Group Name: Delta

Project 02 – Kart-It Project

Design Overview:

High-Level Design Overview:

This toy would be a driving/Mario Kart style bop-it spinoff that would incorporate actual driving actions for the user inputs. The enclosure would be a console-type platform with the game start and power buttons easily accessible. Game input mechanisms would resemble actual car parts.

1. **Steer it**
    a. Players would use a central steering wheel to steer based on the command.
    b. Could incorporate “left” and “right” commands for an added challenge
    c. Inputs could be collected using a rotary encoder to detect the direction of the wheel
    d. Could implement a turn-signal that must be used before the steering wheel can be turned
2. **Shift it**
    a. Players would use a sequential shifter device to shift into gear.
    b. This will be implemented using limit switches to detect the shift up/down user command.
3. **Gas it/Break it**
    a. Players could press the gas pedal when prompted.
        a. Same logic will be used for the separate brake pedal as well.
    b. Potentiometers are used in order to detect the gas/brake pedal being pressed beyond a certain threshold.
    c. Both pedals are attached to the enclosure to be pressed with a hand and an extension cable is used so that the player could press it with their foot as well.

This design for our game will involve using the ATMega328P chip in order to manage the functionalities of the game, such as processing game logic, interfacing with input sensors, and controlling the output signals. The ATMega328P chip will also be used to communicate with a Raspberry Pi in order to emulate the N64 version of Mario Kart. There will be hardcoded checkpoints in the N64 Mario Kart tracks that will display user commands to the user depending on the current part of the selected track. The Raspberry Pi will communicate with the ATMega chip via

two digital pins, which will either detect a pass or fail for the current command on the game will use the hardware serial pins (0 and 1) to communicate with the ATMega328P. Initially, we did not think about the Mario Kart emulator via the Raspberry Pi; however, we had thought that it would be a great addition to the project.

The three distinctly different sensory inputs we are using are limit switches, potentiometers, and rotary encoder. In our game, there will be two limits that either detect a shift up command or a shift down command. The purpose of these limit switches is for the sequential shifter to hit the corresponding limit for the current shifting command. There will also be two potentiometers that either detect a gas it commands or a brake it command. The purpose of these potentiometers is to detect pedal press if the corresponding pedal is pressed beyond a certain threshold for it to register as a press. Finally, there will be a rotary encoder in order to detect a left and a right turn. The rotary encoder uses two signals, A and B, in order to differentiate between a left and right turn and checks in which direction the two signals are 90 degrees out of phase from each other. Other components incorporated in this game include an LCD display, seven segment display, speaker, and a DFPlayer Mini module. The LCD display is to allow the players to select the desired track they would like to play in, which includes Freestyle, Rainbow Road, Bowser's Castle, and Baby Park. The Freestyle game mode generates random user commands, while the other three tracks other hardcoded to replicate how the player actually driving on that track. The LCD display also displays "Correct Response" if the user command was done properly, and display "Game Over" if the user command was either not done on time or done incorrectly. If the player successfully beats the game, then the LCD will display "You won the game" to the player in indicate that the game is done, and the user would have to press the reset button on the system in order to play the game again. The purpose of the seven-segment display is to display the total amount of points the user has gotten in the game. The DFPlayer Mini module and the speaker were used to play the audio for the user commands, correct response, incorrect response, and the introduction for the game as well.

The game will have an on/off rocker switch in order to turn on the entire game and there will also be a start button in order to start the game after being powered on. Pressing the start button will allow the users to select the desired tracks and will operate the same way as mentioned above. There will be left and right arrows in the system in order to go through the track select menu. After selecting the desired track, the user commands will start to emit distinct, audible commands for the user to complete. These user commands will be one of the six corresponding actions, as defined above. If the user does the command successfully, then the LCD display will display "Correct Response,"

and one point will be awarded to the player. Also, the time interval between successive commands becomes smaller with each successful attempt and, in the freestyle mode, after 99 successful attempts, the game ends and the user are notified of their score. For the hardcoded tracks, there will be different maximum points that the player is able to get depending how long the track is. On the other hand, if the user is unsuccessful in responding to the command either by not responding in time or providing an incorrect response, the LCD display will display “Game Over” and the user can play another round by playing the reset button on the game. We will be 3D printing the sequential shifter and the gas/brake pedals as well. We got the design for the sequential shifter from Beavis Motorsport, and got the design for the gas/brake pedals from thinkiverse.

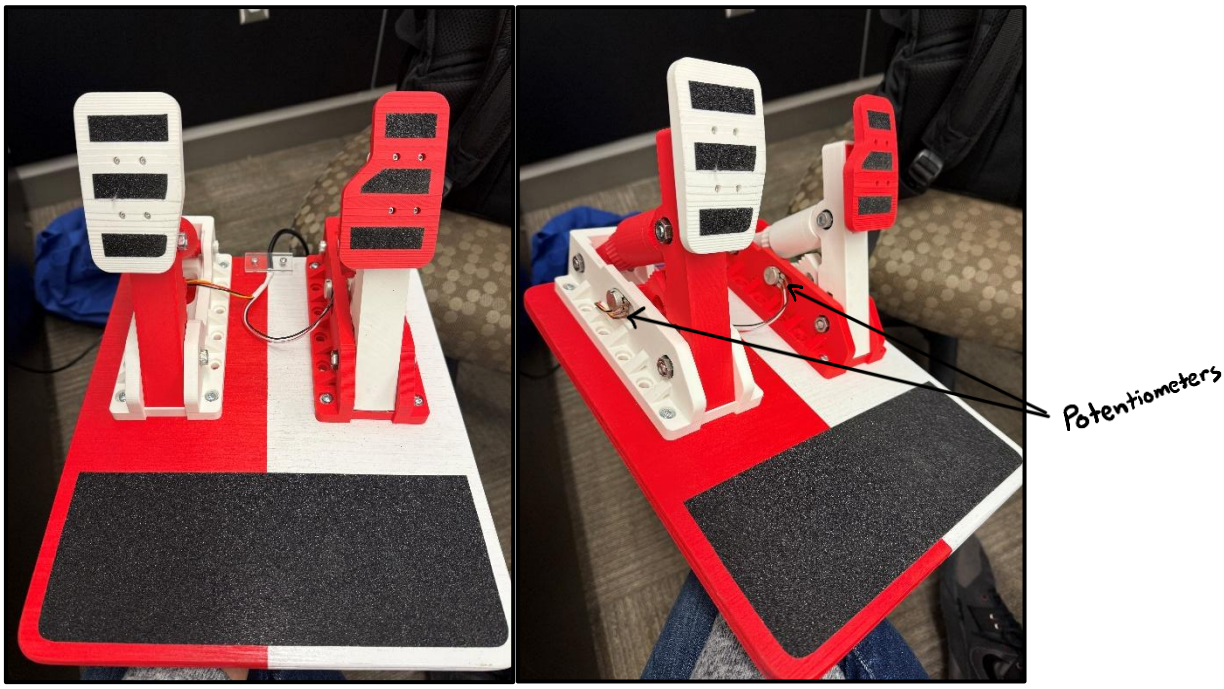


Figure 1, 2: Image of Gas/Brake Pedals

As seen by the screenshot above, the gas/brake pedals have grip tape on them in order to allow the players to have a solid grip to press the pedals. Figure 2 shows the two potentiometers we are using for both pedals in order to detect a gas or brake command depending on if the pedal is pressed past its threshold.

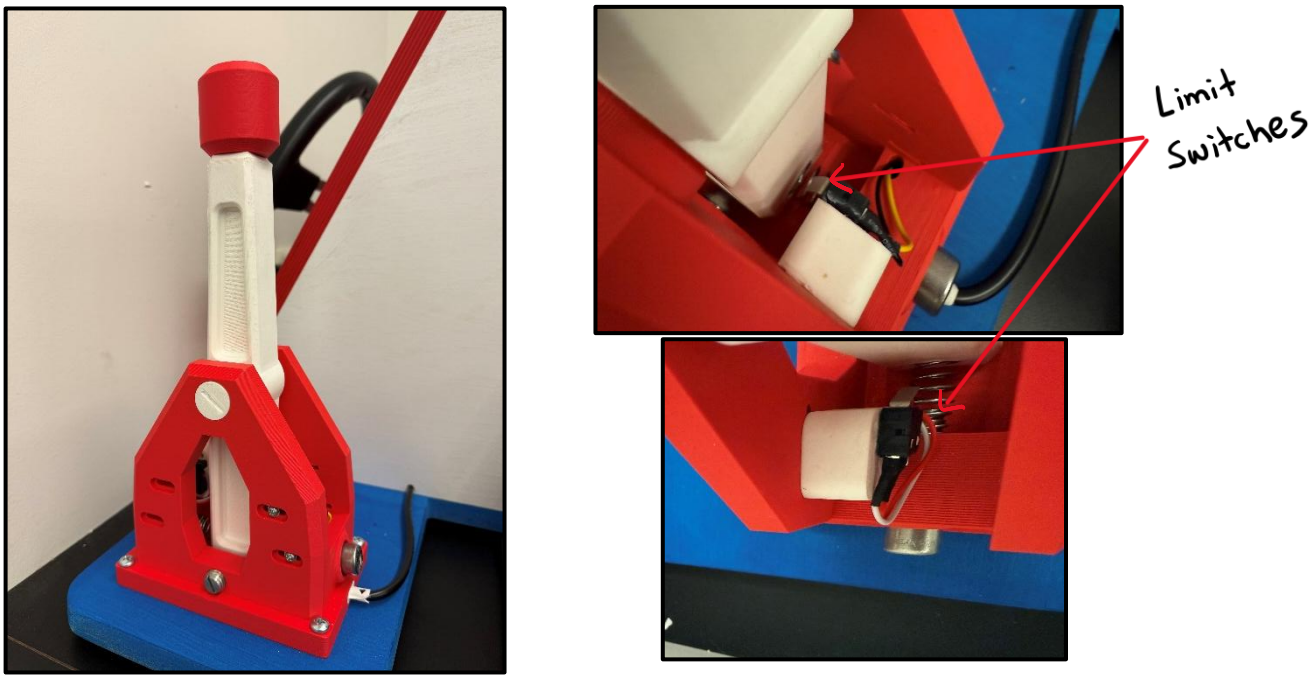


Figure 3, 4, 5: Image of Sequential Shifter

As seen by the screenshot above, the sequential shifter has limit switches on each of the ends in order to register the shift up and shift down command. Figures 4 and 5 show the placement of the limit switches. We designed it such that the shaft of the shifter does not fully push the limit switch in so that it does not wear out as fast.

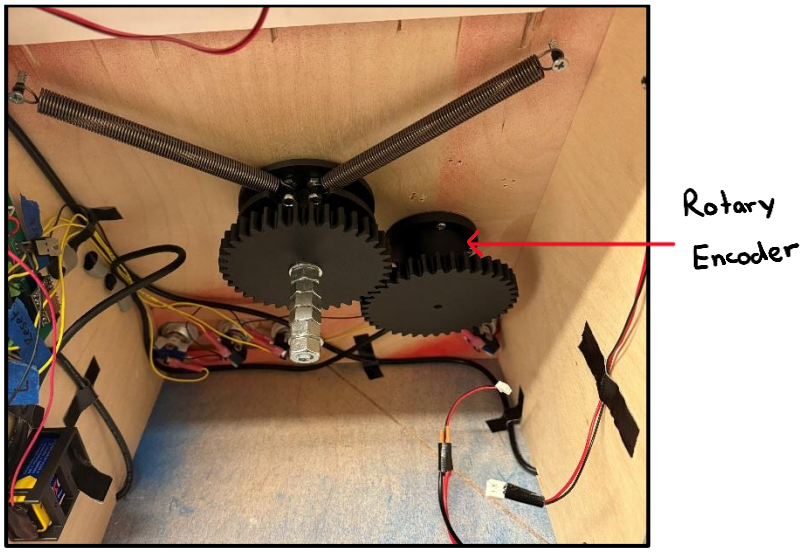


Figure 6: Image of Rotary Encoder and the Steering Column

As seen by the screenshot above, the rotary encoder is inside of the encoder enclosure screwed into the plywood. The way we decided to detect the player has turned the steering wheel is by attaching gears to both the steering column and the shaft of the rotary encoder. We also added a return mechanism to the steering wheel by adding springs to each sides to make the steering wheel oscillate back to the origin. We have also added a hard stop at the 180-degree mark of the steering column by adding nuts to prevent the wheel from turning any more.

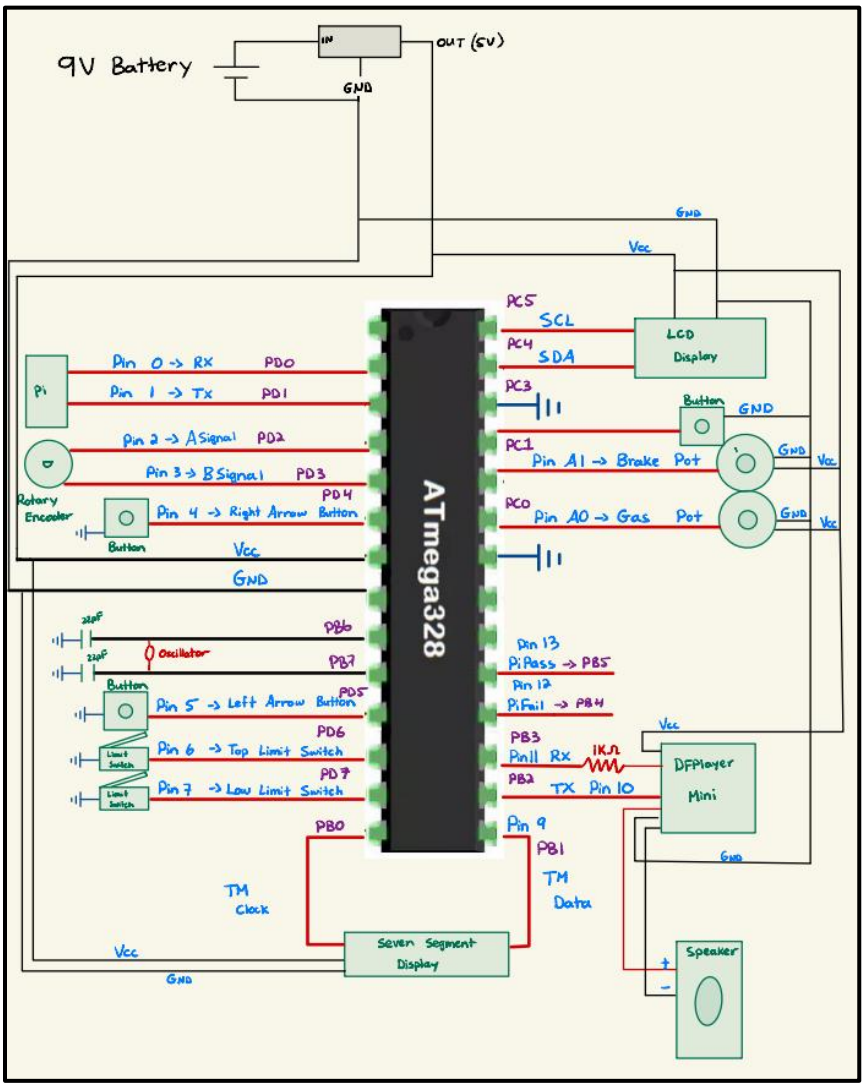


Figure 7 : Initial Schematic of Design

As seen by the screenshot above, our schematic for the design incorporates all of the components explained above. We assigned all of the input/output components to the desired digital or analog pins on the ATMega328P. We

ended up running out of pins, but we did not need to add any more pins from the ATMega328P. There is a voltage regulator on the top of the schematic which steps down the voltage coming from the 9V battery down to 5 volts in order to power on the ATMega chip and the components as well. Since the chip does not have its own SDA and SCL pins, I had to use A4 and A5 in order to power on I2C module components, such as LCD display. The pin numbers, such as PB0, PB1,… are there in order to make our Altium schematic easier to implement, since we already have all pin assignments in the screenshot above.

Design Verification:

Prototype Verification

Regarding the hardware aspect of the design, we decided to use an Arduino dev board to prototype the logic for the game.

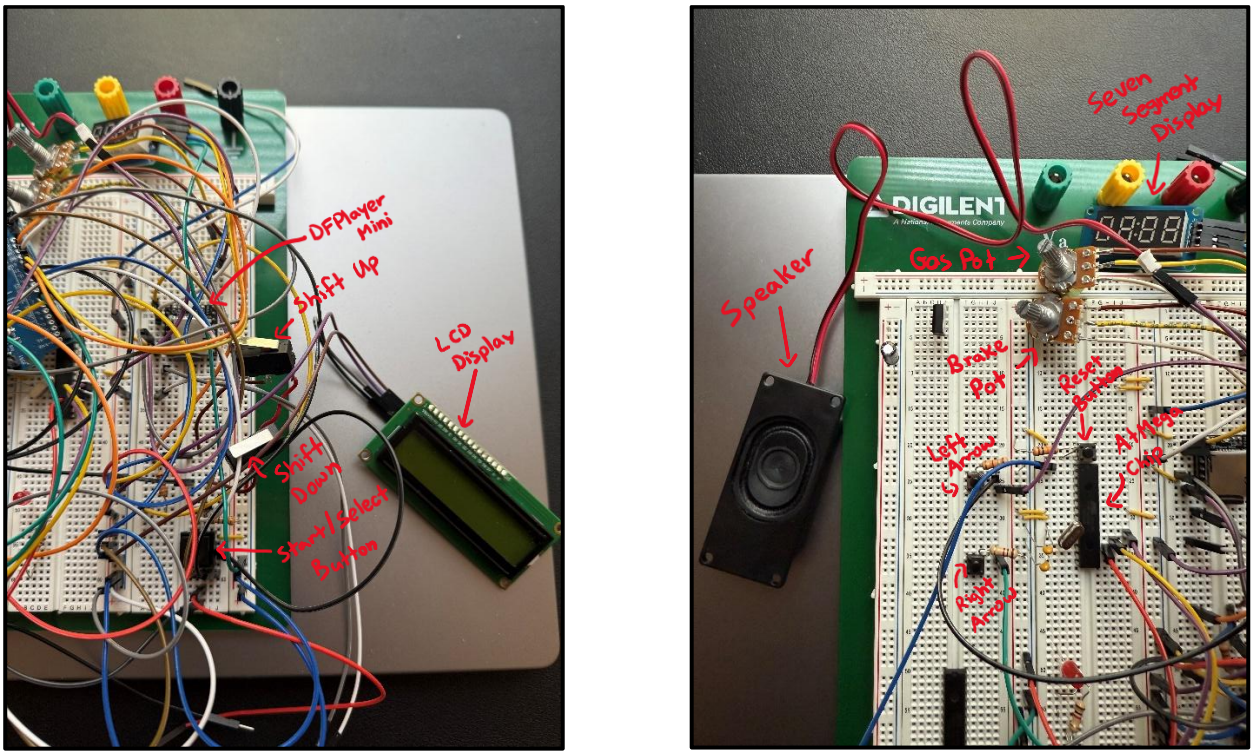


Figure 8, 9: Breadboard Prototype of Design

As seen in the screenshot above, this is the prototype of our design where we put our circuit schematic into a breadboard. We decided to verify all parts of our design on our breadboard to make sure all of the parts work as intended before we design a PCB for this design. We did not want to design a PCB for a design that was not fully prototypes on a breadboard, since that could lead to certain parts of the design not working as intended. The schematic we based out breadboard prototype off of was provided earlier in Figure 7. In Figure 8, we implemented the two limit switches used for the sequential shifter, the LCD display, the start/select button, and the DFPlayer Mini. You can also see the Arduino Uno dev board we are using in order to prototype the game. In Figure 9, we implemented the seven-segment display, the speaker for the DFPlayer Mini, the gas/brake potentiometers, left/right arrow buttons, and the reset button. All of the components are labeled on the figures above. The purposes of all of the components are the same as they were explained above.

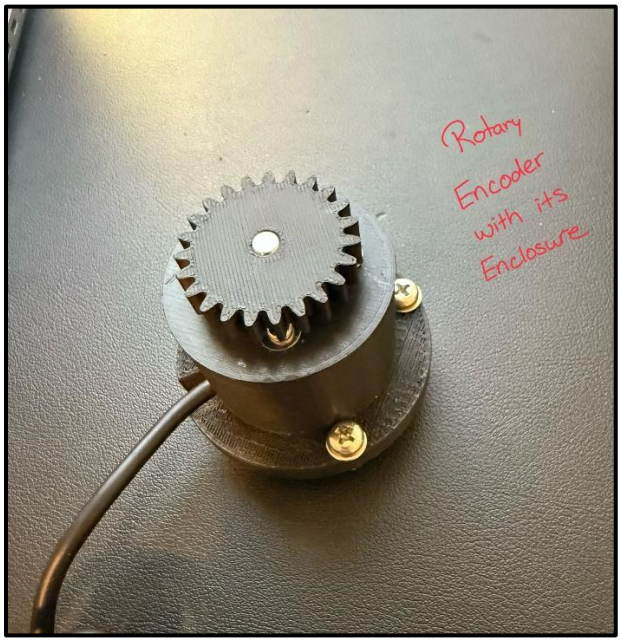


Figure 10: Prototype of Rotary Encoder Enclosure

As seen in the screenshot above, this is the prototype of our design for the enclosure for the rotary encoder. We plan on screwing in the encoder to the inner part of the enclosure, as seen in Figure 6, in order to make the rotary encoder turn alongside the wheel. Initially, we had gears that were too small and did not have enough clearance for in order to reach the other gear mounted onto the middle bolt.

Group Name: Delta

Video of Final Breadboard Prototype Functionality:

Click on this Link to Access Video of Functionality

As seen by the video above, the functionality of the game works as such:

- The player must press the start game button to play the game.
- There are multiple tracks the users can select.
- If the player does the correct command, then it moves onto the next command.
- If the player does not do the correct command, then the game is over.

Regarding the software aspect of the design, we decided to use helper functions for all of the individual sensors and then integrate them together into one file and then add logic around them in order to get the game fully functional.

Helper Files:

The helper files below will help us integrate all of the different sensors into the main file after testing them individually.

Limit Switch Code:

Limit_Switch.ino

```
// TEST THIS LOGIC OUT WITH AN LED OUTPUT

const int limitSwitchPin = 1;  // Define the pin for the limit switch
const int LEDPin = 2;


void setup() {
    pinMode(limitSwitchPin, INPUT_PULLUP);
    pinMode(LEDPin, OUTPUT);

    Serial.begin(9600);
}

void loop() {
    if (digitalRead(LIMIT_SWITCH_PIN) == LOW) {
        Serial.println("Limit Switch has been Pressed");
        digitalWrite(LEDPin, HIGH);
    } else {
        Serial.println("Limit Switch has been Released");
        digitalWrite(LEDPin, LOW);
    }
}

```

Figure [1]: Limit Switch Test File

The screenshot above shows the code needed to operate the limit switch by itself. We start off by assigning pins to the control pin of the limit switch and a pin for the LED to show the output of the limit switch state. The pinMode of the limit switch pin is set to be a pull-up, which means that the switch will have an Active Low state. If the switch is active, then the LED pin is turned on, vice versa.

Hall Effect Magnetic Sensor Code:

Magnetic_Sensor.ino

```
// TEST THIS LOGIC OUT WITH AN LED OUTPUT

const int magneticSensorPin = 3;
const int LEDPin = 4;

void setup() {
    pinMode(magneticSensorPin, INPUT_PULLUP);
    pinMode(LEDPin, OUTPUT);

    Serial.begin(9600);
}

void loop() {
    if (digitalRead(magneticSensorPin) == LOW) {
        Serial.println("Magnet is in front of Sensor");
        digitalWrite(LEDPin, HIGH);
    } else {
        Serial.println("Magnet is not in front of Sensor");
        digitalWrite(LEDPin, LOW);
    }
}

```

Figure [2]: Magnetic Sensor Test File

The screenshot above shows the code needed to operate the magnetic sensor by itself. We start off by assigning pins to the control pin of the magnetic sensor and a pin for the LED to show the output of the magnetic sensor state. The pinMode of the magnetic sensor pin is set to be a pull-up, which means that the switch will have an Active Low state. If the magnetic sensor is active, then the LED pin is turned on, vice versa.

Group Name: Delta

Rotary Encoder Code:

Rotary_Encoder.ino

```
3
4   const int clockPin = 5;
5   const int DTPin = 6;
6   const int switchPin = 7;
7
8   // Other needed variables
9   int counter = 0;
10  int currentStateCLK;
11  int lastStateCLK;
12  String currentDir ="";
13  unsigned long lastButtonPress = 0;
14
15  void setup() {
16
17    // Set encoder pins as inputs
18    pinMode(clockPin,INPUT);
19    pinMode(DTPin,INPUT);
20    pinMode(switchPin, INPUT_PULLUP);
21
22    // Setup Serial Monitor
23    Serial.begin(9600);
24
25    // Read the initial state of CLK
26    lastStateCLK = digitalRead(clockPin);
27  }
28
29  void loop() {
30
31    // Read the current state of CLK
32    currentStateCLK = digitalRead(clockPin);
33
34    // If last and current state of CLK are different, then pulse occurred
35    // React to only 1 state change to avoid double count
36    if (currentStateCLK != lastStateCLK  && currentStateCLK == 1){
37
38      // If the DT state is different than the CLK state then
39      // the encoder is rotating CCW so decrement
40      if (digitalRead(DTPin) != currentStateCLK) {
41        counter --;
42        currentDir ="CCW";
43      } else {
44        // Encoder is rotating CW so increment
45        counter ++;
46        currentDir ="CW";
47      }
48
49      Serial.print("Direction: ");
50      Serial.print(currentDir);
51      Serial.print(" | Counter: ");
52      Serial.println(counter);
53    }
54
55    // Remember last CLK state
56    lastStateCLK = currentStateCLK;
57  }
```

Figure [3]: Rotary Encoder Test File

The screenshot above shows the code obtained from the Arduino website that helps us understand how the rotary encoder works and how to determine what direction the encoder is turned by the user. This code from their website will be extremely useful to us and a good starting point to implement the rotary encoder.

Audio Output Code:

Audio_Output.ino

```
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>

const int RXPin = 10;
const int TXPin = 11;

SoftwareSerial softwareSerial(RXPin, TXPin);
DFRobotDFPlayerMini player;


void setup() {
    Serial.begin(9600);
    softwareSerial.begin(9600);

    if (player.begin(softwareSerial)) {
      Serial.println("OK");
    } else {
      Serial.println("FAIL!");
    }

    player.volume(20); // Set volume to 20 (out of 30)
}

void loop() {
    int track = random(1, 4);

    player.play(track);
}
```

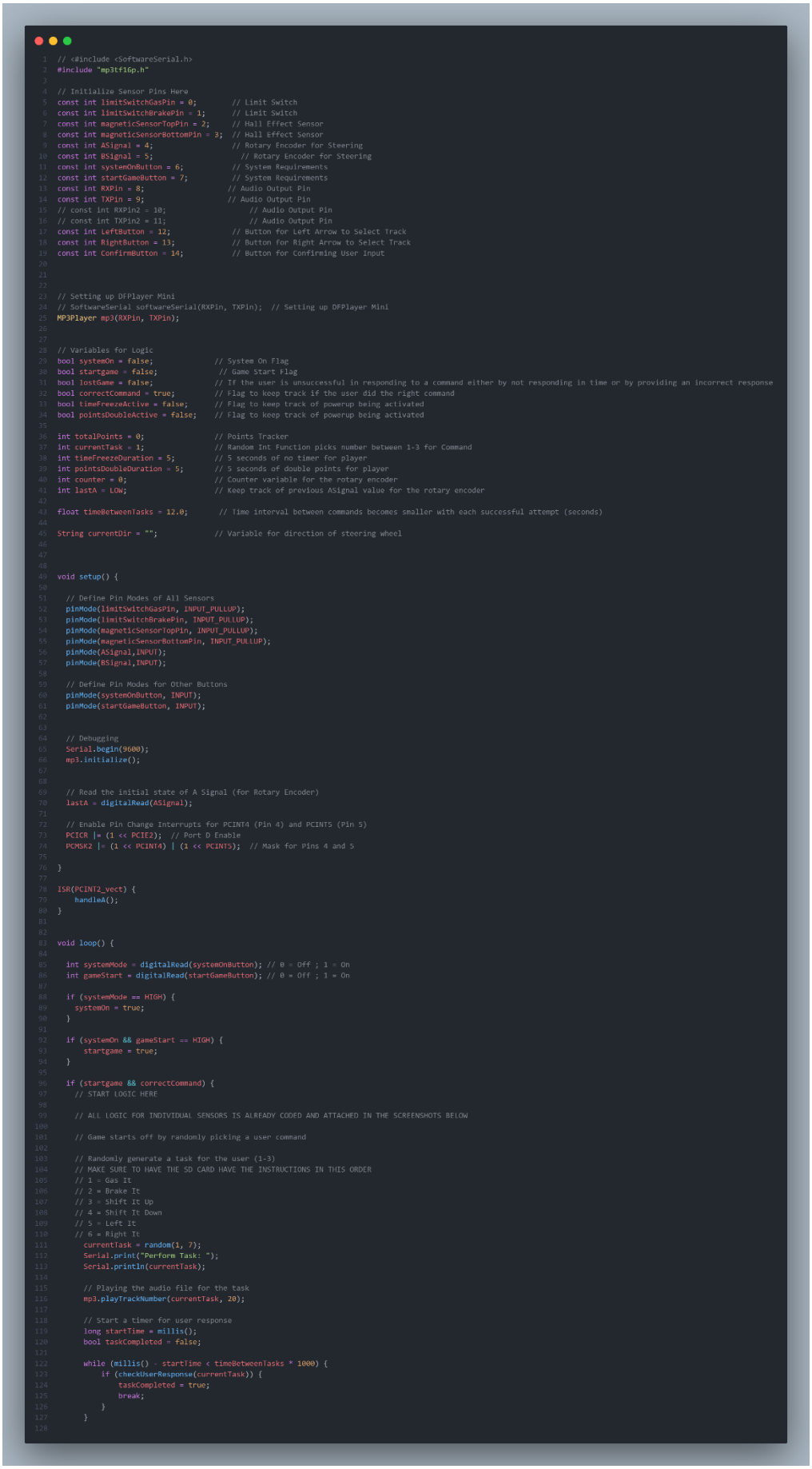
Figure [4]: Magnetic Sensor Test File

The screenshot above shows the code used to play a pre-recorded MP3 audio file via a DFPlayer Mini module and a MicroSD Card. Since there is three different commands in our design, the software will randomly pick a number from 1-3 and the corresponding audio will be played in the speaker that is connected to the DFPlayer Mini module. This code was obtained from a YouTube video I was watching in order to get a better understanding of the module and its connection with the speaker.

Main Logic File:

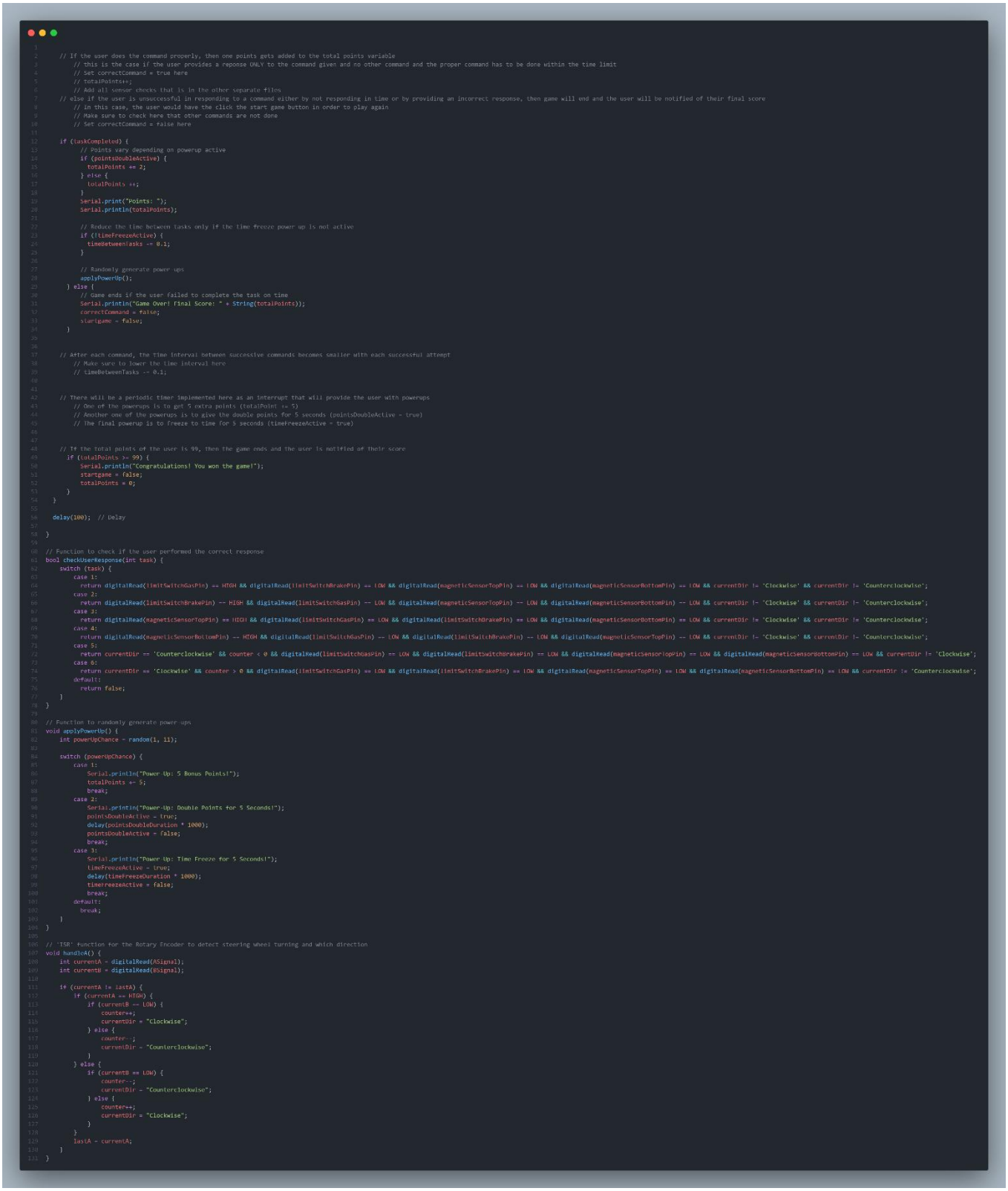
The main file will integrate all of the helper files for all of the sensors we will be using in order to bring our proposed design to life.

Group Name: Delta



```
// <#include <SoftwareSerial.h>
#include "mp3tf16p.h"

// Initialize Sensor Pins Here
const int limitSwitchGasPin = 0;        // Limit Switch
const int limitSwitchBrakePin = 1;      // Limit Switch
const int magneticSensorTopPin = 2;     // Hall Effect Sensor
const int magneticSensorBottomPin = 3;  // Hall Effect Sensor
const int ASignal = 4;                  // Rotary Encoder for Steering
const int BSignal = 5;                    // Rotary Encoder for Steering
const int systemOnButton = 6;           // System Requirements
const int startGameButton = 7;          // System Requirements
const int RXPin = 8;                   // Audio Output Pin
const int TXPin = 9;                   // Audio Output Pin
// const int RXPin2 = 10;                    // Audio Output Pin
// const int TXPin2 = 11;                    // Audio Output Pin
const int LeftButton = 12;              // Button for Left Arrow to Select Track
const int RightButton = 13;             // Button for Right Arrow to Select Track
const int ConfirmButton = 14;           // Button for Confirming User Input



// Setting up DFPlayer Mini
// SoftwareSerial softwareSerial(RXPin, TXPin);  // Setting up DFPlayer Mini
MP3Player mp3(RXPin, TXPin);


// Variables for Logic
bool systemOn = false;              // System On Flag
bool startgame = false;              // Game Start Flag
bool lostGame = false;              // If the user is unsuccessful in responding to a command either by not responding in time or by providing an incorrect response
bool correctCommand = true;         // Flag to keep track if the user did the right command
bool timeFreezeActive = false;      // Flag to keep track of powerup being activated
bool pointsDoubleActive = false;    // Flag to keep track of powerup being activated

int totalPoints = 0;                // Points Tracker
int currentTask = 1;                // Random Int Function picks number between 1-3 for Command
int timeFreezeDuration = 5;         // 5 seconds of no timer for player
int pointsDoubleDuration = 5;       // 5 seconds of double points for player
int counter = 0;                    // Counter variable for the rotary encoder
int lastA = LOW;                    // Keep track of previous ASignal value for the rotary encoder

float timeBetweenTasks = 12.0;       // Time interval between commands becomes smaller with each successful attempt (seconds)

String currentDir = "";             // Variable for direction of steering wheel



void setup() {

  // Define Pin Modes of All Sensors
  pinMode(limitSwitchGasPin, INPUT_PULLUP);
  pinMode(limitSwitchBrakePin, INPUT_PULLUP);
  pinMode(magneticSensorTopPin, INPUT_PULLUP);
  pinMode(magneticSensorBottomPin, INPUT_PULLUP);
  pinMode(ASignal,INPUT);
  pinMode(BSignal,INPUT);

  // Define Pin Modes for Other Buttons
  pinMode(systemOnButton, INPUT);
  pinMode(startGameButton, INPUT);


  // Debugging
  Serial.begin(9600);
  mp3.initialize();


  // Read the initial state of A Signal (for Rotary Encoder)
  lastA = digitalRead(ASignal);

  // Enable Pin Change Interrupts for PCINT4 (Pin 4) and PCINT5 (Pin 5)
  PCICR |= (1 << PCIE2);  // Port D Enable
  PCMSK2 |= (1 << PCINT4) | (1 << PCINT5);  // Mask for Pins 4 and 5

}

ISR(PCINT2_vect) {
    handleA();
}


void loop() {

  int systemMode = digitalRead(systemOnButton); // 0 = Off ; 1 = On
  int gameStart = digitalRead(startGameButton); // 0 = Off ; 1 = On

  if (systemMode == HIGH) {
    systemOn = true;
  }

  if (systemOn && gameStart == HIGH) {
      startgame = true;
  }

  if (startgame && correctCommand) {
    // START LOGIC HERE

    // ALL LOGIC FOR INDIVIDUAL SENSORS IS ALREADY CODED AND ATTACHED IN THE SCREENSHOTS BELOW

    // Game starts off by randomly picking a user command

    // Randomly generate a task for the user (1-3)
    // MAKE SURE TO HAVE THE SD CARD HAVE THE INSTRUCTIONS IN THIS ORDER
    // 1 = Gas It
    // 2 = Brake It
    // 3 = Shift It Up
    // 4 = Shift It Down
    // 5 = Left It
    // 6 = Right It
      currentTask = random(1, 7);
      Serial.print("Perform Task: ");
      Serial.println(currentTask);

      // Playing the audio file for the task
      mp3.playTrackNumber(currentTask, 20);

      // Start a timer for user response
      long startTime = millis();
      bool taskCompleted = false;

      while (millis() - startTime < timeBetweenTasks * 1000) {
          if (checkUserResponse(currentTask)) {
              taskCompleted = true;
              break;
          }
      }
```

Group Name: Delta



```
    // If the user does the command properly, then one points gets added to the total points variable
        // this is the case if the user provides a reponse ONLY to the command given and no other command and the proper command has to be done within the time limit
        // Set correctCommand = true here
        // totalPoints++;
        // Add all sensor checks that is in the other separate files
    // else if the user is unsuccessful in responding to a command either by not responding in time or by providing an incorrect response, then game will end and the user will be notified of their final score
        // in this case, the user would have the click the start game button in order to play again
        // Make sure to check here that other commands are not done
        // Set correctCommand = false here

    if (taskCompleted) {
        // Points vary depending on powerup active
        if (pointsDoubleActive) {
          totalPoints += 2;
        } else {
          totalPoints ++;
        }
        Serial.print("Points: ");
        Serial.println(totalPoints);

        // Reduce the time between tasks only if the time freeze power up is not active
        if (!timeFreezeActive) {
          timeBetweenTasks -= 0.1;
        }

        // Randomly generate power ups
        applyPowerUp();
    } else {
        // Game ends if the user failed to complete the task on time
        Serial.println("Game Over! Final Score: " + String(totalPoints));
        correctCommand = false;
        startGame = false;
    }


    // After each command, the time interval between successive commands becomes smaller with each successful attempt
        // Make sure to lower the time interval here
        // timeBetweenTasks -= 0.1;


    // There will be a periodic timer implemented here as an interrupt that will provide the user with powerups
        // One of the powerups is to get 5 extra points (totalPoint += 5)
        // Another one of the powerups is to give the double points for 5 seconds (pointsDoubleActive = true)
        // The final powerup is to freeze to time for 5 seconds (timeFreezeActive = true)


    // If the total points of the user is 99, then the game ends and the user is notified of their score
     if (totalPoints >= 99) {
        Serial.println("Congratulations! You won the game!");
        startGame = false;
        totalPoints = 0;
    }
  }

  delay(100);  // Delay

}

// Function to check if the user performed the correct response
bool checkUserResponse(int task) {
    switch (task) {
        case 1:
          return digitalRead(limitSwitchGasPin) == HIGH && digitalRead(limitSwitchBrakePin) == LOW && digitalRead(magneticSensorTopPin) == LOW && digitalRead(magneticSensorBottomPin) == LOW && currentDir != 'Clockwise' && currentDir != 'Counterclockwise';
        case 2:
          return digitalRead(limitSwitchBrakePin) == HIGH && digitalRead(limitSwitchGasPin) == LOW && digitalRead(magneticSensorTopPin) == LOW && digitalRead(magneticSensorBottomPin) == LOW && currentDir != 'Clockwise' && currentDir != 'Counterclockwise';
        case 3:
          return digitalRead(magneticSensorTopPin) == HIGH && digitalRead(limitSwitchGasPin) == LOW && digitalRead(limitSwitchBrakePin) == LOW && digitalRead(magneticSensorBottomPin) == LOW && currentDir != 'Clockwise' && currentDir != 'Counterclockwise';
        case 4:
          return digitalRead(magneticSensorBottomPin) == HIGH && digitalRead(limitSwitchGasPin) == LOW && digitalRead(limitSwitchBrakePin) == LOW && digitalRead(magneticSensorTopPin) == LOW && currentDir != 'Clockwise' && currentDir != 'Counterclockwise';
        case 5:
          return currentDir == 'Counterclockwise' && counter < 0 && digitalRead(limitSwitchGasPin) == LOW && digitalRead(limitSwitchBrakePin) == LOW && digitalRead(magneticSensorTopPin) == LOW && digitalRead(magneticSensorBottomPin) == LOW && currentDir != 'Clockwise';
        case 6:
          return currentDir == 'Clockwise' && counter > 0 && digitalRead(limitSwitchGasPin) == LOW && digitalRead(limitSwitchBrakePin) == LOW && digitalRead(magneticSensorTopPin) == LOW && digitalRead(magneticSensorBottomPin) == LOW && currentDir != 'Counterclockwise';
        default:
          return false;
    }
}

// Function to randomly generate power-ups
void applyPowerUp() {
    int powerUpChance = random(1, 11);

    switch (powerUpChance) {
        case 1:
            Serial.println("Power Up: 5 Bonus Points!");
            totalPoints += 5;
            break;
        case 2:
            Serial.println("Power-Up: Double Points for 5 Seconds!");
            pointsDoubleActive = true;
            delay(pointsDoubleDuration * 1000);
            pointsDoubleActive = false;
            break;
        case 3:
            Serial.println("Power Up: Time Freeze for 5 Seconds!");
            timeFreezeActive = true;
            delay(timeFreezeDuration * 1000);
            timeFreezeActive = false;
            break;
        default:
          break;
    }
}

// 'ISR' function for the Rotary Encoder to detect steering wheel turning and which direction
void handleA() {
    int currentA = digitalRead(ASignal);
    int currentB = digitalRead(BSignal);

    if (currentA != lastA) {
        if (currentA == HIGH) {
            if (currentB == LOW) {
                counter++;
                currentDir = "Clockwise";
            } else {
                counter--;
                currentDir = "Counterclockwise";
            }
        } else {
            if (currentB == LOW) {
                counter--;
                currentDir = "Counterclockwise";
            } else {
                counter++;
                currentDir = "Clockwise";
            }
        }
        lastA = currentA;
    }
}
```

Figure [5]: Main Logic File

The screenshot above shows some pseudo code that we have implemented to start the software portion of our design. We started off by assigning all of the sensors we will be using to a pin in the ATMega326P chip and initializing any other modules that we need to. Then, we started to get a list of the variables we would need to use for the logic of the game. In the setup function, we had to assign each Arduino pin we are using as an input or an output. Finally, in the loop function, we start off by checking if the device is powered on. If the device is turned on, we go on to check if the start game button has been pressed. Finally, if both of the conditions mentioned above are true, then we go on to implement the game. We have inserted some comments to explain how the ordering of the game is going to work. We are sure there will be more features added to the game, but this is the starting part of our implementation.

Regarding the enclosure aspect of the design, we decided to use AutoCAD and Fusion 360 in order to build our enclosure. Then, we plan on cutting the enclosure with 3/4" plywood with the CNC machine in the Makerspace.

Laser Cut File Images/Description of the Enclosure Design:

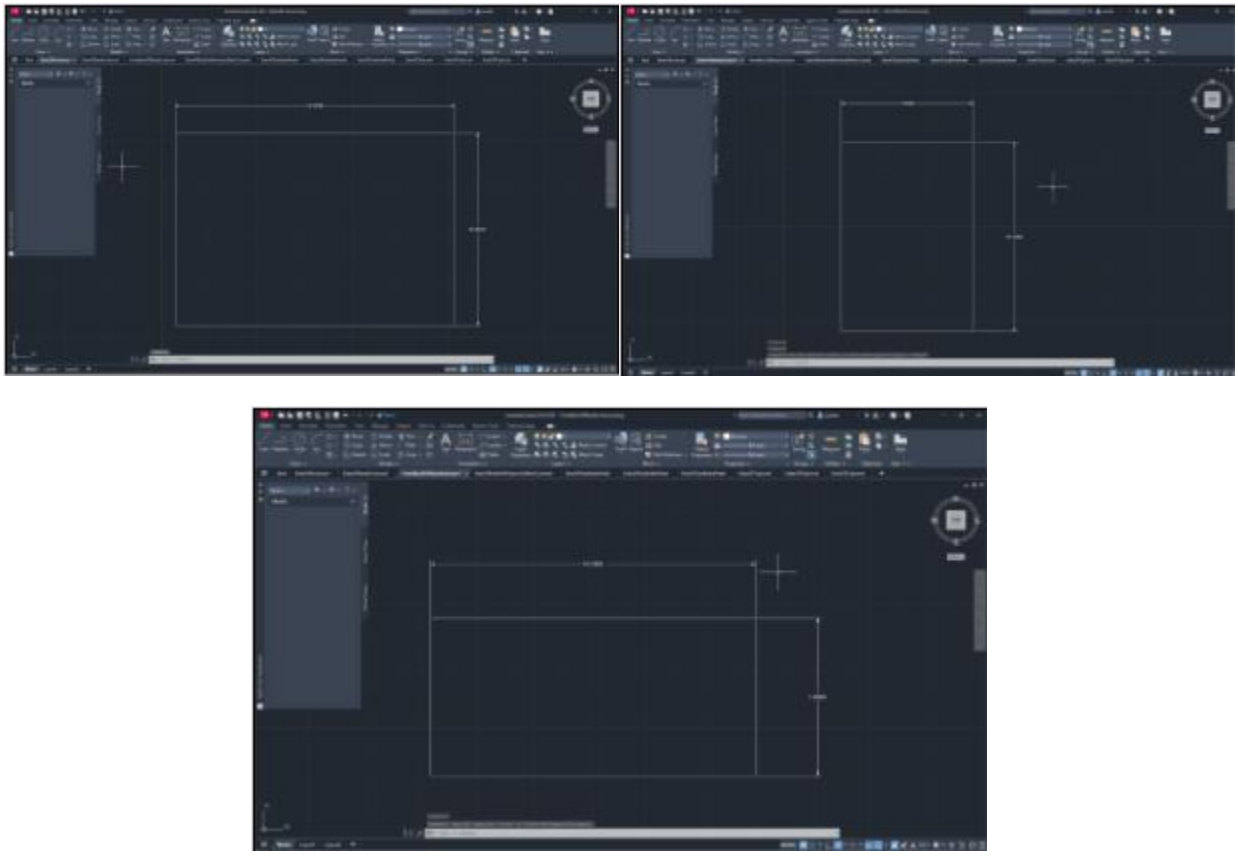


Figure [1-3]: Designs for Base of Enclosure

In the screenshots above, you can see the enclosure designed for the base of our game. We plan to make the base with dimensions of 10” x 14.5” as seen by the dimensions above. We plan to make the sides of the base 10” x 7”, which means that the base will have a height of seven inches. The front and back of the base will have dimensions of 14.5” x 7”. We plan to laser cut these cuts and will drill holes into the Plywood and screw screws into it in order to hold all of the pieces together.

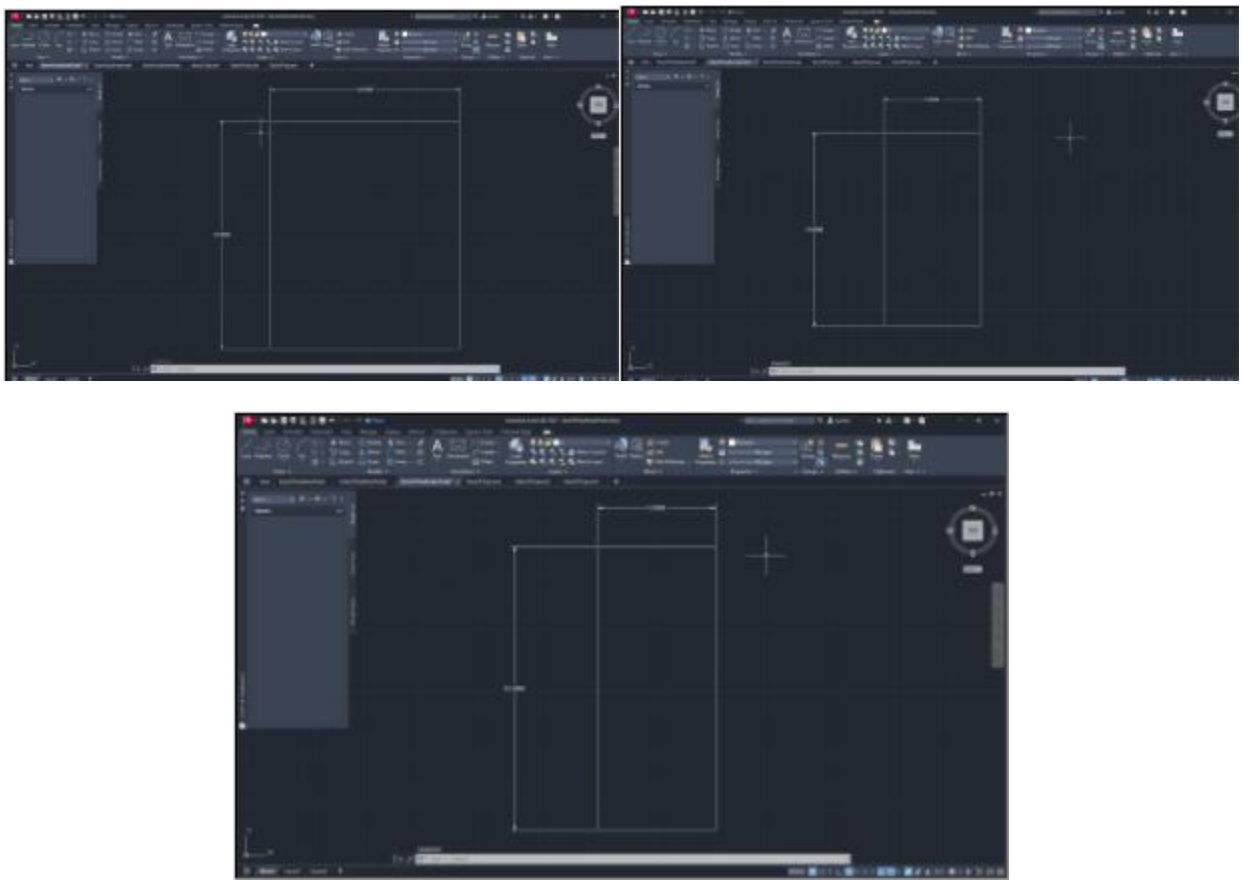


Figure [4-6]: Designs for Gas/Brake Pedal Section

In the screenshots above, you can see the enclosure designed for the section that will house the gas and break pedals. We plan to make the base with dimensions of 10” x 12” as seen by the dimensions above. We plan to make the sides of the base 10” x 5”, which means that the base will have a height of five inches. The back of the base will have dimensions of 12” x 5”. The front of the enclosure will be open so that the user can press the gas and brake pedals without any trouble. We plan to laser cut these cuts and will drill holes into the Plywood and screw screws into it in order to hold all of the pieces together.

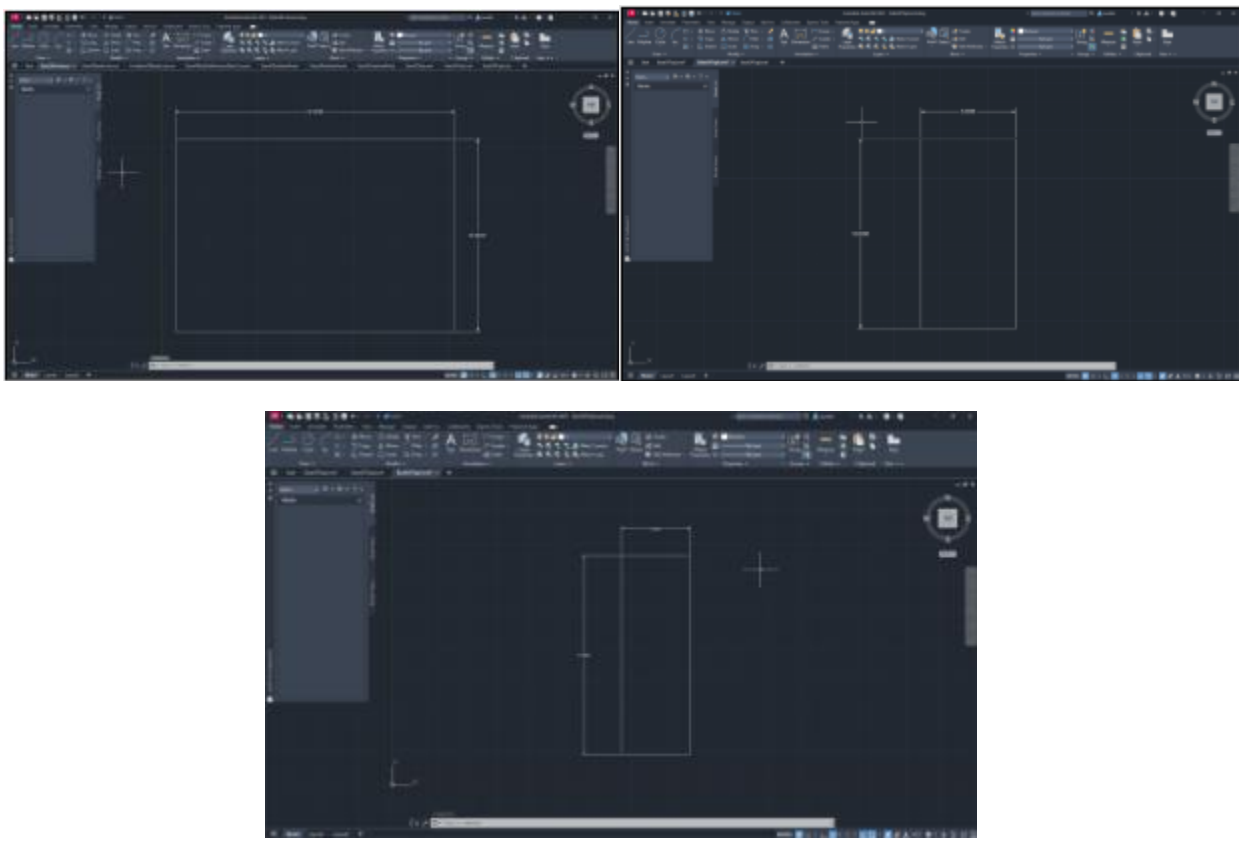


Figure [7-9]: Designs for Top Steering Wheel Section

In the screenshots above, you can see the enclosure designed for the section that will house the steering wheel. We plan to make the base with dimensions of 10” x 14.5” as seen by the dimensions above. We plan to make the sides of the base 10” x 5”, which means that the base will have a height of five inches. The back of the base will have dimensions of 14.5” x 5”. The front of the base will be cut in the CNC machine, since we need cuts that will be diagonal. This will be explained later in Figure 15. We plan to laser cut these cuts and will drill holes into the Plywood and screw screws into it in order to hold all of the pieces together.

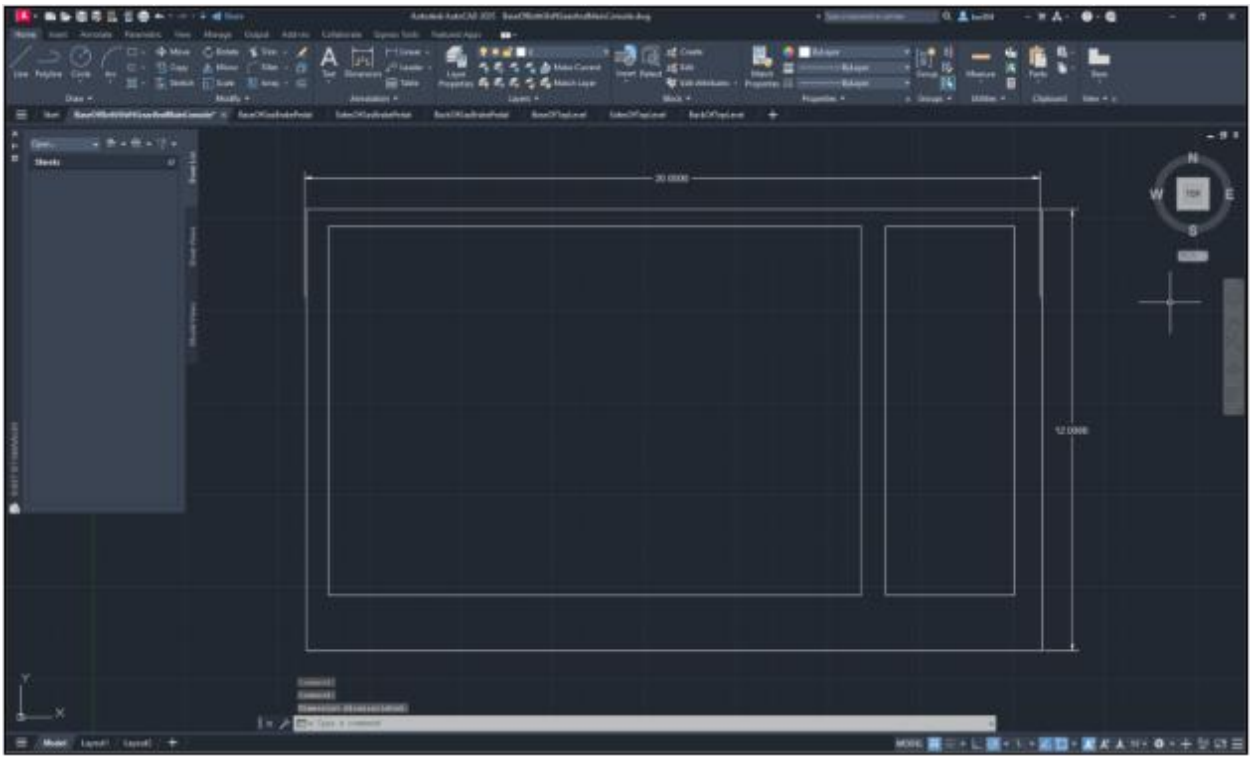


Figure [10]: Design for the Base of Entire Structure

In the screenshot above, you can see the enclosure designed for the entire design. The left portion of the design will be a little divot that will rest the main base with the steering wheel on top of it. The right portion of the design will rest the shift gear we are implementing, which will be screwed into the Plywood.

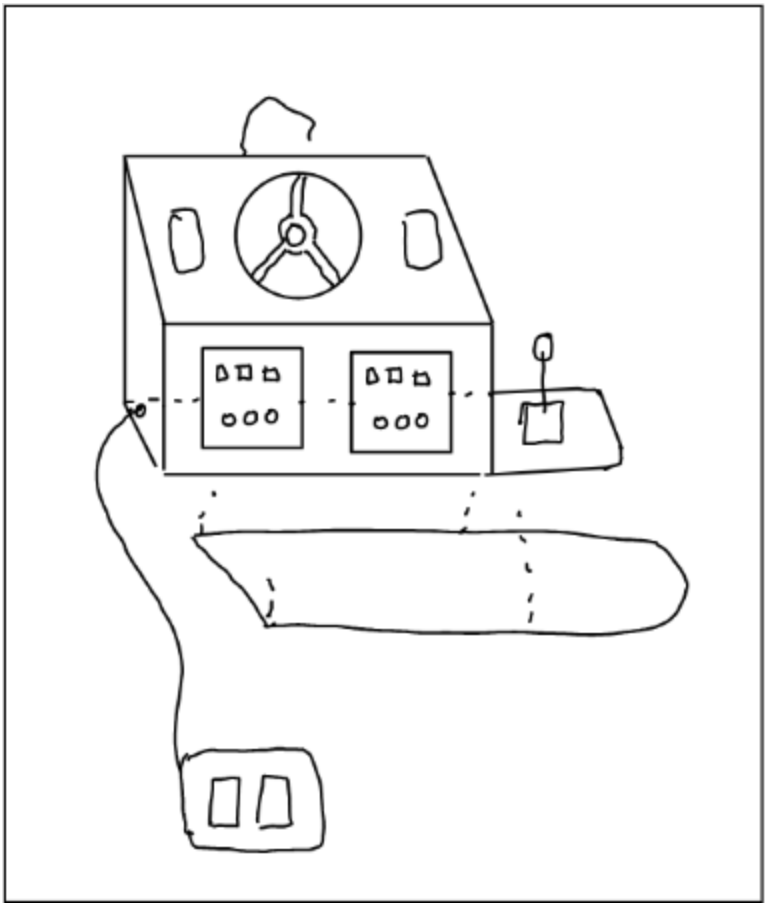


Figure [11]: Overall Design of the System

In the screenshot above, you can see the overall design of our system. The top portion of the image shows the main section of our design, which houses the steering wheel (in a slanted surface as mentioned above) and the shift gear, which will the screwed into the overall base that will be using. The overall base is depicted in the middle of the image having dotted lines drawn to it. The bottom of the image shows a 2D drawing of the gas and brake pedal enclosure we are building.

Group Name: Delta

3-D Model File Images/Description of the Enclosure Design:

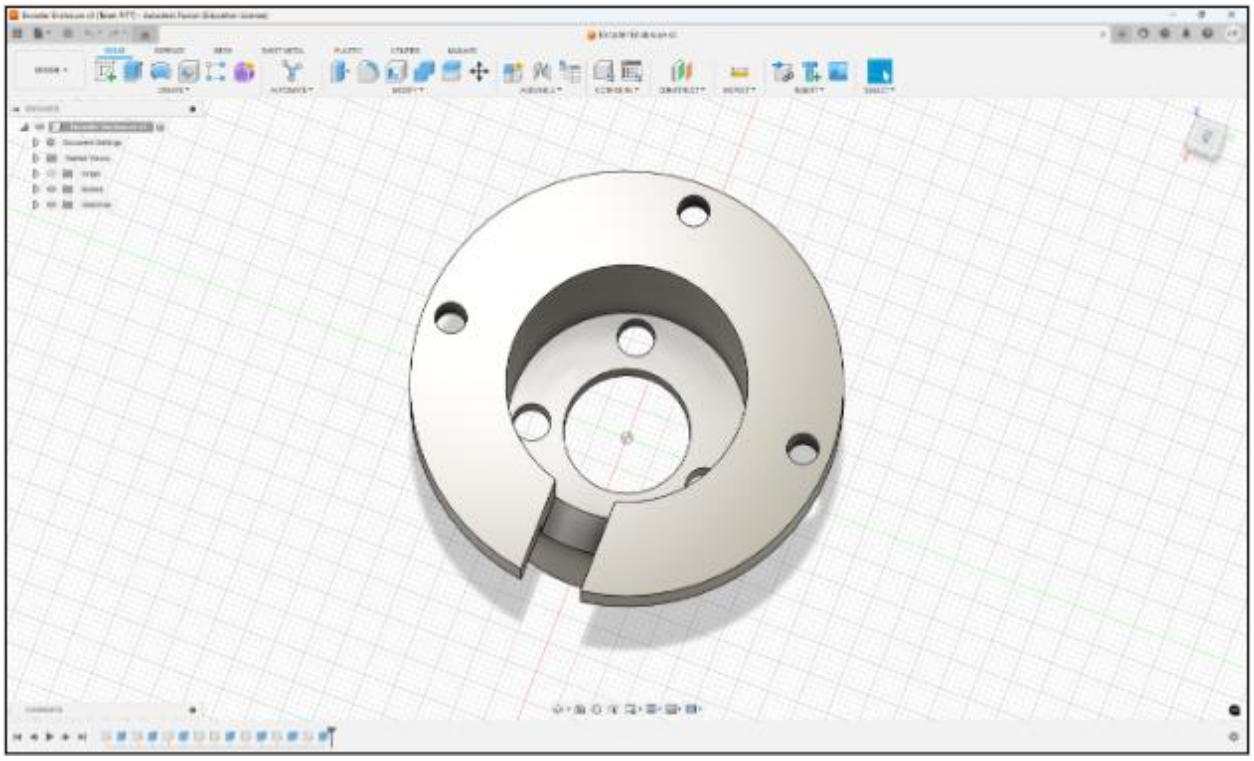


Figure [12]: Rotary Encoder Enclosure

In the screenshot above, you can see the enclosure designed for the Rotary Encoder, which will be used to mount the encoder on our Plywood enclosure. This model will help us to securely place the rotary encoder next to the gear of the Steering Wheel to ensure a clean connection between the Steering Wheel and the Rotary Encoder, which will make the steering much smoother.

Group Name: Delta

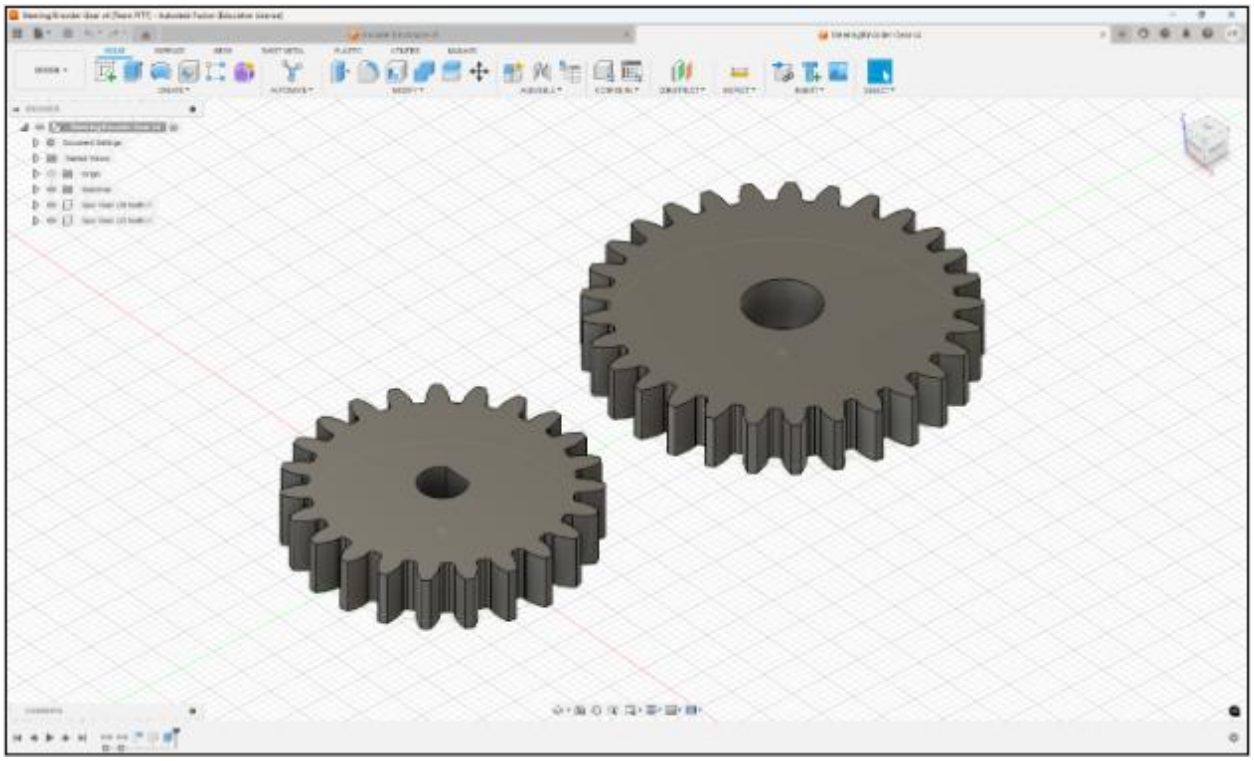


Figure [13]: Rotary Encoder and Steering Wheel Gears

In the screenshot above, you can see the gears I will be using for the Rotary Encoder (Left) and the Steering Wheel (Right). The hole for the Encoder has a flat top for the shaft of the encoder to perfectly fit on it; thus, I had to make the hole of the gear the same shape as the shaft. Also, I chose the number of teeth for each with a method such that the number of teeth on one gear is not a multiple of each other, so that the same teeth won't touch and lead to easy wear.

Group Name: Delta

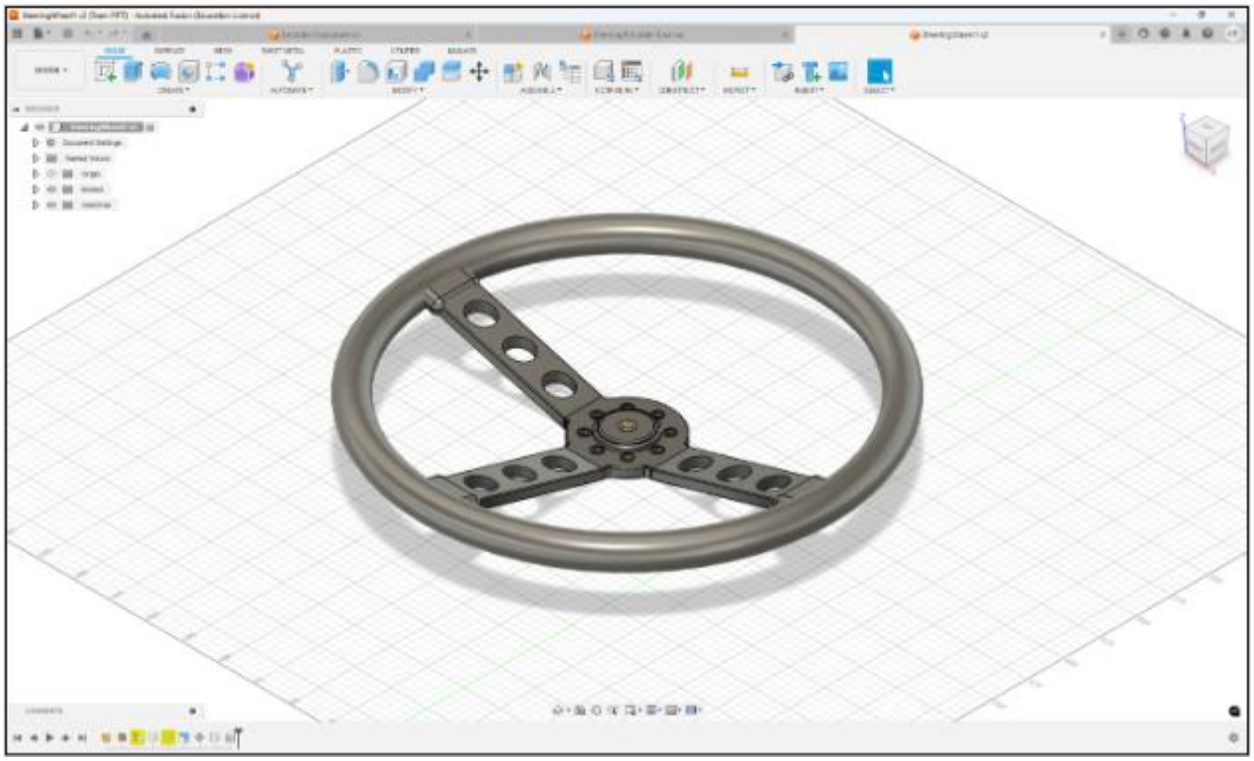


Figure [14]: Steering Wheel Model

In the screenshot above, you can see the Steering Wheel that will be used in our Kart-It game. We got this wheel from a reference model gotten from thinkiverse and we just added a hole in the middle for the bolt to go through. We will attach the bolt to a pair of washers along with a bearing in the middle of them to help with wheel movement. The end of the bolt with has a gear on it, which would be attached to the gear for the rotary encoder. This would allow for easy movement between the steering wheel and the encoder.

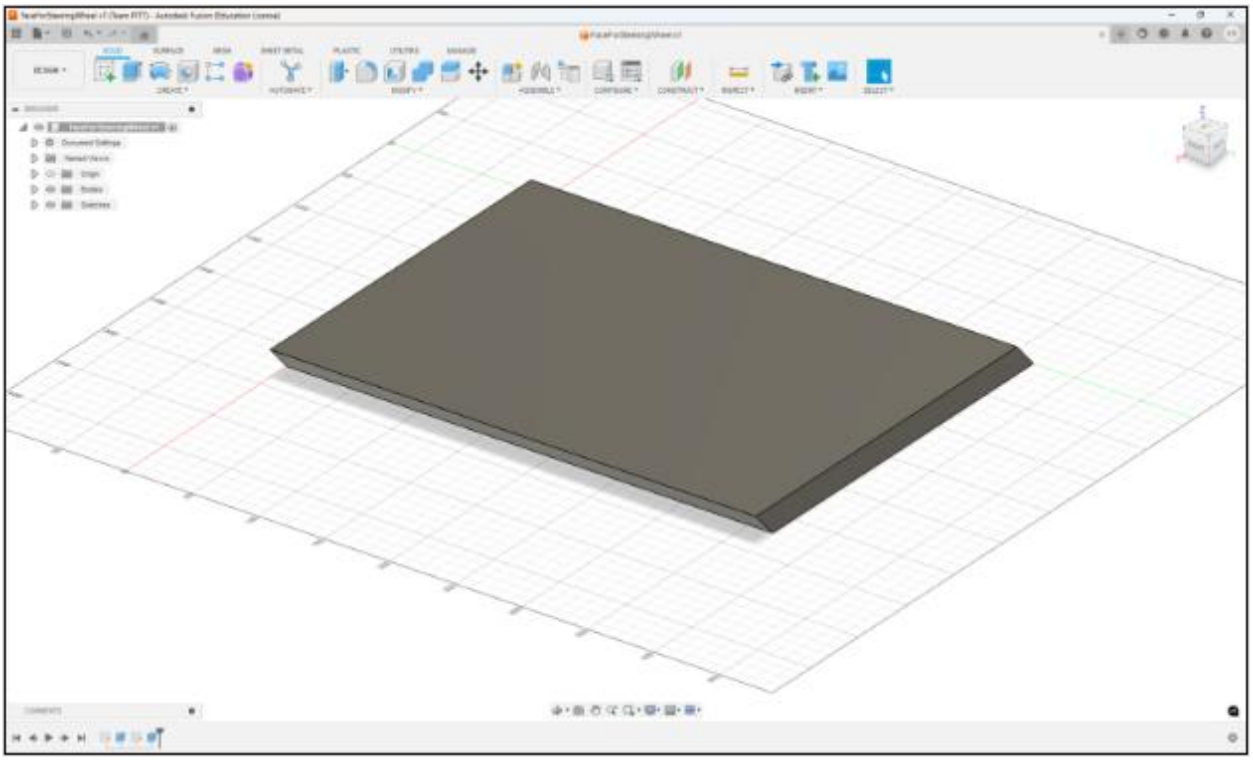


Figure [15]: Top Slanted Face for Steering Wheel

In the screenshot above, you can see the model we are going to use the CNC machine for the cut the part of the Plywood enclosure that will have the Steering Wheel attached to it. We plan to have the wheel a slanted a little up so that it is not directly facing the user and it being uneasy for the user to use. We will import the stl of the model above in Easel, which is the software used on the CNC machine in the Makerspace.

Electronic Design Implementation:

PCB Schematic

Group Name: Delta

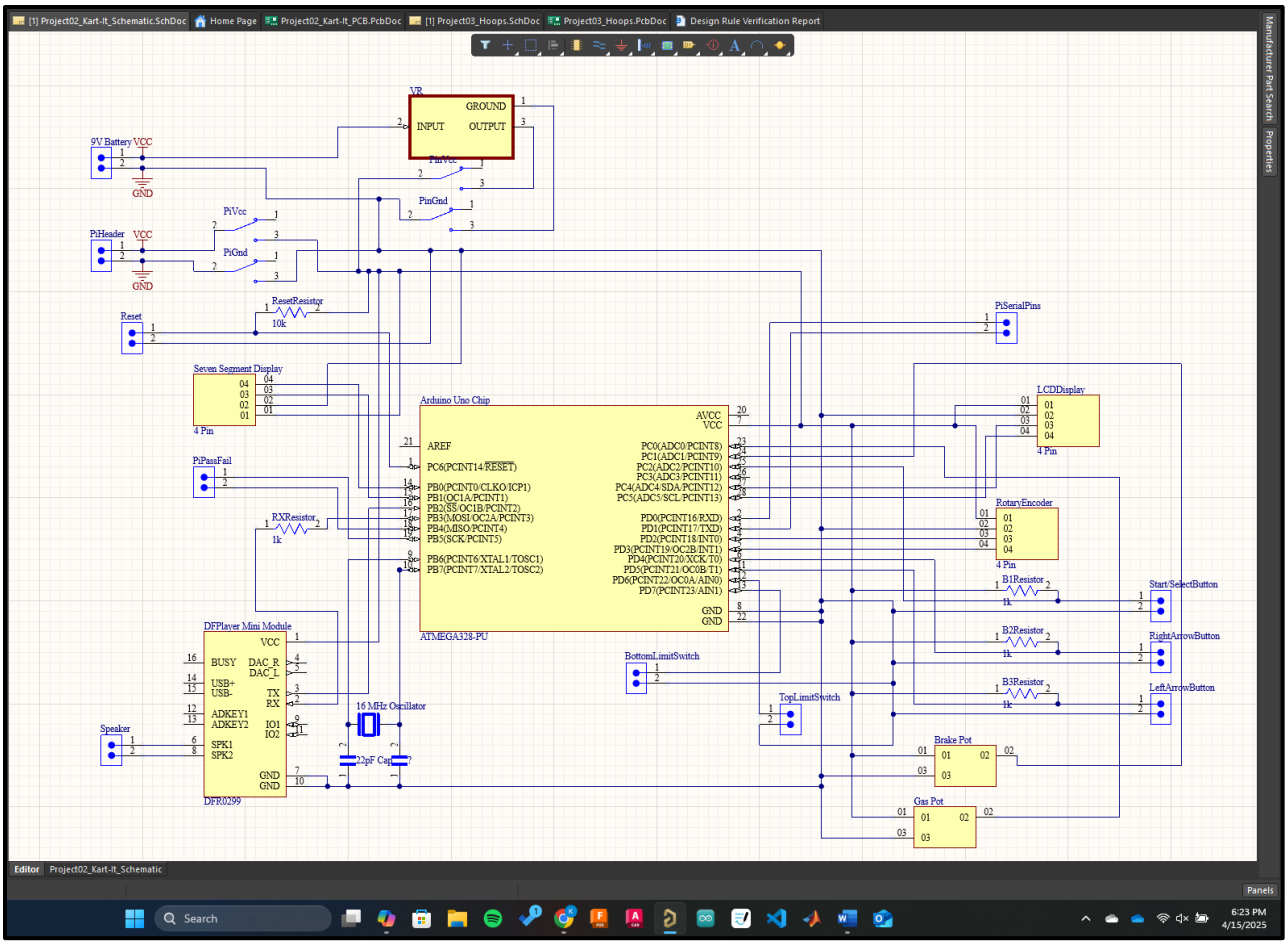


Figure [10]: Project 02 Schematic in Altium

The figure above shows the schematic for the project, which is similar to the initial proposed schematic in Figure 7. There was one change made to the Schematic shown above from Altium compared to the initial schematic. We added four switches to the schematic in order to allow the PCB to be powered by a 9V battery and the Raspberry Pi as well. So, the switches where the power is coming from are switched on before turning on the PCB. We used screw blocks for only of our components except for the ATMega chip, the DFPlayer Mini module, the oscillator, the resistors, the capacitors, and the voltage regulator. Every other component in the schematic above uses a pin headers footprint, since we will be using screw blocks for easy wire placement. From this we plan on making a 3D model of a 9V battery holder that will be attached to the bottom of the PCB to hold the battery, so that there is not a big battery dangling from the PCB. This will make the product easier to manage, since the battery has a little slot it can go in. We had to use a 2-pin screw terminal to power the circuit, since we are designing a PCB and we are unable to stick wires in a PCB for power like we can for a breadboard design. We used all THT components to assemble our schematic,

which included the ATMega328P chip, the DFPlayer Mini, the voltage regulator, the oscillator, resistors, capacitors, 4 switches, 5 resistors, and 16 screw terminals. Since our PCB was going to be a THT board, we had to use THT components to create the schematic as well. As mentioned above, the initial schematic having the pin assignments already on it helped us create the schematic and connect the corresponding wires. The oscillators and capacitors on the ATMega chip are used in order to generate and stabilize the clock signal for its operation. The purpose for all of these components and why they are used are mentioned above in the design implementation explanation section.

PCB Design Layout

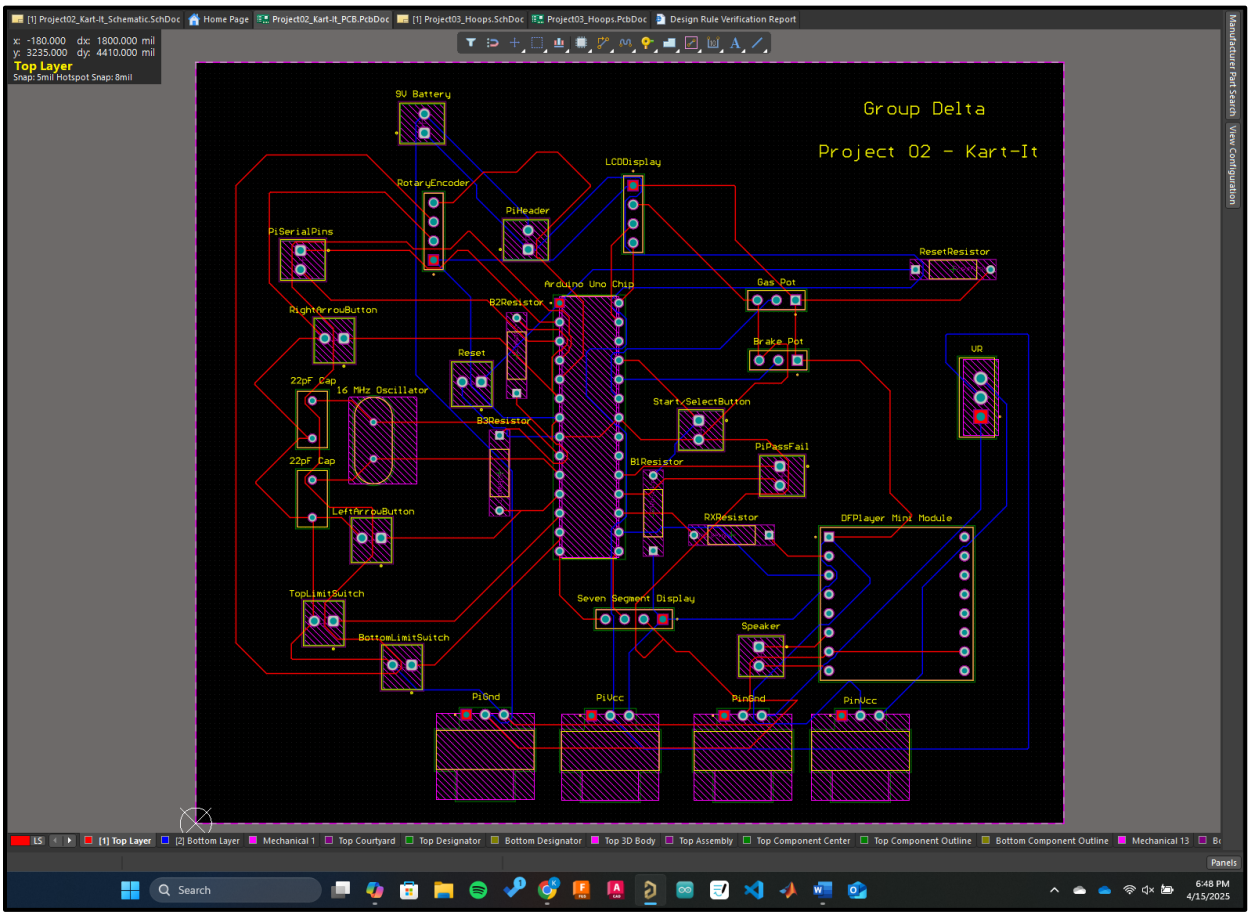


Figure [11]: 2-D View of Altium PCB Design

Group Name: Delta

The figure above represents our PCB layout for the design we are implementing. We used all THT components to assemble my schematic, which included the ATMega328P chip, the DFPlayer Mini, the voltage regulator, the oscillator, resistors, capacitors, 4 switches, 5 resistors, and 16 screw terminals. Since we were using THT components, we are going to have vias in our PCB, which allows for components to go through the PCB and makes soldering easier. In the middle of the PCB design, we decided to place the ATMega328P chip, since all of the components are connected to the chip in some way. After that, we placed all of the components around the ATMega chip, since all we cared about was placing screw blocks for all of our components and just jumping wires out of the PCB and connecting it to the component. In our PCB design, since we will be pulling out jumper wires from the PCB, we used the footprints of pin headers in order to make vias for the different components that we are connecting to the PCB. Our main goal was to get the traces between each component correct and worry about having vias for each pin of the components. So, the positions of the components are not the best, since the PCB will not be seen by anyone and will be inside of the enclosure with wires coming of it to attach to the different components. Finally, we decided to put our group name and the project name on the top of the design to make the board identifiable. Now the 2-pin header will be connected to a 9V battery, which will be stored in a 3D printed 9V battery holder. There were no challenges in the layout of the PCB, since, as mentioned above, the placement of the components was not our main concern because we are just jumping wires from it. This is mainly due to the fact our enclosure is large enough that we cannot afford to not worry about the placement of our components on the PCB. Finally, we decided to put the course name and our name on the top of the design to make our board identifiable.

Best Practice Incorporated in PCB Design: The “3W” Rule was the best practice we chose to incorporate into our PCB design, which states that the distance between traces should be at least 3 times its trace width. For example, all of our trace widths are 10mill, which means that the distance between all traces in our PCB design should be at least 30 mill. Due to the hole of the chips we are using, there are a couple of instances where the distance between the traces is 20 mill, but that only occurs a couple of times and having twice the distance of the trace width is still acceptable for my design, since it does not require high speed communication.

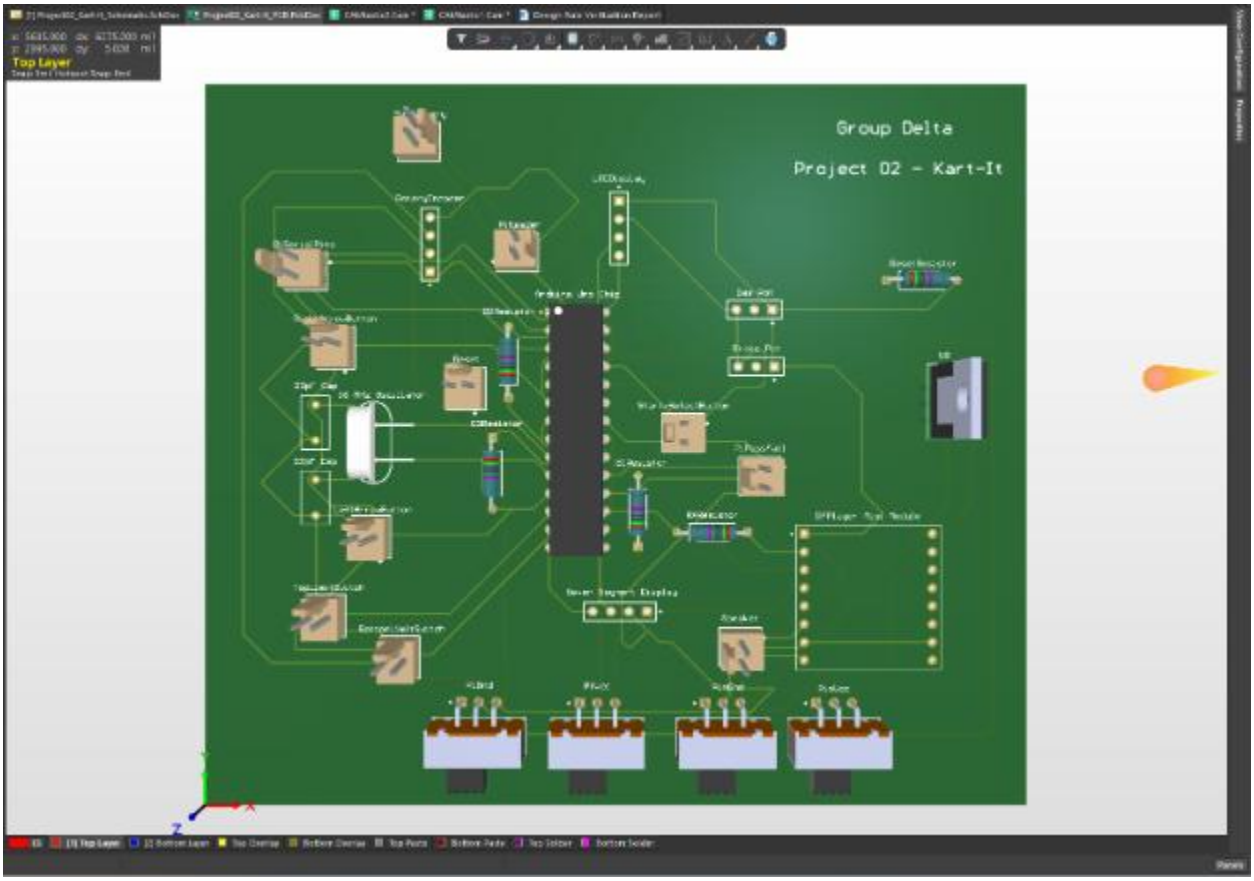


Figure 2: 3-D View of Altium PCB Design

In the figure above, the PCB design is shown in a 3-D view with all of the components inserted as if they were already soldered onto the PCB. The layout of the PCB layout is the exact same as the layout in the 2-D view of it. The oscillator component on the left side of the PCB is not depicted properly due to the footprint I downloaded from the internet, but the size and number of holes match the chip. Since what matters most in PCB design is the holes being drilled, I am not worried about the chip being sideways in the 3-D view. The light green lines shown in the PCB are the traces in the PCB, which connect the whole circuit together.

Group Name: Delta

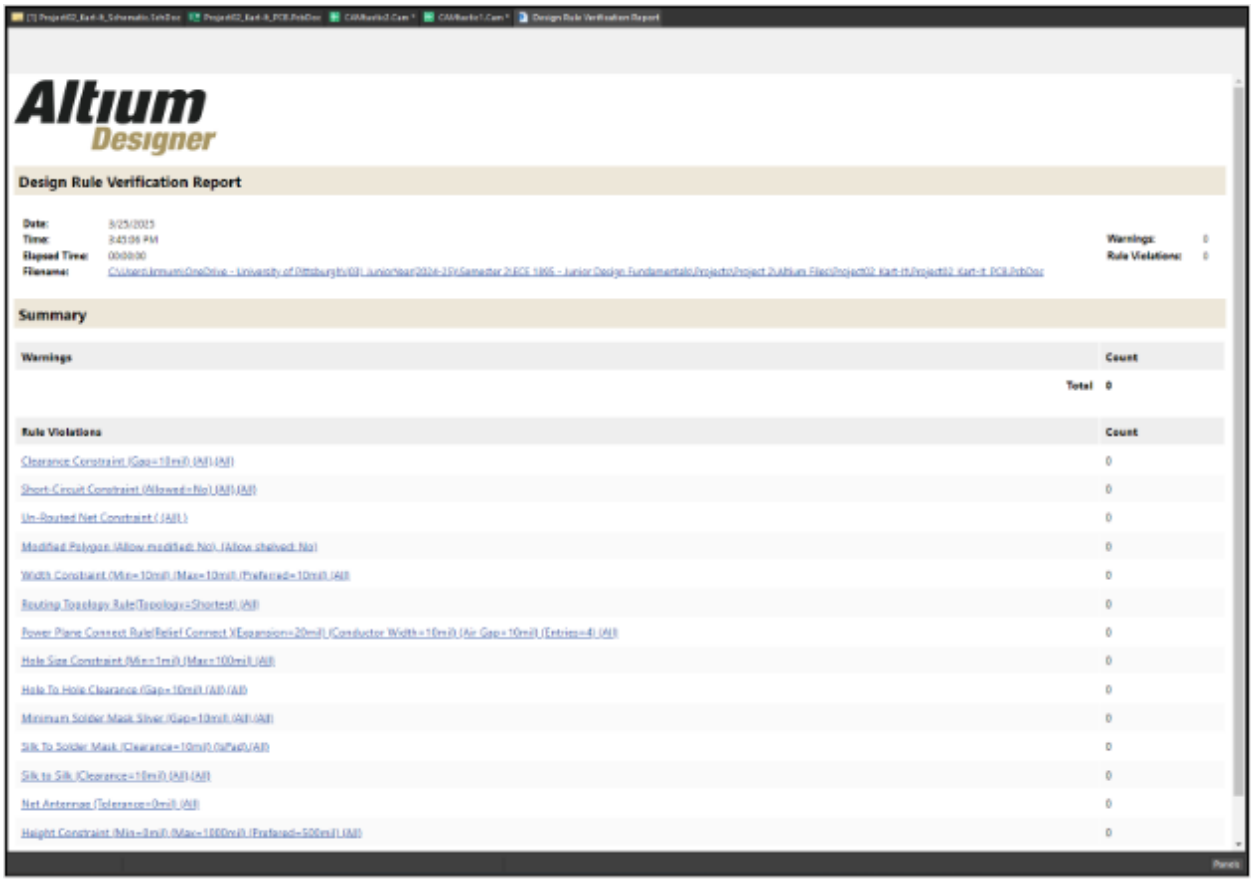


Figure 4: DRC Check Results for PCB Design

After successfully arranging the components in the PCB design, we ran the Design Ruler Check with JLCPCB Rules file to make sure that we did not get any rule violations. Initially, we did get one error because we had overlapping text, but after we fixed that, we got an error count of 0, as seen in the screenshot above. This step was one of the most important steps of the PCB design process, since it checked for any errors that may prevent the printing of a PCB. Also, it is important to make sure that this step is taken after even the slightest change done to the PCB to ensure that the board can be printed.

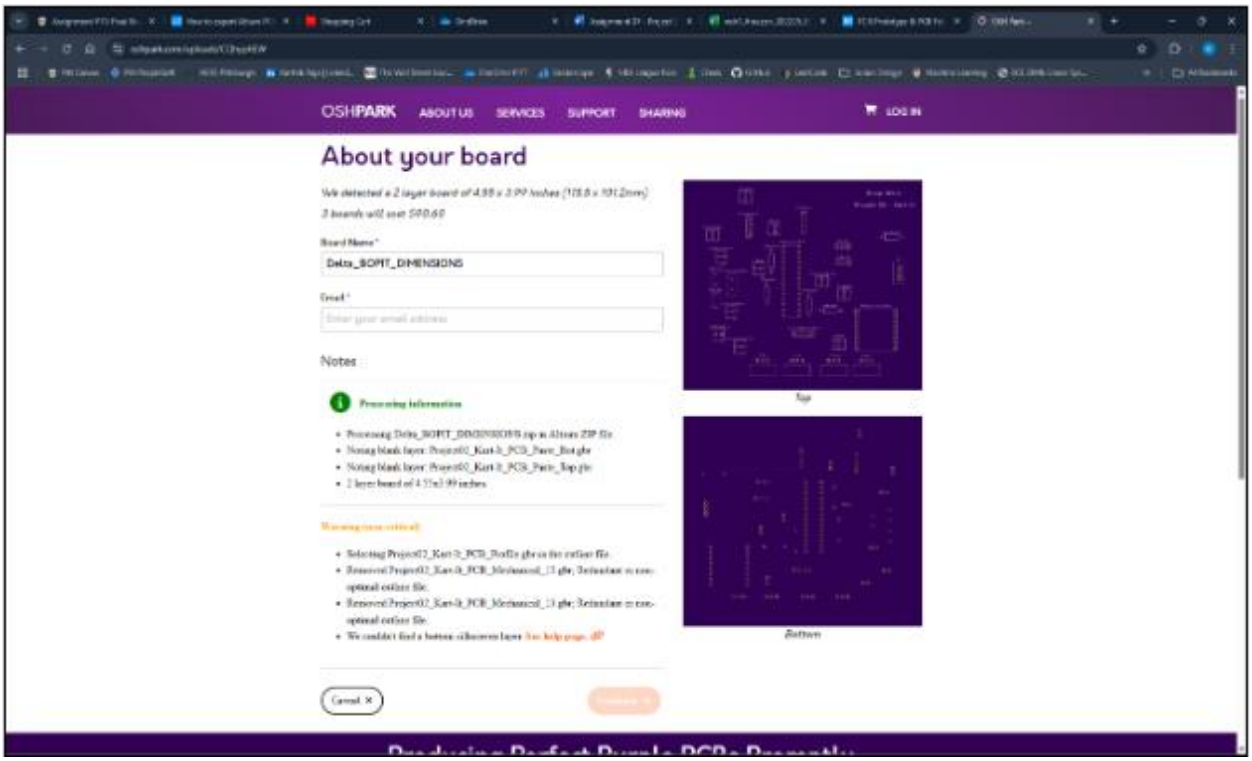


Figure 5/6: Preview of PCB Design in Manufacturer Website

As seen in the figure above, our PCB design successfully gets read by oshpark, which verifies that my design is indeed manufacturable. We uploaded our compressed 'Project Output' folder into oshpark.com and JLCPCB and the screenshot above are the pages that resulted from it. You can clearly see the top and bottom of our PCB for this project along with the traces made in the top and bottom. The design of the top and bottom layer of the PCB is the exact same layout we had above in Altium Designer. The fact that we are able to see my PCB design when we uploaded the output folder to the websites indicates that our PCB can be manufactured by that manufacturer. Hence, after finishing this step, in the PCB design process, you can go to a manufacturer's website, such as JLCPCB, to attach your Gerber file and order a PCB from them.

Software Implementation: **Link to Project GitHub**

The software implementation for this project included making separate implementations for the Mario Kart emulation and the implementation of the input sensor logic being used. Initially, we started working on the basic input logic, which included implementing all six commands with the three input sensors and adding logic around it in order to

meet the requirements for the system. The overall structure of the input sensor logic code was that we had a main function that initially checks for the start button being press and then allows the user to select the track via an LCD display. After the select button has been pressed again, the game will start giving the user commands to perform. For a certain time frame, the main function calls an external function made to check if the user has performed the command correctly or incorrectly. If the user does the correct response, then the score increases by one and the logic moves onto the next command. The game will also display “Correct Response” on the LCD display, and the speaker will have Mario saying “Oh Yea” indicating a good response. On the other hand, if the user does the incorrect response, then the score will reset to 0 and the LCD display will display “Game Over!” The time between tasks will also decrease as the number of sequential commands increases. This was a separate function made for the steering wheel to detect a left vs right turn that uses interrupts. This function was called every time the logic was checking for a certain input. In this logic file, on top of the freestyle mode, there will be three hardcoded Mario Kart tracks: Rainbow Road, Bowser’s Castle, and Baby Park. We used arrays of integers and indexed the array depending on the total number of points the player has. Each integer represents a different command: 1 = Turn Left, 2 = Gas It, 3 = Brake It, 4 = Shift Up, 5 = Turn Right, 6 = Shift Down. This is the implementation of the input logic file.

Now, the Raspberry Pi logic code will be integrated with the input logic sensor mentioned above. We found an open-source Mario Kart code online that we used as a base for our Raspberry Pi logic code. In order to integrate the integration between the ATMega chip and the Raspberry Pi seamless, we had to transfer the connection for the Left/Right arrows, select button, and the reset button onto the Raspberry Pi GPIO pins in order to allow the player interact with the Mario Kart emulation on the Pi. For the five different tracks on the game, we had to insert hardcoded checkpoints at certain points of the track in order to provide the player with the experience as if they were playing the game. We also added a freestyle game mode on the Pi as well in order to meet the requirements of having a randomly generated user commands for the user to try and perform. The Mario Kart emulation will automatically start loading on the Raspberry Pi as soon as the entire system has been turned on by the rocker switch. The gameplay will navigate itself around the track while the only job of the user is to perform the presented command on the Pi. The Pi communicated to the ATMega chip via hardware serial and it uses two digital pins of the ATMega chip. One pin is to indicate that the user correctly the command correctly, while the other pin is to indicate that the user did not do the command correctly. The logic that the ATMega runs depending on these values is the exact same as the logic explained above.

Some challenges that we encountered while implementing the software for the included running out of memory to flash to the ATMega chip. We had to reduce the amount of hardcoded tracks we have on the ATMega chip. As you can guess, implementing the Mario Kart emulation on the Raspberry Pi and communicating it with the ATMega chip was not an easy task as well, which involved many different challenges. These challenges included putting hardcoded checkpoints on all of the available tracks on the game as well as communicating all of the input sensor values to the Pi and back to the ATMega chip. One of the biggest challenges we encountered was putting the wrong voltage regulator footprint on the PCB, which resulted in frying the entire PCB. We were trying to debug the fried PCB for 2 days until we decided to solder an entirely new PCB. The voltage regulator footprint that was used in the PCB has the input and ground pins flipped compared to the physical voltage regulator that we were using on the PCB. Initially, we did not realize that the feeding the board with a negative voltage would result in frying the entire PCB; however, after hours of debugging, we realized that it was the main issue that caused everything else to stop working. Once we soldering on the voltage regulator as it was supposed to be, the PCB worked seamlessly. The GitHub link above as a detailed README file explaining what each files does.

Enclosure Designs:

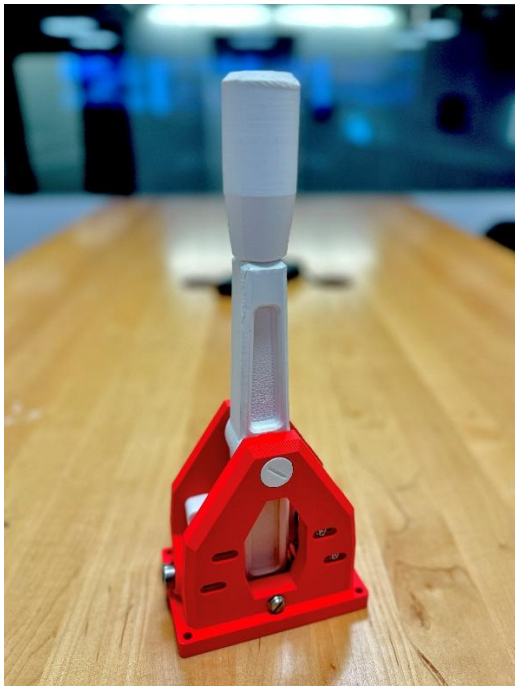
The enclosure design for this project included more than just a housing for a circuit board and is better categorized as a mechanical assembly. This section will be broken up as such to feature the design of the user input mechanisms and the overall enclosure separately.

I. User Input Mechanisms

Sequential Shifter

The sequential shifter, used for the “Shift Up” and “Shift Down” commands was build using 3-D printed parts and traditional hardware fasteners. It featured a pivoting arm about a central axis so that arm was restricted to move only forward and back. This mimicked the actual feel of a sequential shifter found in some manual transmission cars. We did not design the CAD files for many of the 3-D printed parts. Rather, the design credit belongs to Beavis Motorsports. Some alterations were made to certain parts, such as the shifter knob and limit switch mounts, to better suit the needs of our project.

Group Name: Delta



The sequential shifter operated using two limit switches to detect a shift up or down. M12 bolts provided hard stops for the shifter arm so that the limit switches were not damaged. Two springs kept the arm in the middle state, as seen above, after each user input.

Pedals

To allow for two more user commands, “Gas It” and “Break It,” two driving pedals were needed. Like the shifter, the design files for the pedals were found online (credit b505d). Each pedal used a potentiometer to collect an analog voltage reading depending on how far the pedal was depressed. Springs allowed the pedals to return to their original state.

Group Name: Delta

Figure

Steering Wheel

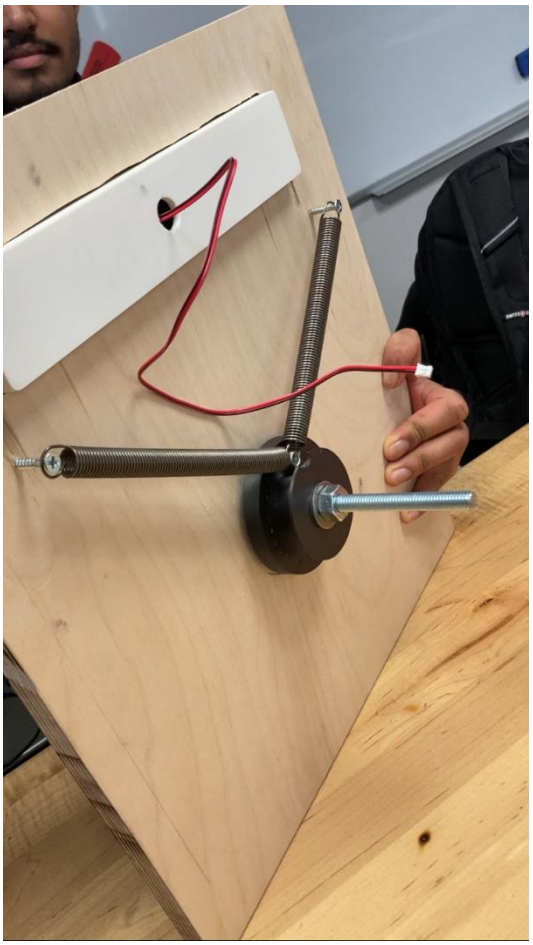
Finally, the steering wheel allowed users to turn left and right in response to these commands. The wheel itself, bearing, and hardware was purchased from Amazon and McMaster-Carr. However, there were several 3-D printed parts that had to be customized to accommodate the desired wheel behavior.

Since we wanted the wheel to be low friction, the wheel shaft was mounted on a roller bearing. The wheel shaft was a fully threaded 3/8” bolt, and it was secured to the wheel first with a nut. Then, the bolt was fed into the bearing, which was press fit into a 3-D printed housing that could be screwed to the enclosure. A nut was threaded on the backside to secure the bolt tightly to the inner race of the bearing, allowing it to spin freely.

Group Name: Delta

The next mechanism to implement was the self-returning feature of the wheel. Two tension springs were used for this application. One end of the each spring was secured to the enclosure, and the other end was looped around a small bolt on a 3-D printed wheel. This wheel was secured to the shaft of the steering wheel, so that when the steering wheel would turn, so would the spring guide wheel. The rotation would tension one of the springs, providing a pulling force to return the steering wheel to center.

Group Name: Delta

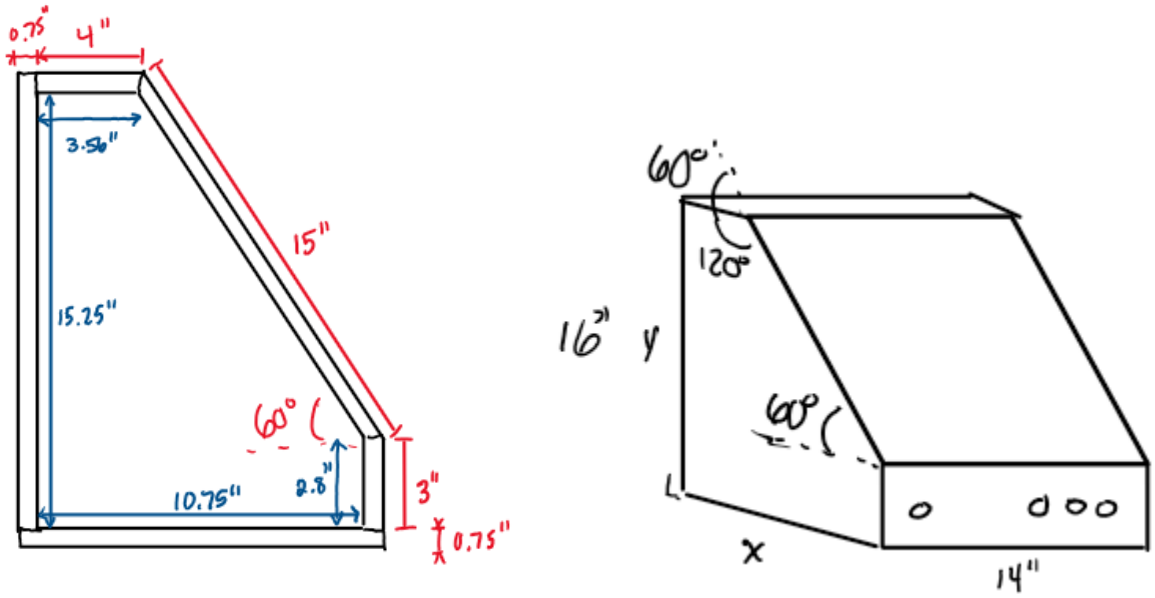


Finally, the steering wheel actually detected an input via a rotary encoder. The encoder had a gear mounted to its shaft that meshed with a gear mounded to the steering wheel shaft. This meant that the encoder shaft turned whenever the steering wheel turned. The encoder was mounted to the enclosure using a 3-D printed housing.

Group Name: Delta

II. Enclosure Design

We wanted our enclosure to be sturdy enough to support the weight of our mechanical components and the forces a user might input on the device when playing, which led us to use plywood. The enclosure was designed to have a slight upslope on the front face for ergonomic reasons, so specific dimensions had to be achieved for each of the 6 panels.



A CNC router was the tool of choice to cut these panels since it was capable of cutting through the thick plywood and could create pockets and miter angles that we planned for in the design. An STL file was created for each panel to account for any 3-D features, and they were sent to the CNC to be cut out. Unfortunately, the CNC machine we used did not accurately cut the pockets to the dimensions specified, so some work had to be done by hand with chisels and sandpaper to get the desired fit for our different components.

Assembly and System Integration:

I. Enclosure Assembly

Once the panels were cut out and roughly to shape, holes were drilled in the bottom, front, middle, and top faces to account for wood screws during assembly. The miter joint angles were also cut using a bandsaw and angle jig. When the panels could fit together properly, wood screws were used to secure the panels together.

Group Name: Delta

Quite a bit of cleanup work was done using handsaws and a random orbital sander to get all the edges flush and smooth. A combination of silicone adhesive and wood filler was used to fille any gaps in the miter joints and secure the top and front panels to the main middle section. At this point, a panel was also cut out for the pedals, and the dashboard, pictured above the steering wheel, was fit into the wooden panel.

The dashboard housed the seven-segment display, speaker, and two-line LCD screen in a 3-D printed part. A white acrylic cover was laser cut and secured with screws flush to the face of the plywood.

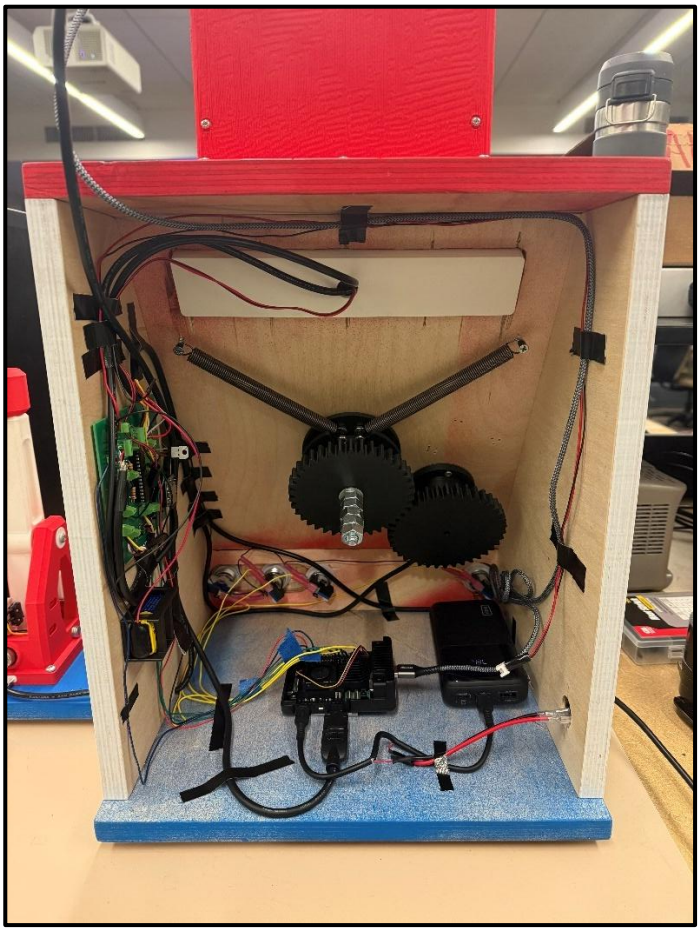
At this point, the panels were disassembled and spraypainted. The final assembly is pictured below. A 7-inch LCD screen was secured to the enclosure with a custom 3-D printed enclosure, which displays the emulated Mario Kart game.

After all of the parts were assembled together, we spray painted the enclosure to make it look more Mario Kart themed. The figure above shows all of the components of the game together.

II. Software/System Integration

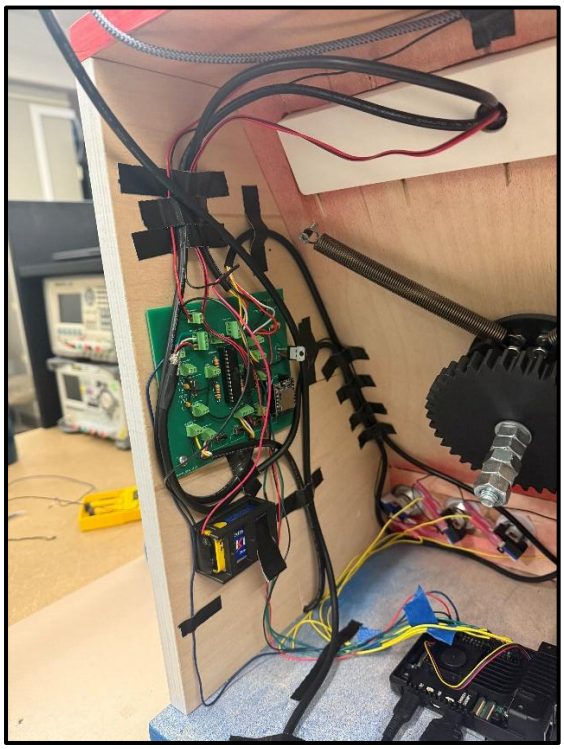
As mentioned above, the integration of the software aspect of the game included integrating the logic sensor logic with the Raspberry Pi emulator logic file, which checks to see if the current command was done correctly or not by the user. We integrated the Race_It_Logic.ino file with the Raspberry Pi pass/fail logic and got the resulting file, Kart-It_atmega328p_code.ino, that contains the final logic that was burned onto the ATMega chip. There was screw blocks placed on the PCB in order to jump wires for the RX communication, Pass/Fail pins, and the Power/Ground pins as well to make the integration between the Pi and the ATMega chip clean. The ATMega chip is being powered on by the 5V output voltage from the Raspberry Pi. After the implementation of the Mario Kart emulator is done, integrating it with the ATMega chip initial logic was not too difficult as the only difference needed was the track select and the Pass/Fail communication to differentiate good vs bad user commands.

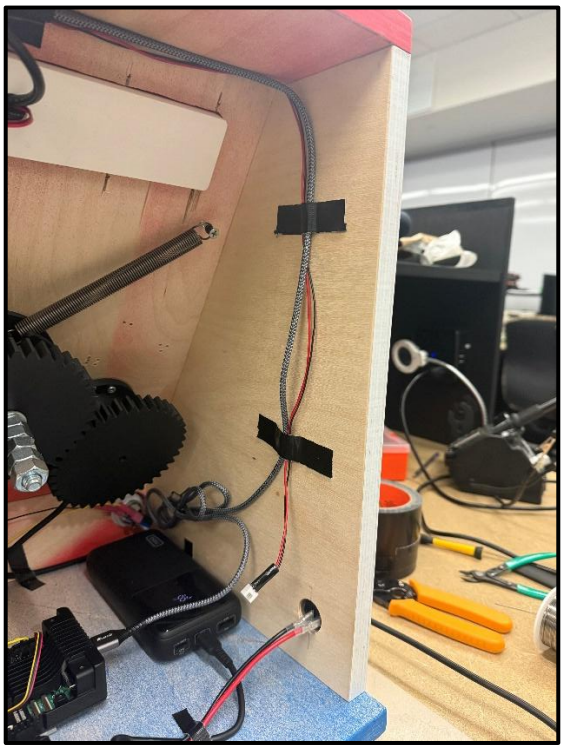
Group Name: Delta



During the assembly of our product, we decided to place the PCB on the side of the enclosure and place the Raspberry Pi and the power bank on the floor of the enclosure. In our system, the power bank is powering on the Raspberry Pi and the Raspberry Pi is powering on the ATMega chip. Wire management, such as taping down the wires across the inside of our enclosure and shortening wires to make it the perfect length, is an important part of our assembly as it would eventually lead to easy debugging scenarios. It will be easier to see where the wires go and come from. We used spade connectors and shrink tubing in order to connect the buttons to their corresponding screw block on the PCB. The assembly of the steering column was mentioned above.

Group Name: Delta



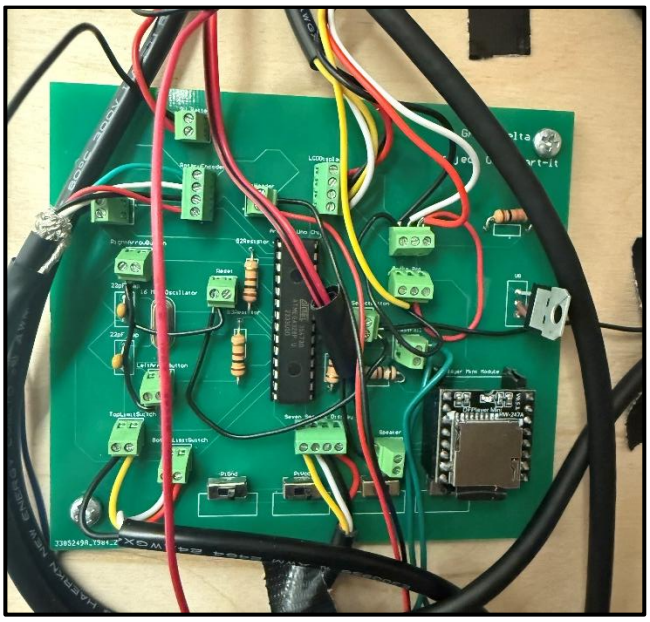


As mentioned above, the PCB was mounted on the side of the enclosure for easy access in case we needed to debug something or add some wires onto the screw blocks. The 9V battery is there only because we needed to test if the input sensor logic works before integrating with the Raspberry Pi logic for the final product. We used springs in order to implement a return mechanism for the wheel that will help the user come back to the origin. Also, we added multiple nuts to the bolt as the steering column was coming out loose sometime when the wheel was turned too hard; thus, eliminating that problem. As mentioned above, the two meshed gears are there to turn the rotary encoder as the steering wheel is turned by the player.

Group Name: Delta

Design Testing:

Images/Description of the Assembled PCB:



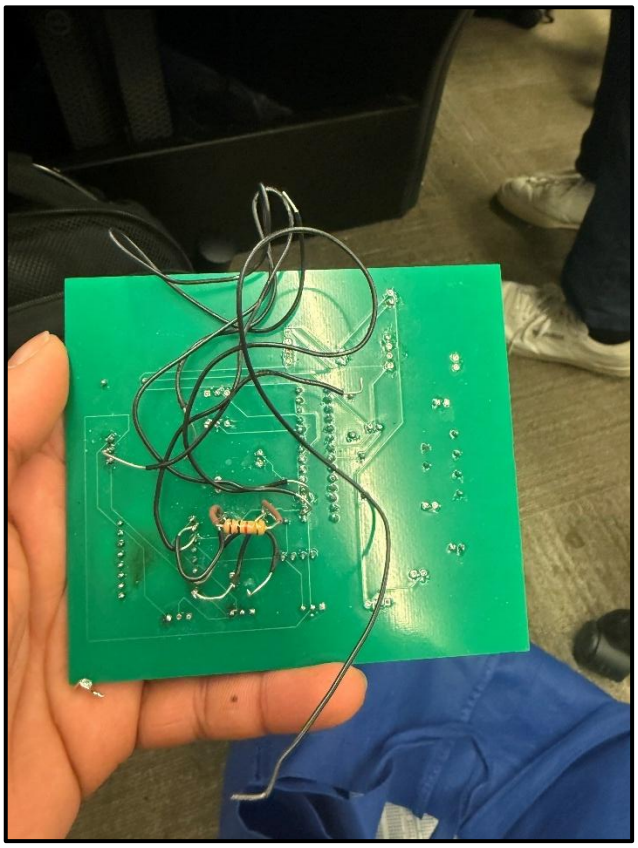
The figure above shows the top view of the assembled PCB for Project 02. As mentioned above, I used all THT components to assemble my PCB, which included the ATMega328P chip, the DFPlayer Mini, the voltage regulator, the oscillator, resistors, capacitors, 4 switches, 5 resistors, and 16 screw terminals. Since we were using THT components, we are going to have vias in our PCB, which allows for components to go through the PCB and makes soldering easier. In the middle of the PCB design, we decided to place the ATMega328P chip, since all of the components are connected to the chip in some way. After that, we placed all of the components around the ATMega chip, since all we cared about was placing screw blocks for all of our components and just jumping wires out of the PCB and connecting it to the component. In our PCB design, since we will be pulling out jumper wires from the PCB, we used the footprints of pin headers in order to make vias for the different components that we are connecting to the PCB. Our main goal was to get the traces between each component correct and worry about having vias for each pin of the components. So, the positions of the components are not the best, since the PCB will not be seen by anyone and will be inside of the enclosure with wires coming of it to attach to the different components. Finally, we decided to put our group name and the project name on the top of the design to make the board identifiable. Now the 2-pin header will be connected to a 9V battery, which will be stored in a 3D printed 9V battery holder. There were no challenges in the layout of the PCB, since, as mentioned above, the placement of the components was not out main concern

because we are just jumping wires from it. This is mainly due to the fact our enclosure is large enough that we can afford to not worry about the placement of our components on the PCB. Finally, we decided to put the course name and our name on the top of the design to make our board identifiable.

Best Practice Incorporated in PCB Design: The "3W" Rule was the best practice I chose to incorporate into my PCB design, which states that the distance between traces should be at least 3 times its trace width. For example, all of my trace widths are 10mill, which means that the distance between all traces in my PCB design should be at least 30 mill. Due to the distance between holes of the same chips, there are a couple of instances where the distance between the traces is 20 mill, but that occurred 2 times and having twice the distance of the trace width is still acceptable for my design, since it does not require high speed communication.

The debugging process we used to test the PCB and the entire system included using continuity testing to make sure that the connections between different components on the PCB were intact and functioning as expected. This process ensured that there were no breaks in the circuit pathways and verified the integrity of soldered joints. We also tested the voltage between various key points on the circuit to ensure that all components were receiving the correct voltage levels as per the design specifications. This was critical to identifying any potential faults, such as voltage drops or inconsistencies that might have caused the system to malfunction. This is what helped us identity the voltage regulator issue mentioned above.

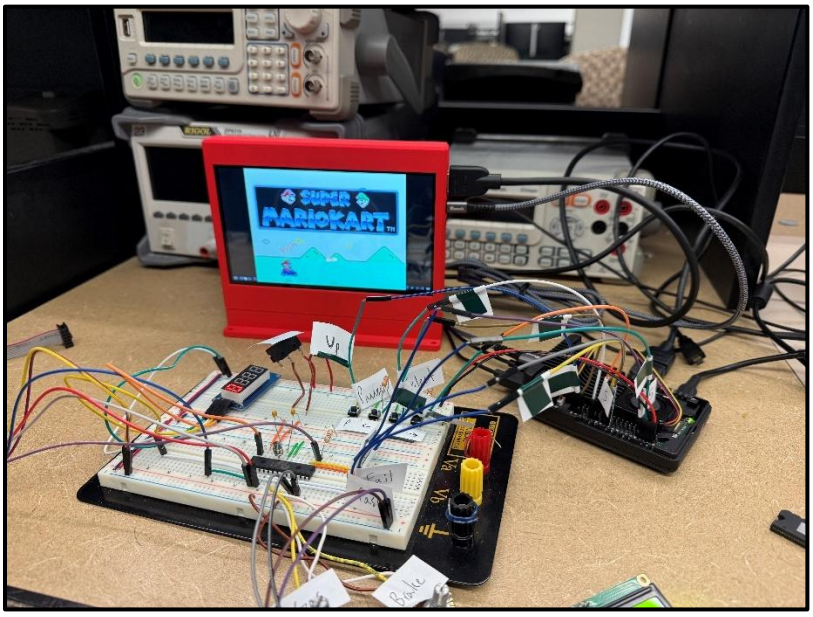
Group Name: Delta



The figure above shows the initial PCB that was fried by the voltage regulator mishap and us thinking that the PCB traces were not correct. We had to cut the traces and jump wires from one joint to the another joint. However, all of this was not needed, since we realized that the voltage regulator was not the correct one and just decided to solder another PCB. But there are no traces cut nor are there any wires jumping from one joint to another on the PCB that we used for the final enclosure. The original PCB design was correct from the beginning.

We were testing the newly soldered PCB by connecting each of the components to the PCB one by one and running the helper functions made for each component to check if it is operating as expected. First, we tested the LCD display and the seven-segment display. Both of those components were working perfectly fine. Then, we moved onto the DFPlayer Mini module. The DFPlayer Mini is another module that gave us a lot of trouble due to the fact that it needed to be reset in order to be perform as expected. We always needed to press reset first in order to make the DFPlayer Mini functional. The solution to this was changing the Software Serial library we were using. We were using the DFRobotDFPlayer library, while the correct library to use was NeoSWSerial. The new library fixed the

timing issue and made the DFPlayer operational on the first try. Furthermore, we decided to test the limit switches placed in the sequential shifter. After successfully testing this, we went to test the potentiometers placed in the gas and brake pedals. We had to debug the potentiometers in order to change the threshold in which it detects a gas or brake press. Finally, we went onto test the rotary encoder to make sure it can distinguish between left and right turns.



The way we tested the Mario Kart emulator was we connected the Raspberry Pi to a breadboard level prototype of a separate ATMega chip. We labeled each of the jumper wires with the corresponding action to not get mixed up with anything. We set up the breadboard prototype with the input sensors being jumped from the same pins as the original schematic, so that once the testing of the Raspberry Pi is done, we can replace the ATMega chip that is already in the PCB dip socket with the one on the picture above.

Video of PCB and Inputs Working:

Click on this Link to Access Video of Functionality

As seen by the video above, the functionality of the game works as such:

- The player must press the start game button to play the game.
- There are multiple tracks the users can select.

- If the player does the correct command, then it moves onto the next command.
- If the player does not do the correct command, then the game is over.
- The PCB is functioning as expected and the same as the breadboard prototype as well.
    - Our design is fully functional.

Video of Game working with Mario Kart emulator:

Click on this Link to Access Video of Functionality

As seen by the video above, the functionality of the game works as such:

- The player must press the start game button to play the game.
- There are multiple tracks the users can select.
- If the player does the correct command, then it moves onto the next command.
- If the player does not do the correct command, then the game is over.
- The PCB is functioning as expected and the same as the breadboard prototype as well.
    - Our design is fully functional.

Budget and Cost Analysis:

Upon careful review of the different Bill of Materials, soldering components, and plywood used for the enclosure, the cost for our project is $622.71. The items placed in the Bill of Materials were $542.71, while the estimated cost for the plywood we used was around $50.00. I also estimated that we used around $30.00 of cost associated with assembly

(i.e. soldering components on to the board). The estimated cost of producing 10,000 copies of our design would come out to $6,227,100.

Team Member and Contributions:

1. William Roper – Enclosure, PCB, and CAD
2. Zachary McPherson – Mario Kart Emulator Integration with Initial Input Logic and CAD
3. Karthik Raja – PCB, Initial Input Logic, and CAD

We decided to divide the project and let everyone focus on what they do best and are most interested in. We made this decision because when people work on something they enjoy, it's more likely to be completed successfully. We met at least 3 times per week outside of class time and dedicated a few hours per meeting to solely work on our project. All team members attended and contributed to all of the meetings. For communication, we used text messages.

Timeline:

1. **Week 1**
    a. Finalize overall design and gathering required materials
2. **Week 2**
    a. Finalize Initial Prototype with all sensor logic working
    b. Start 3D printing
3. **Week 3**
    a. Finalize PCB Design
    b. Finalize 3D prints
4. **Week 4**
    a. PCB Assembly
    b. Cutting out and Assembling Enclosure

c. Finalize Mario Kart Emulation

5. **Week 5**

a. Assembling/Integrate all components together and testing/debugging

b. Presentation/Inspection

Summary, Conclusions, and Future Work:

One of the most important things we learned throughout this project was that it is important to breadboard designs exactly as it is on the PCB schematic. Initially, we prototyped using the Arduino Uno dev board. This was one of the mistakes we had made due to the fact that the DFPlayer Mini was working on the dev board; however, it was not operating on the PCB that just used the ATMega 328 chip. Another important thing we learned was that there are many different types of voltage regulators and not all of them have the same pinouts. If we had used the correct footprint for the desired voltage regulator, we could have saved multiple hours of debugging.

Overall, our project is functioning exactly how it was intended to per our initial proposal. We did not have the Mario Kart emulator planned on during our initial proposal; however, we decided to have one and it be functional alongside the input sensors.

If we were to iterate upon the current design, we would make one PCB that has the ATMega chip and the Raspberry Pi, and we would have traces going from the Pi to the ATMega chip, vice versa, instead of the jumper wires we have inside of the enclosure. Another thing we would improve is the power bank that is powering the Raspberry Pi. Initially, we had thought to have an auto-on power bank, where it will turn on whenever it detects a load. Currently, the power bank needs to be manually turned on to turn on the system. We would have to figure out a way to make the power bank automatically turn on when a load is detected. Finally, we would also try and add more tracks to the Mario Kart emulator in order to allow the players to have more tracks to select from while playing.