealing, and Termination.

2.2.1 - State 1: Standby

The four different states are toggled through the user's command via the buttons on the website, which are illustrated in Appendix A. At the Standby state, which is the default state the microcontroller boots up to, KIRBY waits for the user to input the type of game to be played, the number of players of the game, as well as the speed of the card being released.

2.2.2 - State 2: Calculation

Once the input has been received, the website transitions to the Calculation state, where the microcontroller pre-calculates the motor speed and turn angles to deal out cards to players appropriately. The microprocessor determines the angle of rotation by dividing 360° by the number of players, with a maximum of 15 players (24°-180°). The input for the number of cards is an integer value requiring high precision, the angle of rotation is a floating-point value with ±0.5° precision, and the amount of degrees per encoder value is a floating-point value as well. During the rotation of the base motor, the Arduino uses a floating-point division to determine the input encoder value needed for the motor. Using float over fixed-point typing and being aware of data representation limits ensured minimal errors during the rotation of the design.

2.2.3 - State 3: Dealing

Once the microcontroller receives and processes the user inputs, KIRBY is ready to deal out cards and is initiated by the user pressing the "START!" button on the webpage. Once the button has been pressed, the *dealCards()* and *sendLCD()* functions are called within the *webServer()* function to deal out cards to players and display the status on the LCD board.

2.2.4 - State 4: Termination

The Termination state can be reached either automatically when KIRBY has finished dealing out cards, or when the user executes an emergency stop command due to unforeseen circumstances. At this stage, the *webServer()* function calls the function *disableMotors()* to terminate any motors that are actuated, and clears the LCD panel. Then, the *musicEnd()* function is called to play Kirby's theme song as an indication of the Termination state. After playing the music, KIRBY automatically transitions to its Standby state.

2.2.5 - Optimization & Methods Used

Within the finite state machine, subroutines such as *dealCards()* and *sendLCD()* are called rather than inline functions or macros. This was beneficial in saving resources on the Arduino since those subroutines can be called multiple times while only being compiled once. Additionally, since the subroutines were lengthy, the effect of the call/return overhead can be neglected, resulting in minimum drawbacks in using subroutines over macros [3].

Since the team decided to use subroutines, it was crucial to specify the variable types before and after they are passed into functions. Although specifying types of global variables can potentially exhaust resources, it is a powerful method of preventing overflows, so the team decided to specify the type of all variables used globally. (See Appendix B)

Additionally, floating-point representation was used over fixed-point representation for arithmetic calculations since the numerical variables needed to be as accurate as possible. Some examples where the floating point numbers were used include the calculation of the rotation angle of the base and the calibration of the motor encoders. [4] (See Appendix C)

Global optimization techniques were also used to increase the efficiency of the system. One particular technique the team used was factoring out unchanging portions of the code from loops. For example, in the *musicEnd()* function, the delay between each note was not altered within the for loop, so it was taken out to be called before the for loop [5]. (See Appendix D)

Interrupts were used in various parts of the software to prevent the microcontroller from failing due to unforeseen circumstances. Whenever the user prompts to change KIRBY's state via the private webpage, it enters an interrupt state where it terminates its current outputs and reassesses its current state, preparing to execute actions according to its next state. This accommodates any delays that might occur when transitioning states on the webpage. KIRBY exits the interrupt state when there are bytes to read from the client, ensuring that the webpage has loaded completely, and is safe to execute actions [6].

Additionally, interrupt pins are used with the DC encoder motor to allow for external interrupt handling [6]. The interrupt pin was initialized to update the base motor's encoder count in the interrupt service routine (ISR). In this instance, as the motor encoder spins, the encoder outputs square waves on both its A and B pins. The ISR runs whenever it detects encoder A's square wave changes, transitioning from the low to high state or high to low state. (See Appendix E) Within the ISR, the base motor's encoder value updates in positive increments or negative increments depending on whether A and B are high or low, to keep track of how much the motor is spinning in each direction. (See Appendix F)

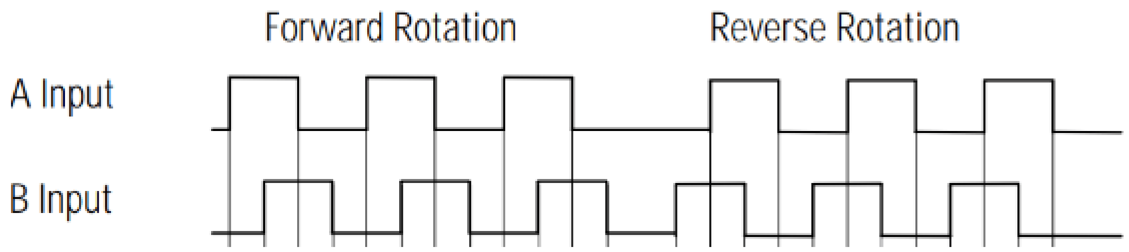


Figure 4: DC motor encoder A and encoder B states for rotation.

To ensure each player gets the right amount of cards, KIRBY is calibrated to turn a certain interval in terms of encoder values. During testing, it was found that the motor updates 200 in encoder value for 360 degrees of rotation. Keeping the base motor's encoder values updated is important in this application to ensure each player gets the right amount of cards in their pile. However, since the motor was tested without any load applied, after completing KIRBY's assembly, the ratio of encoder values per revolution of the motor varied from the testing phase.

2.3 Structural

KIRBY's structure stands from a foundation of 3D-printed parts, using PLA as the material of choice for its rigidity and strength. Each wall is fastened with 3mm bolts to create a sturdy cubic frame to hold all the hardware, including the DC motors, motor driver, breadboard, LCD, and the Arduino Uno R4 Wifi. Custom parts were made to attach to the hex-shaped motor adaptors, press-fitted to allow for base rotation and card dealing. These custom parts and the full frame body were all modelled using SolidWorks.

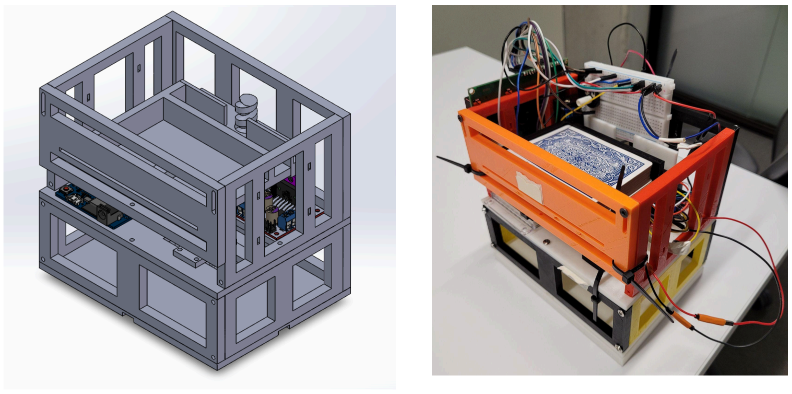


Figure 5: Full CAD of KIRBY (left) and completed assembly of KIRBY (right).

2.4 Testing and Calibration

Testing mostly consisted of ensuring the motors were moving accordingly. Firstly, the base motor's job is to ensure it rotates at equal increments to each player. Due to the additional frictional force with the added weight of KIRBY's body, the motor's turning speed is different from test results obtained with no load on top of the base motor. As a result, adjustments were made to the motor's PWM pin by modifying the input voltage, a range of 0-255, to ensure the motor has sufficient power to spin and not stall out. The base motor also accounts for some errors to compensate for overshooting as it rotates to deal cards, which were determined through trial and error with the full weight of KIRBY to ensure precision in turning.

To calibrate the motor that launches the card, rubber bands were attached to the motor adapter for additional grip strength and friction to deal the card out from the bottom of the pile. This DC motor only turns on and off when prompted, turning at a max PWM input of 255 for a brief time delay. After trial and error testing, this delay was found to work best at 300 to 350 milliseconds to give a short burst of the motor turning to deal out one card at a time. Additional weight was also added to sit on top of the deck of cards to provide more pressure to the card contacting the rubber-banded roller.

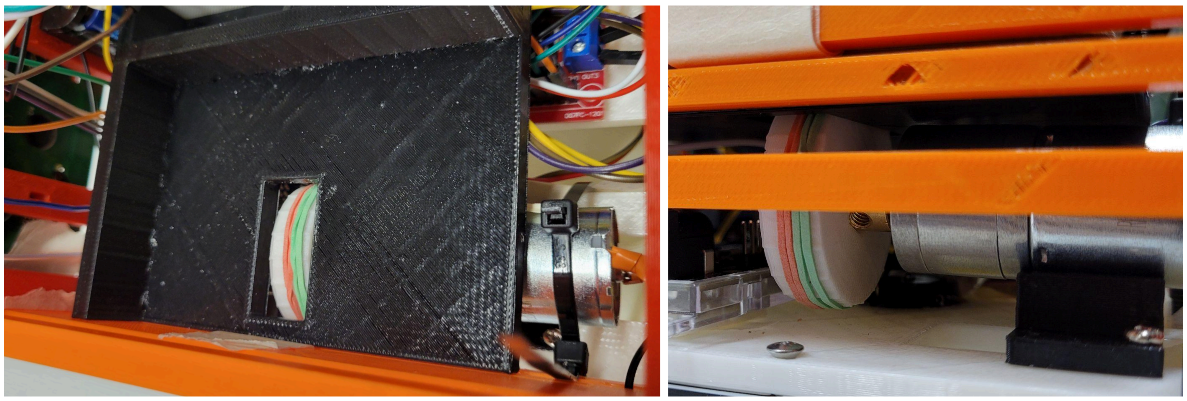


Figure 6: The card dealing motor uses rubber bands for additional gripping force.

Additionally, the website hosted via the ESP32 module on the Arduino board was tested to add an appropriate amount of delays when transitioning pages. When an emergency stop command is prompted via the website, the microcontroller has to quickly receive the message and execute stop commands to the motors, hence only 10 ms of delays were added. In other states of the *webServer()* function, 500 ms of delay was added after trial and error to allow the webpage to refresh properly.

2.5 Analysis of Choices

The Arduino Uno R4 was selected based on various design criteria such as the interfacing and the design process. In terms of interfacing, although only 4 GPIOs were required for the design (speaker, LCD, and 2 motors), Arduino was preferred since it has enough pins to connect with the motor encoders and LCD panel. Additionally, the selected Uno R4 model had Wifi capability, which was crucial for easily setting up the finite state machine for our design. The ability to connect to Wifi allowed the team to visualize the finite states via different web pages, which executed different actions according to their state. As well, the integrated Pulse Width Modulus (PWM) pins were extremely useful when adjusting the speed of motors during the testing phase.

In terms of the design process, since most of the team had prior experience with Arduino, no training was required other than learning to host a webpage with the integrated ESP32 module. The Arduino platform also comes with various libraries such as the <*Servo*>, <*ezBuzzer*>, and <*LiquidCrystal*> libraries which were mostly responsible for controlling the GPIOs of KIRBY [1].

3.0 Conclusion & Future Recommendations

The current embedded system implements a card dealer using an Arduino Uno R4 with Wifi communication to take in user input such as speed setting, number of players, and game type, to deal cards using the two DC motors while displaying audio and visuals via the speaker and LCD. These components are housed in a structure of 3D-printed parts. The system uses a finite state machine hosted with *webServer()* on Arduino Uno R4 and functions to deal cards/display (*dealCards()* & *sendLCD()*) are called within states. Techniques such as optimization, using subroutines, and interrupts are used to efficiently allocate resources and run the program. Calibration and testing were done to ensure the functionality and precision of the dealer. A lesson learned is that calibration for the motor encoders should be done with the full weight of the completed system at the end to ensure more precise turns rather than calibrating with no load.

A future change that could be made to the current design to add complexity includes using distance sensors and another encoder DC motor to determine the distance between the user and the dealer and then adjusting the speed of the outputted card accordingly. This would require implementing analog-to-digital conversion techniques such as using external analog-to-digital converters, which would be costly but more efficient [7]. As well, various filtering techniques would be required to reduce noise while obtaining sensor values. Given the course knowledge the team has now, a control system would be implemented such as PID control using control loops to use feedback to control the speed of the motor outputting the cards. The control system would effectively reduce noise and overcome any disturbances present in the environment. Input would be collected using interrupt-based methods as data only needs to be collected after each base rotation, allowing the CPU to perform other tasks such as rotating the base motor [8]. Since the user could be sitting at any distance, the control system would require past reference input values and as such, the sensor input would be stored using a rolling buffer and calculating the reference signal can be done using the free-running built-in timer.

This additional hardware component would increase the number of GPIO, which requires an additional microcontroller unit. During the integration process, the team used an additional Arduino and implemented the I2C protocol to communicate between the devices [9]. The team had implemented I2C protocols to communicate between different Arduinos in the intermediate steps of the project, but decided not to implement them as the current design does not require such protocols, and would add unnecessary complexities to the project. Thus, this protocol can easily be implemented if distance sensors were to be used.

4.0 References

[1] M.Mackay. (2024). *MIE438 - Lecture Slides 1 - Introduction to Microcontrollers and Embedded Systems* [Slides].

[2] M.Mackay. (2024). *MIE438 - Lecture Slides 12 - State Machines* [Slides].

[3] M.Mackay. (2024). *MIE438 - Lecture Slides 6 - Code Reuse and Subroutines* [Slides].

[4] M.Mackay. (2024). *MIE438 - Lecture Slides 2 - Numbering Systems* [Slides].

[5] M.Mackay. (2024). *MIE438 - Lecture Slides 7 - Variable Storage, Memory Use, Compilation, and Optimization* [Slides].

[6] M.Mackay. (2024). *MIE438 - Lecture Slides 8 - Timers and Interrupts* [Slides].

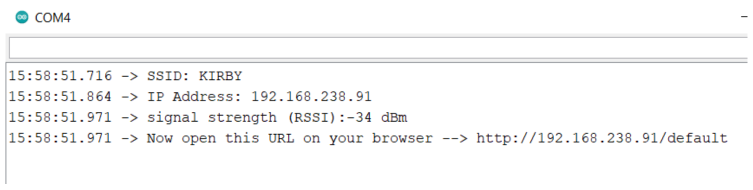
[7] M.Mackay. (2024). *MIE438 - Lecture Slides 9 - Analog Input and ADCs* [Slides].

[8] M.Mackay. (2024). *MIE438 - Lecture Slides 11 - Control Loops* [Slides].

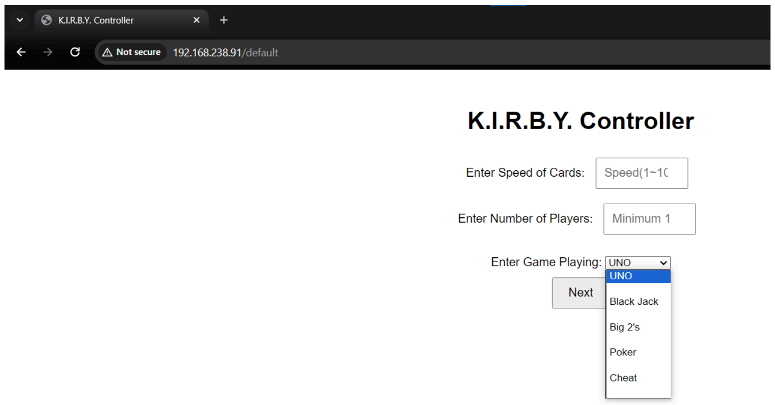
[9] M.Mackay. (2024). *MIE438 - Lecture Slides 10 - Digital Communication Protocols* [Slides].

Appendices

Appendix A: Webpages of Different States of KIRBY


COM4

```
15:58:51.716 -> SSID: KIRBY
15:58:51.864 -> IP Address: 192.168.238.91
15:58:51.971 -> signal strength (RSSI):-34 dBm
15:58:51.971 -> Now open this URL on your browser --> http://192.168.238.91/default
```


K.I.R.B.Y. Controller

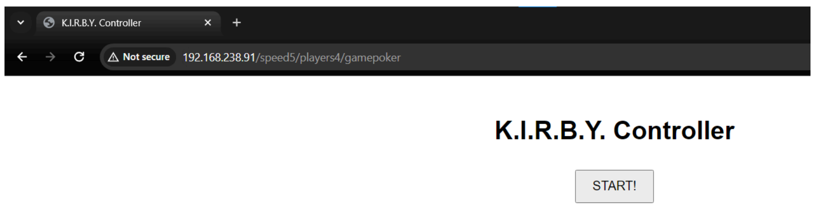
Not secure 192.168.238.91/default

K.I.R.B.Y. Controller

Enter Speed of Cards: Speed(1~10

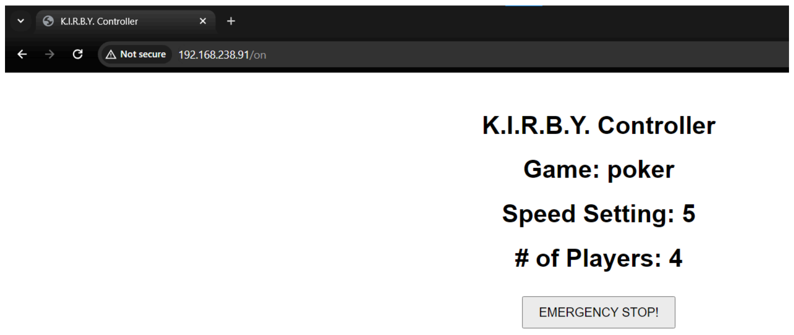
Enter Number of Players: Minimum 1

Enter Game Playing: UNO


K.I.R.B.Y. Controller

Not secure 192.168.238.91/speed5/players4/gamepoker

K.I.R.B.Y. Controller


K.I.R.B.Y. Controller

Not secure 192.168.238.91/on

K.I.R.B.Y. Controller

Game: poker

Speed Setting: 5

# of Players: 4

Appendix B: Specification of Global Variables

```
//---TIMEOUT VARIABLES---
unsigned long currentTime = millis(); //current time
unsigned long previousTime = 0; //previous time
const long timeoutTime = 2000; //define timeout time
```

Appendix C: Use of Floating-point Representation

```
int RotateBase(float rotDegrees) {
  float encValChange = (rotDegrees/degEncRatio);
  int prevCount = baseMotorCount;
```

Appendix D: Before and After Optimization

```
void musicEnd(){
  for (noteIndex = 0; noteIndex <
sizeof(endMelody); noteIndex++){
    int noteDuration = 1000 /
endNoteDurations[noteIndex];
    tone(musicPin,
endMelody[noteIndex],
noteDuration);
    int pauseBetweenNotes =
noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(musicPin);
  }
}
```

```
int pauseBetweenNotes =
noteDuration * 1.30;
void musicEnd(){
  for (noteIndex = 0; noteIndex <
sizeof(endMelody); noteIndex++){
    int noteDuration = 1000 /
endNoteDurations[noteIndex];
    tone(musicPin,
endMelody[noteIndex],
noteDuration);
    delay(pauseBetweenNotes);
    noTone(musicPin);
  }
}
```

Appendix E: Interrupt Functions

```
void InitInterrupts(void){
  pinMode(baseEncA, INPUT);
  pinMode(baseEncB, INPUT);
  attachInterrupt(digitalPinToInterrupt(baseEncA), EncoderEvent, CHANGE);
}
```

Appendix F: Low-level Encoder Control

```
void EncoderEvent() {
  if (digitalRead(baseEncA) == HIGH) {
    if (digitalRead(baseEncB) == LOW) {
      baseMotorCount--;
    } else {
      baseMotorCount++;
    }
  } else {
    if (digitalRead(baseEncB) == LOW) {
      baseMotorCount++;
    } else {
      baseMotorCount--;
    }
  }
}
```