IS1901
Microcontroller Based ICT Project

Final Report

Supervisor :- Mr. B. H. Sudantha

Group No : 14 **Group Name :** Arduspark

Index Number	**Name**
235013M	Asmitha Udayachandran
235079T	Arulanantharasa Nirsanth
235037N	M.R.F.Fahma
235033A	Janaka Sachin Dissanayaka
235031P	W.P Imasha Dilshani

Bachelor of Science Honours in Information Technology & Management (BScHons (IT&M))
Faculty of Information Technology
University of Moratuwa

Table of Contents

1. Introduction

The thrill of competitive tabletop games like football has long been a favorite among players of all ages, offering hours of fun through fast-paced, hand-controlled action. These traditional games encouraged real-time interaction, friendly rivalry, and quick reflexes. However, as technology advanced, the way we interact with games also evolved shifting from mechanical setups to intelligent, sensor-based systems that offer a new dimension of interactivity and engagement.

Inspired by the spirit of classic table football, the Robo Rail Football Game reimagines the experience using modern microcontroller-based technology. This project combines motorized kicking arms, ultrasonic sensors, and LDR-based scoring systems to create a fully automated and responsive two-player football-style game. Players control their kicking arms to score goals while defending against their opponent, all in a compact and competitive game arena.

To maintain continuous gameplay, the system features side kicking arms that activate automatically whenever the ball stops in the middle of the field kicking it back into motion without the need for manual resets. Real time scores are displayed on an LCD screen, intensifying the competitive spirit and immersing players in the action.

The Robo Rail Football Game blends the nostalgic excitement of classic table games with the interactive capabilities of today’s hardware offering a fun, fast-paced, and futuristic take on traditional gameplay.

1.1 Problem in Brief

Traditional tabletop football games, while enjoyable and competitive, rely heavily on manual interaction and offer limited automation or technological innovation. With the rise of interactive digital entertainment and sensor-driven experiences, players now seek more immersive, responsive, and dynamic gameplay that bridges the physical and digital worlds.

Existing football-style games often lack automation, real-time feedback, and adaptive features that maintain fluid gameplay. Manual resets when the ball gets stuck, absence of intelligent scoring mechanisms, and limited use of modern hardware make these games less engaging for today’s tech-savvy generation.

Therefore, there is a need for a modern, sensor-based football game that blends physical gameplay with smart technology. The Robo Rail Football Game addresses this gap by introducing a fully interactive, microcontroller-powered gaming setup featuring automated side kicking arms, LDR-based scoring, ultrasonic ball tracking, and live score display, offering players a competitive and futuristic gaming experience that redefines conventional play.

2. Literary Study

Interactive gaming platforms have evolved significantly with the integration of automation, microcontrollers, and real-time sensor-based systems. From traditional mechanical games to modern electronic alternatives, the shift toward intelligent systems has opened new possibilities for enhanced user experience, competitiveness, and engagement. This literature study explores the underlying technologies and existing applications relevant to the development of a microcontroller-based football-style game system, such as the Robo Rail Football Game.

2.1. Existing Interactive Gaming Systems

- Mechanical Football and Table Football Games: Traditional tabletop football games depend entirely on manual control and human reflexes. These systems, while entertaining, lack automation and real-time interaction capabilities, leading to limitations in scoring accuracy, game flow, and player engagement.
- Electronic Arcade Systems: Some arcade-style football games incorporate limited automation, such as goal sensors or scoreboards. However, these systems often use predefined mechanics and do not respond adaptively to the game environment, such as ball positioning or automated ball retrieval.
- Virtual Sports Simulations: With advancements in mobile and console gaming, many virtual football games now offer high levels of realism, physics-based gameplay, and AI-driven opponents. However, these experiences are confined to screens and lack the tangible, physical engagement of real-world interactions.

The Robo Rail Football Game fills a unique niche by combining the physical interactivity of traditional games with the precision and responsiveness of sensor driven automation.

2.2. Microcontroller Applications in Interactive Systems

Microcontrollers such as Arduino, serves as the core of modern automation systems in educational, industrial, and entertainment domains. Their ability to process data, control actuators, and drive real-time decision making makes them ideal for interactive gaming projects.

In the context of the Robo Rail Football Game, microcontrollers are used to:

- Control motor-driven kicking arms.
- Read input from sensors (e.g., ultrasonic for ball position, LDRs for goal detection).
- Trigger automated mechanisms like side kickers.
- Update live scores on an LCD screen.

Programming environments such as Arduino IDE or allows developers to implement complex logic for responsive and dynamic gameplay.

2.3. Sensor Technologies for Real-Time Gameplay

The success of an interactive gaming system relies heavily on accurate and responsive sensor integration. Several sensor technologies have been explored in literature and practice for similar applications:

- Ultrasonic Sensors: Widely used for distance measurement, ultrasonic sensors (e.g., HC-SR04) can detect the position of the ball and trigger events when it enters specific zones of the playfield.
- Light Dependent Resistors (LDRs): These sensors help detect changes in light levels, making them ideal for simple goal detection when the ball passes through or obstructs the sensor’s view.
- Infrared Sensors: IR sensors are commonly employed in similar systems for object detection and scoring triggers.

- Motor and Servo Control: Stepper motors and DC motors can be used for precision-controlled movements, such as activating the kicking arms or side ball retrievers. Motor drivers (e.g., TB6600, L298N) help in interfacing motors with microcontrollers.

The integration of sensors and actuators through careful calibration and real-time logic control allows for fluid, uninterrupted gameplay. This creates a system that mimics real-world reactions while maintaining fairness and competition.

3. Aims & Objectives

3.1 Aim

The primary aim of the Robo Rail Football Game is to design and develop an interactive, microcontroller-based football gameplay system that merges the fun of traditional tabletop football with the intelligence and responsiveness of modern automation. The game is built to allow two players to control motorized kicking arms to shoot and defend a ball within a compact field, while smart sensors handle scoring, ball detection, and game continuity.

The system aims to eliminate the common drawbacks found in manual tabletop games such as manual scorekeeping, ball retrieval delays, and inconsistent play by integrating LDR sensors for goal detection, ultrasonic sensors for ball tracking, and automated side-kicking arms to restart the game when the ball gets stuck.

Beyond just entertainment, the project also aims to serve as an educational tool that demonstrates core concepts in robotics, real-time systems, sensor-actuator coordination, and embedded programming. The game targets a seamless blend of physical interactivity, digital feedback, and real-time automation to deliver a unique and engaging experience for users.

3.2 Objectives

One of the key objectives is to blend traditional physical gameplay with modern embedded system technologies to offer an immersive and automated gaming experience.

- Encouraging Real-Time Interaction: Create a system where players can interact with the game in real time using mechanical arms, offering a physical gaming experience enhanced by digital feedback.
- Automated Scoring System: Use Light Dependent Resistors (LDRs) to automatically detect goals scored, minimizing human intervention and increasing the reliability and fairness of scoring.

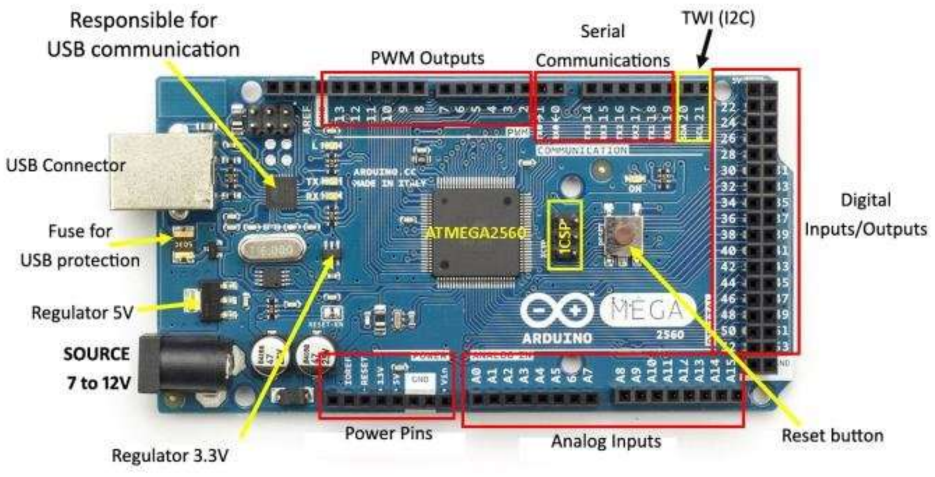
- Ball Position Tracking: Integrate ultrasonic sensors to detect when the ball is stuck in the middle zone of the field, ensuring the game remains dynamic and fast-paced.
- Automatic Ball Retrieval: Introduce side-mounted kicking arms that are automatically triggered when the ball is stationary for too long, kicking it back into play and reducing gameplay interruptions.
- Live Score Display: Use an OLED to display each player's score in real time, increasing the competitive spirit and providing instant feedback to the players.
- Stability and Accuracy in Actuation: Implement precise control over motors using reliable drivers to ensure the kicking arms and side mechanisms respond promptly and accurately during gameplay.
- User-Friendly Setup: Design the game with ease of use in mind, allowing quick setup and reset for continuous rounds of play, suitable even for users with minimal technical knowledge.

4. Analysis and Design

The Robo Rail Football Game is designed as a fully automated, sensor-integrated, microcontroller-based gaming platform. It aims to replicate a football-style two-player game using hardware components that allow for real-time interaction, responsive ball detection, automated scoring, and smooth gameplay. This section outlines the analysis and functional role of each hardware and software component used in the system.

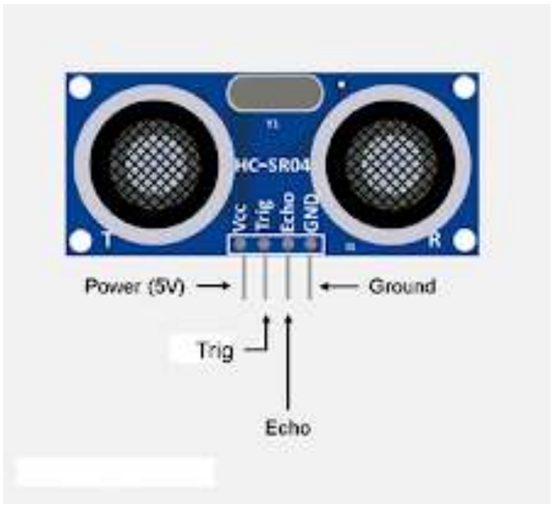
4.1 Components Required for the Proposed Solution

01. Arduino Mega 2560

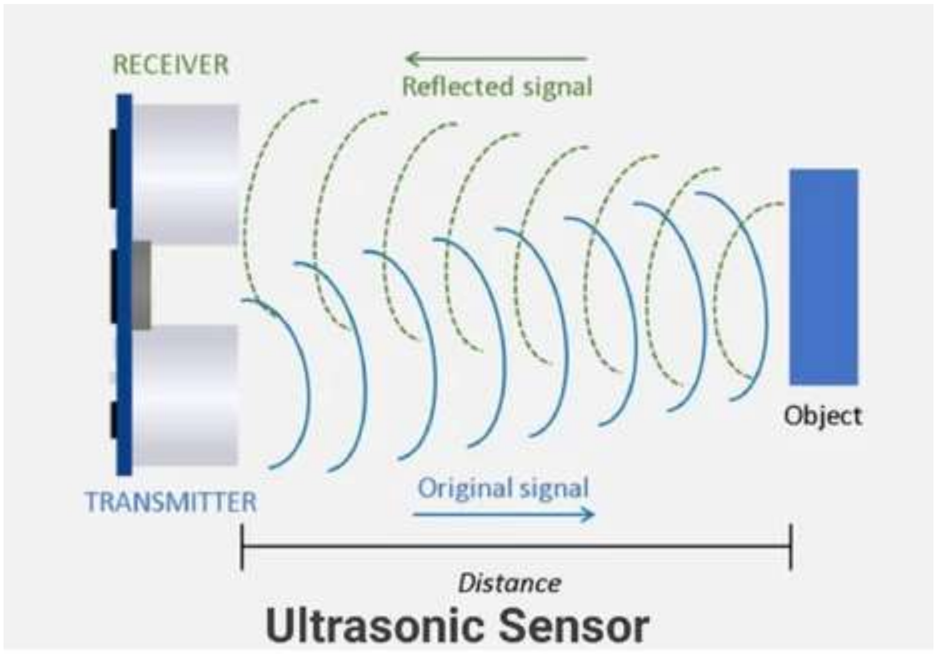


The Arduino Mega 2560 acts as the central processing unit of the entire system. With its extended number of I/O pins and high memory capacity, it effectively manages the control of multiple peripherals such as motors, sensors, and displays. It processes data from input sensors (LDRs and ultrasonic sensors) and accordingly controls the output devices like servo motors, stepper motors, and the OLED display.

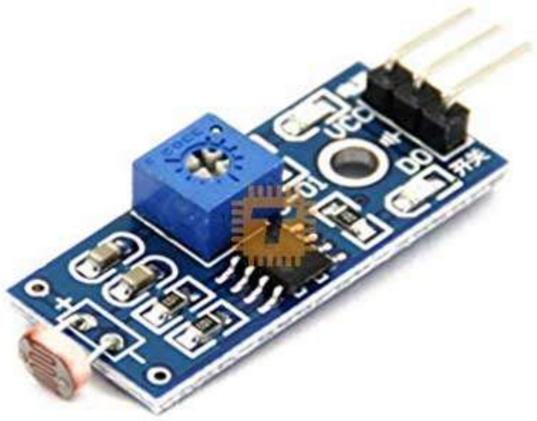
02. Ultrasonic Sensor (HC-SR04)



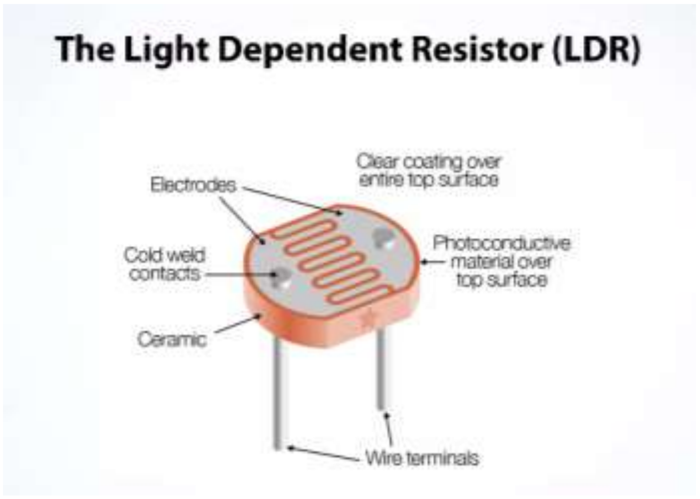
This sensor is used to track the ball's position on the board. When the ball remains in the central area for more than a defined time (e.g., 3 seconds), the ultrasonic sensor triggers the side-mounted kicking arms to hit the ball and resume gameplay. This ensures continuous motion and avoids interruptions.



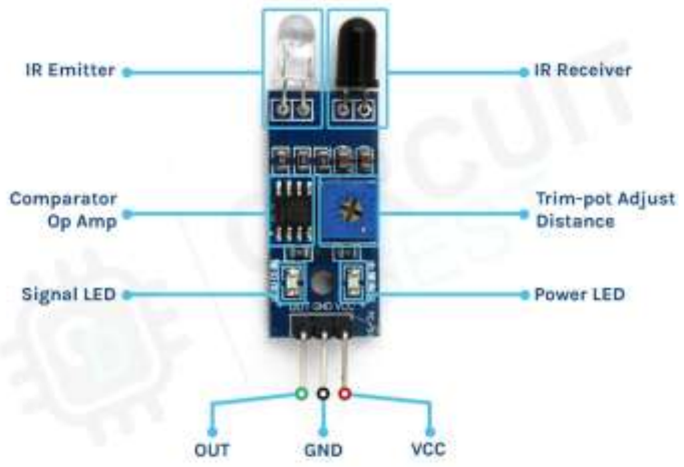
03. LDR Sensors (Light Dependent Resistors)



LDR sensors are placed at the goal areas to detect when a ball passes through. When light is blocked momentarily by the ball, a goal is registered. This method enables accurate and low-cost scoring detection without the need for physical contact.


The Light Dependent Resistor (LDR)
Electrodes
Clear coating over entire top surface
Cold weld contacts
Photoconductive material over top surface
Ceramic
Wire terminals

04. IR Sensors



In early testing or in specific detection zones, IR sensors may be used to sense ball presence. However, for scoring, LDRs are primarily preferred for simplicity and effectiveness.

05. DC Motors

DC motors are used to control horizontal movement or kicking mechanisms of the main player arms. They are suitable for fast and simple linear motion, making them ideal for reactive gameplay where rapid response is needed.

06. Servo Motors

Servo motors are used to control the side-kicking arms, providing precise rotational control needed to reposition or push the ball back into play when it becomes idle in the center of the board. Their compact design and accurate angle control are ideal for quick, localized action.

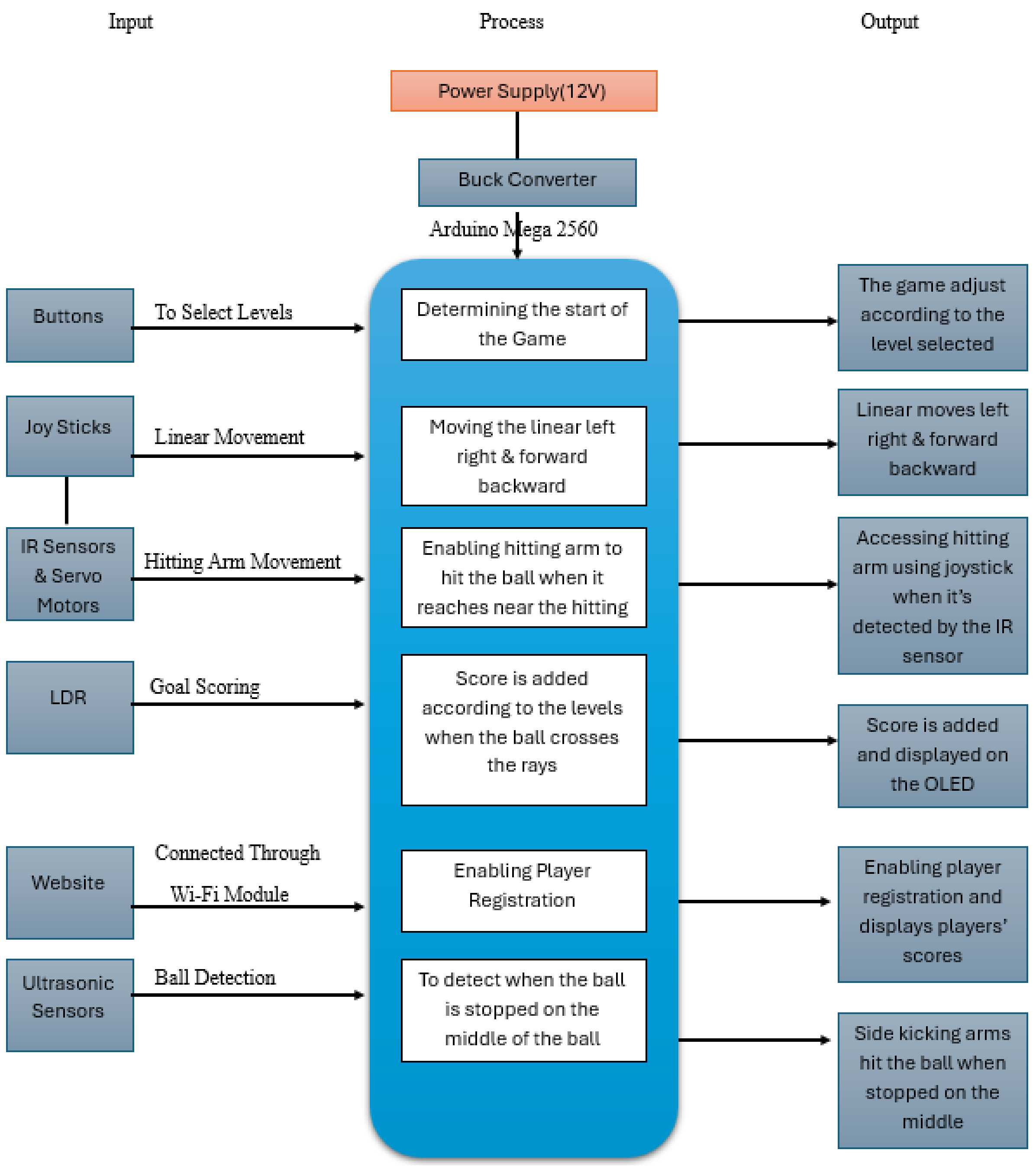
07. Stepper Motors

Stepper motors provide precise and repeatable motion, useful in more controlled arm positioning or vertical/hybrid movements if included in gameplay mechanics. They ensure that game mechanisms move to exact locations as programmed.

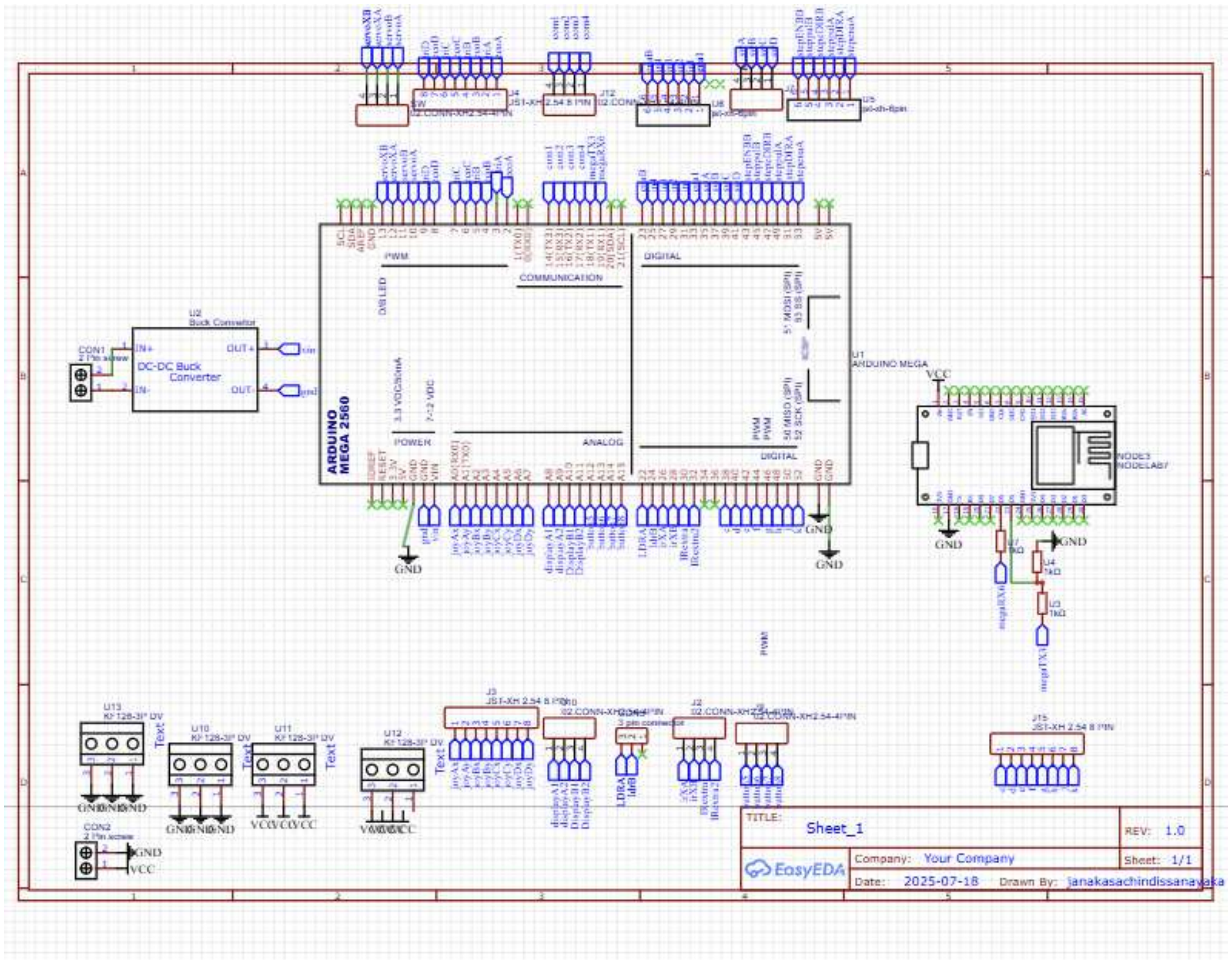
08. OLED Display

An OLED is used to display the real-time score, providing instant visual feedback to players. It enhances the gaming experience by maintaining a competitive and immersive environment.

4.2 System Block Diagram


Input
Process
Output
Power Supply(12V)
Buck Converter
Arduino Mega 2560
Buttons
To Select Levels
Joy Sticks
Linear Movement
IR Sensors & Servo Motors
Hitting Arm Movement
LDR
Goal Scoring
Website
Connected Through Wi-Fi Module
Ultrasonic Sensors
Ball Detection
Determining the start of the Game
Moving the linear left right & forward backward
Enabling hitting arm to hit the ball when it reaches near the hitting
Score is added according to the levels when the ball crosses the rays
Enabling Player Registration
To detect when the ball is stopped on the middle of the ball
The game adjust according to the level selected
Linear moves left right & forward backward
Accessing hitting arm using joystick when it's detected by the IR sensor
Score is added and displayed on the OLED
Enabling player registration and displays players' scores
Side kicking arms hit the ball when stopped on the middle

4.3 Schematic Design for PCB



4.4 PCB Design

The PCB was designed using EasyEDA.

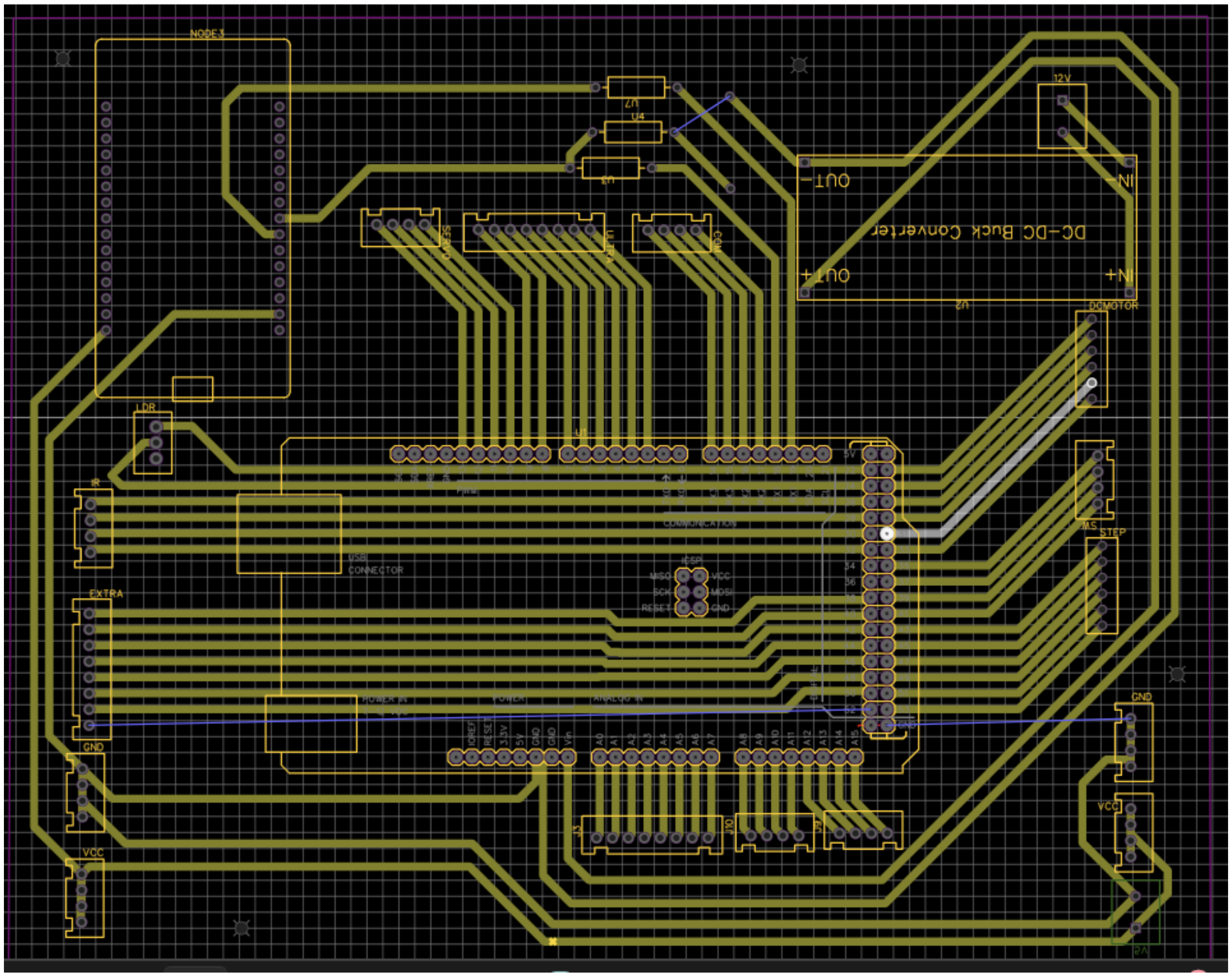
Dimension:187.70mm-152.02mm

Routine Width: 1.30mm

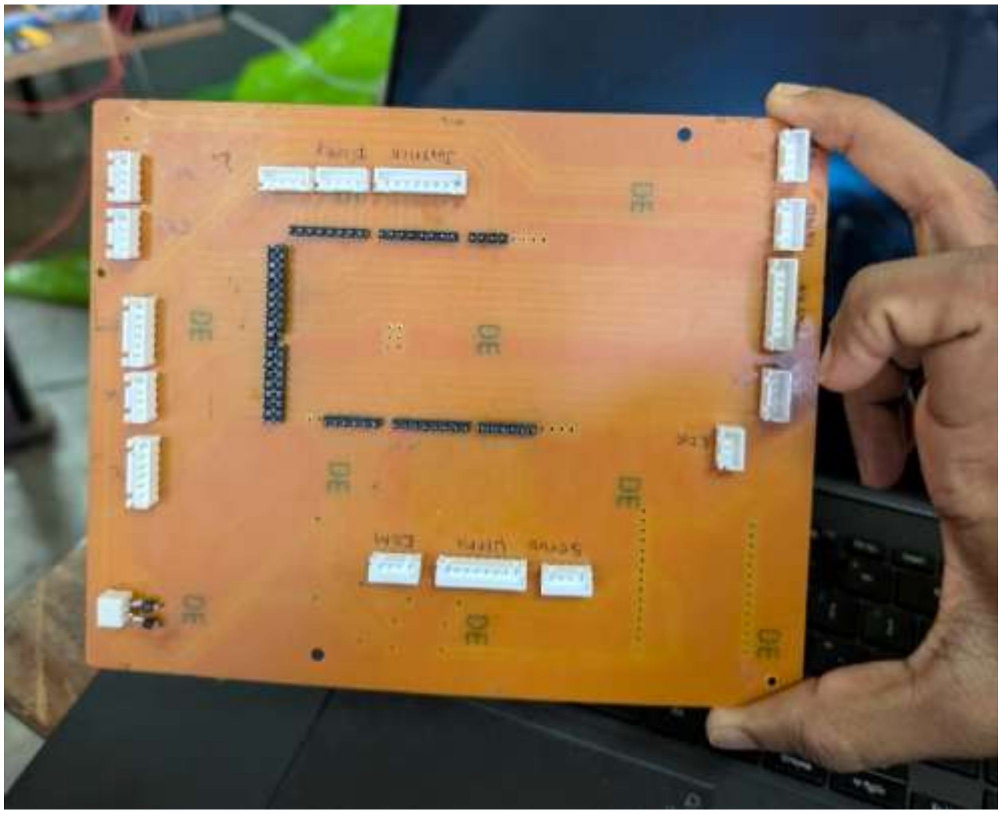
Power tracks: 1.30mm

Data Tracks: 1.30mm(min)

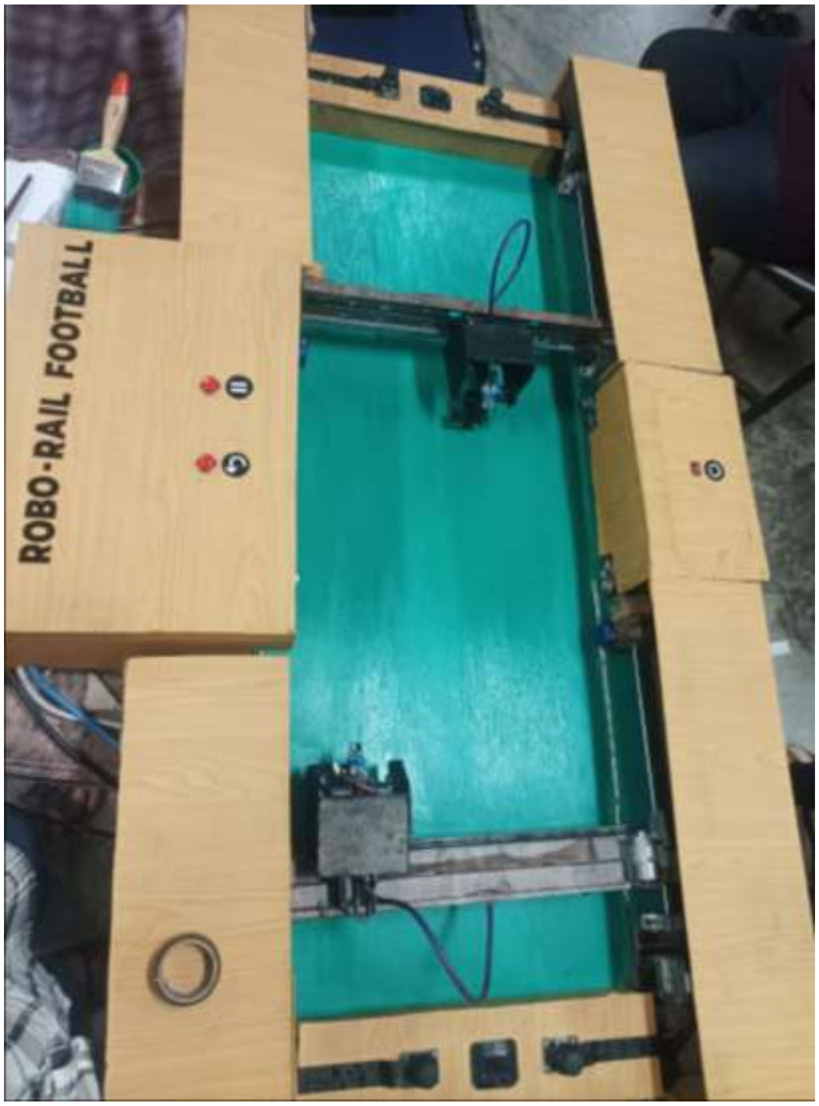
Minimum Clearance: 0.6mm



4.5 PCB Real Design



4.6 Product View



5. Testing & Implementation

When the game starts, players can kick the ball using a joystick toward their opponent, who can also respond with their own joystick controls. Side kicking arms, powered by servo motors, are automatically activated when the ball remains stationary in the middle of the board for more than 3 seconds, ensuring continuous gameplay. Two OLED displays are mounted one for each player showcasing the live scores to boost excitement and engagement throughout the match.

A website is integrated with the game board via a Wi-Fi module to enhance the player experience by player registration and displaying real-time scores.

5.1 Testing & Implementation Process

The tests and implementations were done throughout the development process.

Implementation Process

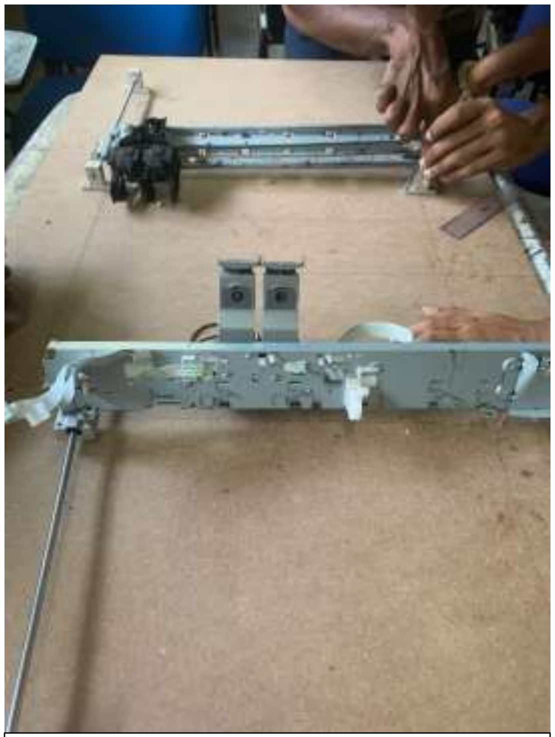
1. Integrating all hardware components on the board

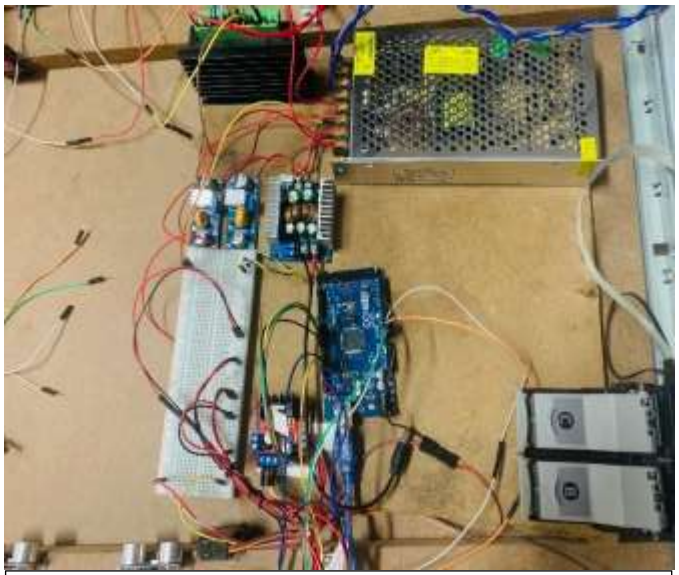
2. Connecting power supply (12V batter) and buck converter for regulated 5V logic

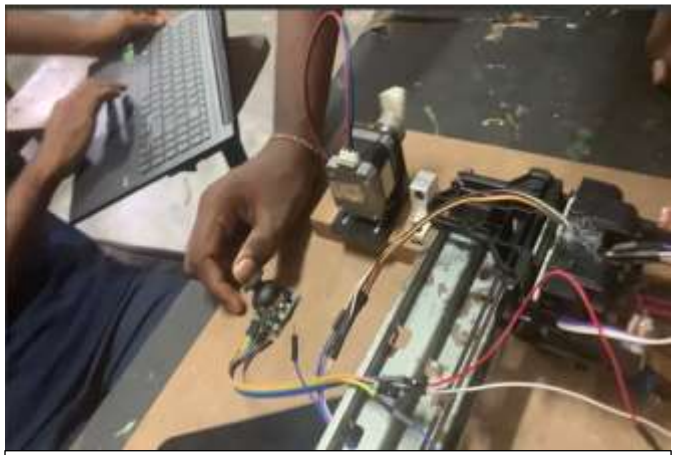
3. Programmed the Arduino with separate logic for each component

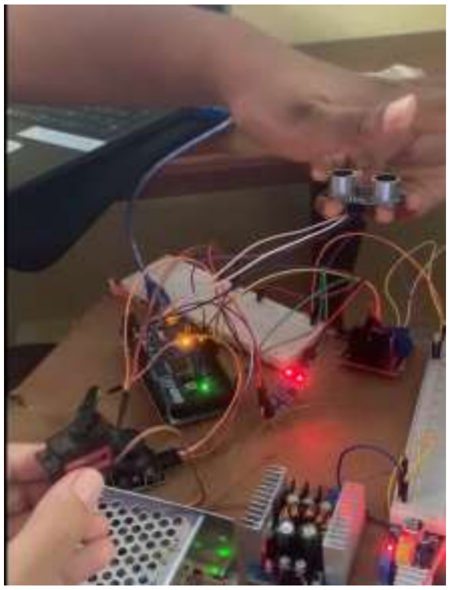
- Player joystick control.
- Automatic kicking arm activation using IR sensors.
- Ball stuck detection using ultrasonic sensors.
- Score updates via laser-LDR module.

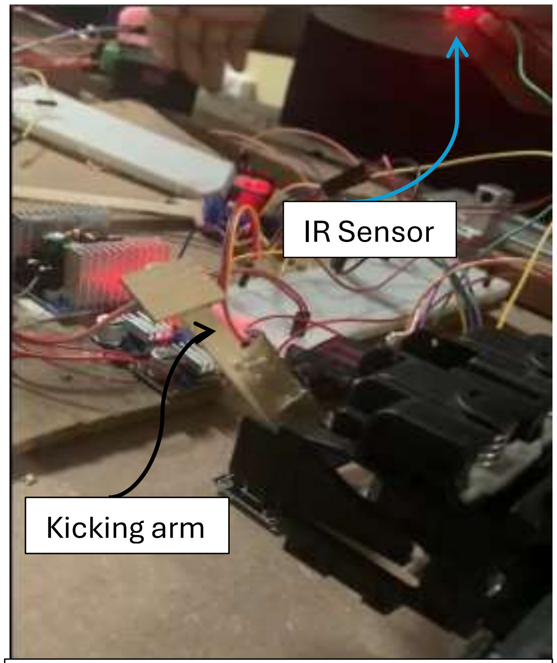
4. Adding dual OLED displays to show real-time scores for both players.
5. Implemented three levels of gameplay with increasing complexity.
6. components on the game board and secured wiring.


Measuring Gameboard


Connecting power supply and buck converter


Checking Joystick


Checking servo motor through ultrasonic sensor



Checking the kicking arm through IR sensor object detection

Testing Process

1. Unit testing

   Verifying individual components (Joy sticks, servo motors, IR sensors, ultrasonic sensors, stepper motors, DC motors)

2. Integration testing
   - Checking the integration between the components
   - Working of servo motor when ball is detected by ultra sonic sensors.
   - Working on kicking arm when the ball is detected by the IR sensor and when the joystick is pressed.

3. Delay testing

   Ensuring kicking arms activate only after the ball is stuck for more than 3 seconds.

4. Score accuracy testing

   Validated laser-LDR module correctly adds points when ball crosses.

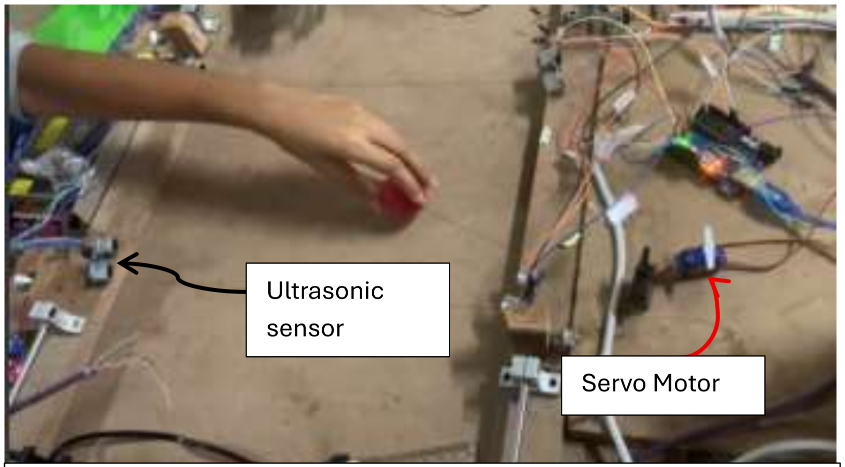
5. Website connectivity testing

   Testing data transmission from Arduino to website through Wi-Fi module.

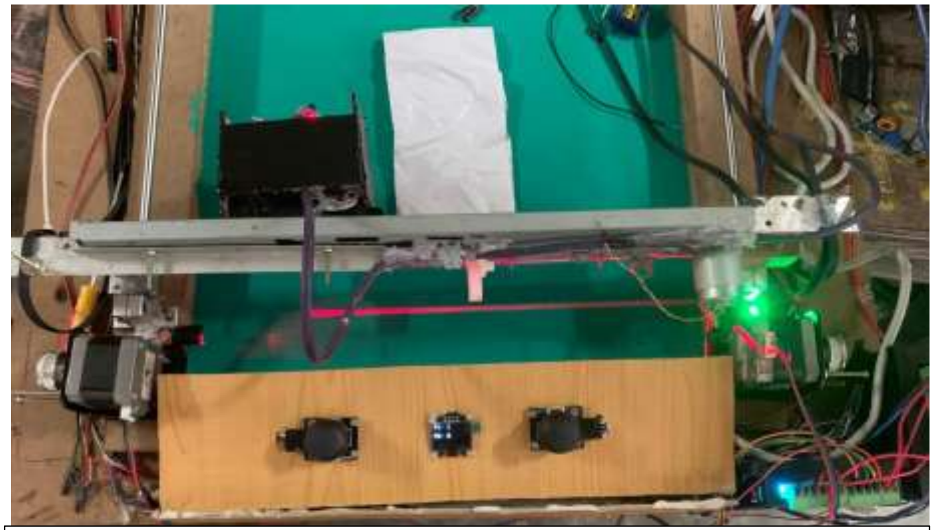
6. Bug fixing and calibration

   Adjusting sensor pulley rotation direction, servo angles, and motor speeds for optimal gameplay.

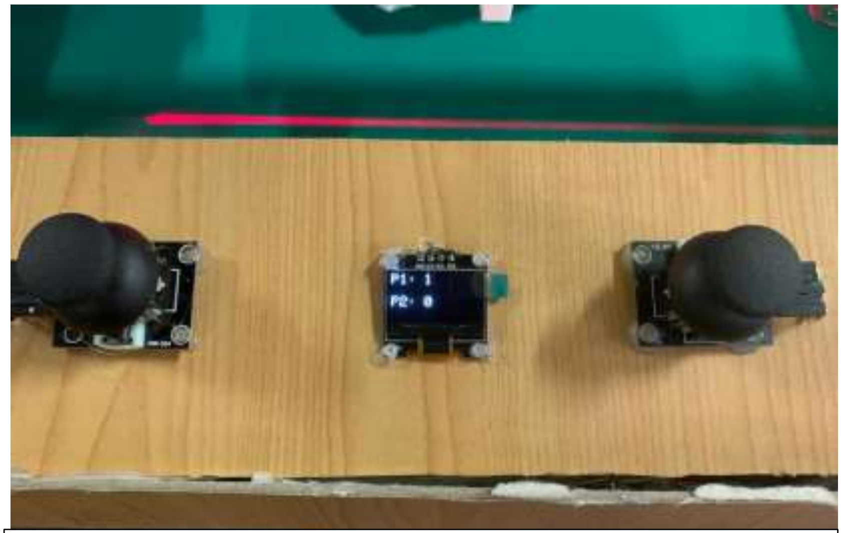
Linear movement using joystick



Activating servo motors through object detection by ultrasonic sensors



Fixing LDR with the laser light to implement goal scoring



Placed OLED display to showcase the score of each player

6.Further Improvements

1. Ball detection enhancement
   Increased the number of ultrasonic sensors from 2 to 4 to improve accuracy and coverage of ball placement on the field.

2. Improved movement control
   Introduced separate joysticks for X and Y axis movement allowing more intuitive and precise control of the player's paddle/platform.

3. Laser + LDR scoring mechanism
   Used laser and LDR modules for accurate goal detection and automatic scoring when the ball crosses the goal line.

7. Individual Contribution

7.1 235013M - Asmitha Udayachandran (Leader)

In this project, my responsibility was to connect and control four ultrasonic sensors along with two servo motors to create an automated system that detects and resolves situations where a ball gets stuck in the middle of the setup. The ultrasonic sensors are positioned around the central area to monitor the space continuously and provide real-time feedback on the ball's position.

Each ultrasonic sensor emits high-frequency sound waves and listens for their echoes to measure the distance to nearby objects. If any sensor detects that an object is within a 30 cm range consistently for more than 3 seconds, it indicates that the ball is likely stuck rather than just passing by briefly. This time threshold is essential to avoid false alarms caused by quick or accidental obstructions.

Once this condition is met, the system sends control signals to both servo motors. These servos then rotate 90 degrees to physically nudge or clear the stuck ball. After a short pause to allow the ball to move, the servos return smoothly to their original 0-degree position, prepared to respond to future detections.

The continuous cycle of measurement, detection, and response ensures that the gameplay remains uninterrupted by automatic clearing of stuck balls. This approach improves the system's reliability by combining precise distance sensing with timely mechanical intervention, maintaining smooth and engaging interaction throughout the game's operation.

Alongside the physical components of the project, I developed a fully functional, responsive website using HTML for the structure, CSS for styling, and JavaScript for interactivity and real-time updates. The website serves as an intuitive interface where players begin by entering their names into designated input fields. Upon clicking the start button, the game session is initiated, and the website establishes a connection to receive live data updates.

The key feature of the website is its ability to display the players' scores dynamically as the game progresses. Leveraging JavaScript's real-time data handling capabilities, the scores are refreshed continuously without the need for manual page reloads, providing a seamless and engaging user experience. This live score display is optimized for mobile devices, allowing players to conveniently view their performance from anywhere during gameplay.

Additionally, the interface includes visual feedback and clear prompts to enhance usability. By combining client-side programming with backend communication, the website bridges the physical gameplay with real-time digital monitoring. This integration not only improves player engagement but also offers a modern, interactive way to track game progress alongside the hardware system.

After successfully establishing serial communication between the ESP32 and Arduino Mega, I proceeded to integrate the website's functionality directly into the ESP32 code. This involved configuring the ESP32 to act as a web server that hosts the website, allowing players to connect via their mobile devices. Whenever the Arduino Mega detected a game event—such as a score change—it communicated the update to the ESP32 over the serial connection.

The ESP32 received this information and immediately updated the web interface by broadcasting the new score data. Through this setup, players could see their scores update live on their mobile screens as the game progressed, without any delay. This real-time feedback greatly enhanced the user experience, providing a seamless connection between the hardware actions and digital display. By embedding the website into the ESP32 and leveraging its WiFi capabilities, I ensured that the system delivered accurate, up-to-the-moment tracking of each player's performance right from the game to their devices.

Code for Ultrasonic Sensors and Servo Motors

```
#include <Servo.h>


// Define ultrasonic sensor pins

#define TRIG_A 8

#define ECHO_A 4


#define TRIG_B 7

#define ECHO_B 3


#define TRIG_C 6
```

```
#define ECHO_C 2

#define TRIG_D 5
#define ECHO_D 9

#define SERVO_LEFT_PIN 10
#define SERVO_RIGHT_PIN 11

Servo servoLeft;
Servo servoRight;

unsigned long detectionStartTime = 0;
bool objectDetected = false;

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

  // Sensor pins
  pinMode(TRIG_A, OUTPUT); pinMode(ECHO_A, INPUT);
  pinMode(TRIG_B, OUTPUT); pinMode(ECHO_B, INPUT);
  pinMode(TRIG_C, OUTPUT); pinMode(ECHO_C, INPUT);
  pinMode(TRIG_D, OUTPUT); pinMode(ECHO_D, INPUT);

  // Attach servos
  servoLeft.attach(SERVO_LEFT_PIN);
  servoRight.attach(SERVO_RIGHT_PIN);
  // Initialize servos to 0°
  servoLeft.write(0);
  servoRight.write(0);
}

long getDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 30000); // 30ms timeout
  return duration * 0.034 / 2;
}

void loop() {
  long distA = getDistance(TRIG_A, ECHO_A);
  long distB = getDistance(TRIG_B, ECHO_B);
  long distC = getDistance(TRIG_C, ECHO_C);
  long distD = getDistance(TRIG_D, ECHO_D);

  Serial.print("A: "); Serial.print(distA);
  Serial.print(" | B: "); Serial.print(distB);
  Serial.print(" | C: "); Serial.print(distC);
  Serial.print(" | D: "); Serial.println(distD);
  bool isClose = (distA < 30 && distA > 0) || (distB < 30 && distB > 0) ||
                 (distC < 30 && distC > 0) || (distD < 30 && distD > 0);

  if (isClose) {
    if (!objectDetected) {
      objectDetected = true;
      detectionStartTime = millis(); // Start timing
    } else if (millis() - detectionStartTime > 3000) {
      // Object is continuously detected for > 3 seconds
      servoLeft.write(90);
      servoRight.write(90);
      delay(500); // Wait for kick
      servoLeft.write(0);
      servoRight.write(0);
      objectDetected = false; // Reset
    }
  } else {
    objectDetected = false; // Reset if not continuously detected
  }

  delay(100);
}
```

7.2 235079T - Arulanantharasa Nirsanth

In our Robo Rail Football Game, I was responsible for setting up the Y-axis movement system. The main objective of this subsystem was to control the linear motion of the hitting arms along the Y-axis to intercept the ball effectively. Initially, our group used two NEMA 17 stepper motors, one on each side of the linear rail, controlled using TB6600 stepper motor drivers. However, during testing, we noticed that the movement was not symmetric. One side moved correctly while the other side lagged or misaligned. This issue caused instability in the hitting arm's motion.

To solve this problem, we upgraded the setup to use four NEMA 17 stepper motors, two on each side of the Y-axis rail. This ensured balanced torque and stable movement of the entire rail. The motors were connected in synchronized pairs so that both motors on a single side rotated at the same time, sharing the load evenly. We fixed a pulley system to the front of each motor and connected them via timing belts to a sliding platform that holds the hitting arms. As the motors rotate, the belts move the platform smoothly along the rail.

For structural support and smooth guidance of the linear movement, we used four 6mm x 300mm shaft smooth rods and SCUL 6mm shaft linear bearings. These components provided rigid and accurate rail guidance, ensuring stable sliding motion and maintained alignment of the Y-axis platform.

The joystick provides analog input, which is read by the Arduino Mega 2560. When the joystick is moved forward or backward, the corresponding stepper motors rotate in the same direction. The direction and number of steps are calculated using the joystick value, and the motor drivers are signaled accordingly.

We selected NEMA 17 stepper motors due to their precision, compact size, and reliability. These motors are ideal for linear motion tasks where accurate positioning is essential. Their ability to move in precise steps ensures that the hitting arm can be placed accurately to hit the ball at the right moment. NEMA 17 motors also offer sufficient torque to move the relatively heavy sliding platform and maintain stable motion without vibrations.

The TB6600 stepper motor drivers were chosen for their robustness and high current capacity. Each TB6600 driver can handle higher current than basic drivers like the A4988, making it suitable for controlling two motors in parallel. It supports microstepping, which provides smoother and more precise movement. Moreover, the TB6600 has built-in protection against overheating and overcurrent, which was essential for a game that operates continuously during gameplay.

Overall, the upgraded 4-motor setup with TB6600 drivers successfully solved the instability issue and enabled a responsive and balanced Y-axis control system.

OLED Display for Score Tracking

In addition to Y-axis control, I implemented the score display system using a 0.96-inch I2C OLED display. Each side has one OLED display positioned for visibility to both players. The display shows the score of both players simultaneously and announces the winner at the end of the match.

The OLED is connected to the Arduino Mega 2560 via the I2C interface (SDA and SCL pins). I used the Adafruit SSD1306 and Adafruit GFX libraries to control the display. These libraries allow easy rendering of text and graphics. Whenever the laser and LDR module detects a scoring event (when the ball crosses the goal line and breaks the laser beam), the Arduino updates the respective player's score and refreshes the OLED display.

We chose the 0.96-inch I2C OLED display for several reasons:

- Compact Size: Fits well on the small game board without obstructing gameplay
- High Contrast: OLED provides sharp, readable text even in bright environments
- Low Power: Suitable for embedded systems where power efficiency is important
- I2C Communication: Uses only two pins (SDA and SCL), leaving other GPIO pins available for motors, sensors, and joysticks

At the start of the game, the OLED screen displays the welcome message "ROBORAIL FOOTBALL GAME" followed by a level selection menu. The players can choose between Easy, Medium, and Hard levels by moving the joystick vertically. The selected difficulty level is highlighted and confirmed before gameplay begins.

After the level is selected, the game proceeds to score tracking. The display shows:

- Player 1's score at the top
- Player 2's score below it
- Game level at the bottom (e.g., Level: Easy)

This layout provides clear visibility of scores and gameplay difficulty. Once a player reaches the winning condition, the OLED display shows the final result, clearly announcing the winner.

Code for OLED Display

```
// ====== OLED DISPLAY FUNCTIONS ======


void splashScoreDisplays() {
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(2);
  scoreDisplay1.setTextColor(SSD1306_WHITE);
  scoreDisplay1.setCursor(10, 8);
  scoreDisplay1.println("ROBORAIL");
  scoreDisplay1.setCursor(10, 30);
  scoreDisplay1.println("FOOTBALL");
  scoreDisplay1.setTextSize(1);
  scoreDisplay1.setCursor(34, 54);
  scoreDisplay1.println("> PLAY <");
  scoreDisplay1.display();


  scoreDisplay2.clearDisplay();
  scoreDisplay2.setTextSize(2);
  scoreDisplay2.setTextColor(SSD1306_WHITE);
  scoreDisplay2.setCursor(10, 8);
  scoreDisplay2.println("ROBORAIL");
  scoreDisplay2.setCursor(10, 30);
  scoreDisplay2.println("FOOTBALL");
  scoreDisplay2.setTextSize(1);
  scoreDisplay2.setCursor(34, 54);
  scoreDisplay2.println("> PLAY <");
  scoreDisplay2.display();
```

```
}

void showPauseScreen() {
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(3);
  scoreDisplay1.setCursor(15, 20);
  scoreDisplay1.println("PAUSED");
  scoreDisplay1.display();

  scoreDisplay2.clearDisplay();
  scoreDisplay2.setTextSize(3);
  scoreDisplay2.setCursor(15, 20);
  scoreDisplay2.println("PAUSED");
  scoreDisplay2.display();
}

void showResumeScreen() {
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(2);
  scoreDisplay1.setCursor(14, 24);
  scoreDisplay1.println("RESUMED");
  scoreDisplay1.display();

  scoreDisplay2.clearDisplay();
  scoreDisplay2.setTextSize(2);
  scoreDisplay2.setCursor(14, 24);
  scoreDisplay2.println("RESUMED");
  scoreDisplay2.display();
}

void difficultySelectionMenu() {
  static unsigned long lastUpdate = 0;
  static bool firstEntry = true;
  if (firstEntry) {
    selectedDifficulty = EASY;
    firstEntry = false;
  }
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(2);
  scoreDisplay1.setCursor(15, 0);
  scoreDisplay1.println("Select");
  scoreDisplay1.setTextSize(3);
  scoreDisplay1.setCursor(14, 28);
  scoreDisplay1.print(DIFFICULTY_NAMES[selectedDifficulty]);
  scoreDisplay1.display();

  scoreDisplay2.clearDisplay();
  scoreDisplay2.setTextSize(3);
  scoreDisplay2.setCursor(14, 16);
  scoreDisplay2.print(DIFFICULTY_NAMES[selectedDifficulty]);
  scoreDisplay2.display();

  int joyX1 = analogRead(joyA1_X);
  int joyX2 = analogRead(joyA2_X);
```

```
if (millis() - lastUpdate > 200) {
  if ((joyX1 > 700 || joyX2 > 700) && selectedDifficulty < HARD) {
    selectedDifficulty = static_cast<Difficulty>(selectedDifficulty + 1);
    lastUpdate = millis();
  }
  if ((joyX1 < 300 || joyX2 < 300) && selectedDifficulty > EASY) {
    selectedDifficulty = static_cast<Difficulty>(selectedDifficulty - 1);
    lastUpdate = millis();
  }
  if (digitalRead(joyA1_SW) == LOW || digitalRead(joyA2_SW) == LOW) {
    difficultySelected = true;
    WIN_SCORE = WIN_SCORE_OPTIONS[selectedDifficulty];
    scoreDisplay1.clearDisplay(); scoreDisplay1.setTextSize(2);
    scoreDisplay1.setCursor(14,24);
    scoreDisplay1.println(DIFFICULTY_NAMES[selectedDifficulty]);
    scoreDisplay1.display();

    scoreDisplay2.clearDisplay(); scoreDisplay2.setTextSize(2);
    scoreDisplay2.setCursor(14,24);
    scoreDisplay2.println(DIFFICULTY_NAMES[selectedDifficulty]);
    scoreDisplay2.display();

    delay(1000);
    firstEntry = true;
  }
}
```

```
}


void showScoresOnBoth() {
  scoreDisplay1.clearDisplay(); scoreDisplay1.setTextSize(2);
scoreDisplay1.setCursor(0, 0);
  scoreDisplay1.print("P1: "); scoreDisplay1.println(player1Score);
scoreDisplay1.setCursor(0, 32);
  scoreDisplay1.print("P2: "); scoreDisplay1.println(player2Score);
scoreDisplay1.setCursor(0, 56);
  scoreDisplay1.print("LVL:");
scoreDisplay1.print(DIFFICULTY_NAMES[selectedDifficulty]);
  scoreDisplay1.display();


  scoreDisplay2.clearDisplay(); scoreDisplay2.setTextSize(2);
scoreDisplay2.setCursor(0, 0);
  scoreDisplay2.print("P1: "); scoreDisplay2.println(player1Score);
scoreDisplay2.setCursor(0, 32);
  scoreDisplay2.print("P2: "); scoreDisplay2.println(player2Score);
scoreDisplay2.setCursor(0, 56);
  scoreDisplay2.print("LVL:");
scoreDisplay2.print(DIFFICULTY_NAMES[selectedDifficulty]);
  scoreDisplay2.display();
}


void showWinnerOnBoth() {
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(2);
  scoreDisplay1.setCursor(0, 0);
  if (player1Score > player2Score) {
    scoreDisplay1.println("WINNER:"); scoreDisplay1.setCursor(0,30);
scoreDisplay1.println("PLAYER 1");
  } else if (player2Score > player1Score) {
    scoreDisplay1.println("WINNER:"); scoreDisplay1.setCursor(0,30);
scoreDisplay1.println("PLAYER 2");
  } else { scoreDisplay1.println("DRAW"); }
  scoreDisplay1.display();


  scoreDisplay2.clearDisplay(); scoreDisplay2.setTextSize(2);
scoreDisplay2.setCursor(0, 0);
  if (player1Score > player2Score) {
    scoreDisplay2.println("WINNER:"); scoreDisplay2.setCursor(0,30);
scoreDisplay2.println("PLAYER 1");
  } else if (player2Score > player1Score) {
    scoreDisplay2.println("WINNER:"); scoreDisplay2.setCursor(0,30);
scoreDisplay2.println("PLAYER 2");
  } else { scoreDisplay2.println("DRAW"); }
  scoreDisplay2.display();
}


void showRestartSplash() {
  scoreDisplay1.clearDisplay();
  scoreDisplay1.setTextSize(2);
  scoreDisplay1.setCursor(0, 0);
  scoreDisplay1.println(" New Game ");
  scoreDisplay1.setCursor(0,28);
  scoreDisplay1.println(" Get Ready!");
  scoreDisplay1.display();
  scoreDisplay2.clearDisplay();
  scoreDisplay2.setTextSize(2);
  scoreDisplay2.setCursor(0, 0);
  scoreDisplay2.println(" New Game ");
  scoreDisplay2.setCursor(0,28);
  scoreDisplay2.println(" Get Ready!");
  scoreDisplay2.display();
}
```

Code for Y Axis (Stepper Motor)

```
// === Stepper Motor Control Pins ===
const int DIR1 = 51, STEP1 = 49;
const int DIR2 = 47, STEP2 = 45;
const int DIR3 = 43, STEP3 = 41;
const int DIR4 = 39, STEP4 = 37;

// === Joystick Inputs for Y-Axis Movement ===
const int joyY1 = A3;  // Controls Stepper 1
const int joyY2 = A2;  // Controls Stepper 2
const int joyY3 = A4;  // Controls Stepper 3
const int joyY4 = A5;  // Controls Stepper 4

// === Stepper Timing Variables ===
unsigned long prevStepTime1 = 0, prevStepTime2 = 0;
unsigned long prevStepTime3 = 0, prevStepTime4 = 0;
unsigned long stepInterval1 = 1000, stepInterval2 = 1000;
unsigned long stepInterval3 = 1000, stepInterval4 = 1000;
```

```
bool stepState1 = false, stepState2 = false;
bool stepState3 = false, stepState4 = false;

const int center = 512;
const int threshold = 100;  // Deadzone around joystick center

void setup() {
  pinMode(DIR1, OUTPUT); pinMode(STEP1, OUTPUT);
  pinMode(DIR2, OUTPUT); pinMode(STEP2, OUTPUT);
  pinMode(DIR3, OUTPUT); pinMode(STEP3, OUTPUT);
  pinMode(DIR4, OUTPUT); pinMode(STEP4, OUTPUT);
}

void loop() {
  int y1 = analogRead(joyY1);
  int y2 = analogRead(joyY2);
  int y3 = analogRead(joyY3);
  int y4 = analogRead(joyY4);

  controlStepper(y1, DIR1, STEP1, prevStepTime1, stepInterval1, stepState1);
  controlStepper(y2, DIR2, STEP2, prevStepTime2, stepInterval2, stepState2);
  controlStepper(y3, DIR3, STEP3, prevStepTime3, stepInterval3, stepState3);
  controlStepper(y4, DIR4, STEP4, prevStepTime4, stepInterval4, stepState4);
}

// === Stepper Motor Control Function ===
void controlStepper(int yVal, int dirPin, int stepPin,
                    unsigned long &prevTime, unsigned long &interval, bool &stepState) {
  int delta = yVal - center;

  if (abs(delta) > threshold) {
    digitalWrite(dirPin, delta > 0 ? LOW : HIGH);
    interval = map(abs(delta), threshold, 512, 1500, 200);

    if (micros() - prevTime >= interval) {
      prevTime = micros();
      stepState = !stepState;
      digitalWrite(stepPin, stepState);
    }
  } else {
    digitalWrite(stepPin, LOW);
  }
}
```

7.3 235037N – M.R.F. Fahma

I was responsible for developing the goal scoring system in our Robo Rail Football Game. To detect goals accurately, I used a laser and an LDR (Light Dependent Resistor) sensor setup.

In this system, a laser beam is aimed at an LDR placed on the opposite side of the goal. Under normal conditions, the LDR receives the laser light continuously, keeping its resistance low. When the ball crosses the goal line, it blocks the laser beam, which causes the LDR to receive less light. This results in a sudden change in its resistance.

Arduino reads the LDR's analog values. When the light level drops below a set threshold, the system recognizes it as a goal. Once a goal is detected, the following actions are performed:

- The player's score is increased by 1,
- The OLED display is updated to show the new scores for both players,
- A short delay is added to prevent multiple counts from the same goal.

This system provides a fast, contactless, and reliable way of scoring, which makes the gameplay smoother and more enjoyable.

In addition to the scoring system, we also implemented three game levels to increase the challenge and fun.

I was also responsible for implementing the game levels which are Easy, Medium, Hard for the Robo Rail Football game. Game levels are displayed on the OLED and can be selected as preferred using the joystick. Game levels are explained as follows.

Game Levels

- Level 1:
  - Only X axis movement is activated.
  - IR sensor automatically detects the ball and kick the ball.
  - players need to reach a total score of 3 points to win.
- Level 2:
  - Both X axis and Y axis movements are activated.
  - IR sensor automatically detects the ball and kick it.

  - players need to reach a total score of 4 points to win.
- Level 3:
  - Both X axis and Y axis movements are activated.
  - IR sensors are disabled- only joystick button presses can trigger servos.
  - players need to reach a total score of 5 points to win.

In level 1 & 2 also players can use the joystick button to trigger the kicking arm.

Code for the Goal Scoring From LDR

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 displayA(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);  // Player A
Adafruit_SSD1306 displayB(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);  // Player B

const int ldrA = A8;  // Player A's goal detector (Laser - LDR)
const int ldrB = A9;  // Player B's goal detector

int scoreA = 0;
int scoreB = 0;
```

```
const int threshold = 500;  // Light threshold for goal detection
unsigned long lastGoalTimeA = 0;
unsigned long lastGoalTimeB = 0;
const unsigned long debounceDelay = 3000; // 3 seconds cooldown

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

  // Initialize OLEDs
  displayA.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  displayB.begin(SSD1306_SWITCHCAPVCC, 0x3D); // Change if your second OLED
has different I2C address

  displayA.clearDisplay();
  displayB.clearDisplay();

  displayA.setTextSize(2);
  displayA.setTextColor(SSD1306_WHITE);

  displayB.setTextSize(2);
  displayB.setTextColor(SSD1306_WHITE);

  updateScores();
}

void loop() {
  int ldrValueA = analogRead(ldrA);
```

```
 int ldrValueB = analogRead(ldrB);
 unsigned long currentTime = millis();

 // Check Player A's goal (Player B scores)
 if (ldrValueA < threshold && currentTime - lastGoalTimeA > debounceDelay) {
  scoreB++;
  lastGoalTimeA = currentTime;
  updateScores();
  Serial.println("Player B Scored!");
 }

 // Check Player B's goal (Player A scores)
 if (ldrValueB < threshold && currentTime - lastGoalTimeB > debounceDelay) {
  scoreA++;
  lastGoalTimeB = currentTime;
  updateScores();
  Serial.println("Player A Scored!");
 }

 // Add a small delay to reduce CPU usage
 delay(50);
}
```

```
// ========== Update OLED Scores ==========
void updateScores() {
  displayA.clearDisplay();
  displayA.setCursor(0, 0);
  displayA.print("A: ");
  displayA.print(scoreA);
  displayA.display();


  displayB.clearDisplay();
  displayB.setCursor(0, 0);
  displayB.print("B: ");
  displayB.print(scoreB);
  displayB.display();
}
```

<u>Code for the 3 Game Levels</u>

```
// ========== GAME LOOP MOVEMENT & SERVO CONTROL BASED ON LEVEL
==========
if (selectedDifficulty == EASY) {
  // LEVEL 1: Only X-axis DC motor movement
  controlMotor(xA1, motorA1, motorA2, enableA);
  controlMotor(xA2, motorB1, motorB2, enableB);
} else {
  // LEVEL 2 & 3: Both X-axis (DC motor) and Y-axis (stepper) movement
  controlStepper(yB1, DIR1, STEP1, prevStepTime1, stepInterval1, stepState1);
  controlStepper(yB2, DIR2, STEP2, prevStepTime2, stepInterval2, stepState2);
  controlMotor(xA1, motorA1, motorA2, enableA);
  controlMotor(xA2, motorB1, motorB2, enableB);
}

// --- BUTTON-BASED SERVO CONTROL (ALWAYS ALLOWED) ---
bool btnA1_extra = (digitalRead(joyA1_SW) == LOW);
bool btnA2_extra = (digitalRead(joyA2_SW_EXTRA) == LOW);
if (btnA1_extra) { servo1.write(90); delay(200); servo1.write(0); }
if (btnA2_extra) { servo2.write(90); delay(200); servo2.write(0); }

// --- IR SENSOR SERVO CONTROL: ENABLED ONLY FOR LEVEL 1 & 2 ---
if (selectedDifficulty == EASY || selectedDifficulty == MEDIUM) {
  if (digitalRead(IR1_PIN) == LOW) { servo1.write(90); delay(200); servo1.write(0);
delay(200);}
  if (digitalRead(IR2_PIN) == LOW) { servo2.write(90); delay(200); servo2.write(0);
delay(200);}
}
```

7.4 235033A - Janaka Sachin Dissanayaka

I designed a single-layer PCB to support the core components of the Robo Rail Football Game, including the Arduino Mega, power lines, I/O headers, and other circuit elements. The PCB was designed using EasyEDA, with careful attention to routing and layout.

PCB Design Measurements

Dimension:187.70mm-152.02mm

Routine Width: 1.30mm

Power tracks: 1.30mm

Data Tracks: 1.30mm(min)

Minimum Clearance: 0.6mm

X-Axis Movement

To enable horizontal movement of the player units, we implemented X-axis control using two DC motors driven by the HW-095 motor driver module. Two joysticks were used, and only their X-axis (A0, A1) readings were used to control the motors. The analog input values were processed in software to determine motor direction and speed. This design filtered out Y-axis noise and ensured smooth lateral movement. A motor control function was implemented to interpret joystick input and apply appropriate motor direction and PWM signals for movement.

ESP 32 Wi-Fi Module

We integrated an ESP32 development board to enable WiFi communication for live score updates. A hardware UART link was established using Serial3 (pins 14 and 15) on the Arduino Mega, with correct TX-RX and GND connections. Arduino code was written to format and send live score data (e.g., "P1:3,P2:2") to the ESP32. This allowed real-time game information to be displayed on a connected web interface, enhancing the interactive experience.

Sound & Light Effects

We added sound and lighting effects to enhance player feedback. A buzzer was used to signal goal events and victory, while an LED strip (connected to pin 48) provided visual feedback. The LED remains on during normal play, blinks twice when a player scores, and blinks five times during the winning celebration. These effects were programmed using simple timed routines and added excitement to the overall gameplay experience.

Pause/Resume Buttons

We developed the main game loop to manage gameplay, input, and scoring logic. All user controls were implemented, including buttons for pause/resume and undo/redo score adjustments. The system handled real-time sensor data, joystick input, and scoring updates to ensure a fluid and responsive multiplayer game. Display updates and sound/lighting cues were fully synchronized with gameplay events.

Code Finalization

The codebase was refactored for clarity, modularity, and maintainability. Proper comments and structure were used throughout the Arduino program. Each module (scoring, movement, menu, feedback) was tested individually and as part of the full system. We used serial debugging and live gameplay tests to verify all features. The final integration confirmed that all hardware connections and software modules were aligned with the PCB and wiring layout.

Wiring Finalization

We performed detailed mapping of each hardware component to specific Arduino Mega and PCB pins, including motors, joysticks, IR and ultrasonic sensors, HW-095 drivers, ESP32, OLED displays, LDRs, buzzer, and LED strip. Proper color-coded wiring was used to distinguish signal lines from power (Vcc and GND). For power-intensive components such as motors and LED strips, 1.3 mm wide traces and appropriate wire gauges were used.

All connectors were carefully soldered and tested to avoid cold joints. Motor and sensor harnesses were arranged for minimal interference, with twisted-pair wires used where necessary. Joystick X-axis lines (A0, A1) and button inputs were routed directly with slack to allow full movement. Sensors were placed near MCU inputs for faster readings.

ESP32 integration involved connecting ESP TX to Arduino RX3 (pin 15), ESP RX to Arduino TX3 (pin 14), and GND to GND. These UART connections were tested for signal continuity and correct logic levels. Dual OLED displays were wired via I2C (SDA/SCL), sharing pull-up resistors, and powered with stable voltage.

Buzzer and LED strip were connected to pins 42 and 48 respectively, ensuring proper polarity and spacing from other high-noise components. All Vcc and GND lines were routed with low resistance, with voltage confirmed at all key points.

Difficulty Selection Menu

I implemented a difficulty selection menu that players can navigate using the X-axis of the joysticks, allowing them to select the desired game level with a simple button press. To manage the flow of the game, I developed a robust state machine encompassing key stages such as the splash screen, menu selection, active gameplay, pause/resume, and reset. Each state was carefully designed to ensure smooth transitions, maintaining user engagement and system responsiveness. Additionally, I integrated appropriate hardware feedback mechanisms in every state, ensuring that all components responded accurately to user inputs and transitions.

To ensure durability, every wire was labeled at both ends, insulated using heat shrink, and tied down to the chassis. The complete wiring system was tested section by section motors, displays, sensors, communication modules, lights, and buzzer ensuring correct functionality, clean signal flow, and robust operation during gameplay.

Code for the DC Motor

```
const int motorA1 = 31, motorA2 = 29, enableA = 33; // change pin numbers as needed
const int joyA1_X = A0;
const int center = 512, threshold = 100, motorSpeed = 200;

void setup() {
  pinMode(motorA1, OUTPUT);
  pinMode(motorA2, OUTPUT);
  pinMode(enableA, OUTPUT);
}

void loop() {
  int joyVal = analogRead(joyA1_X);  // Read joystick X-axis

  int diff = joyVal - center;        // Centered joystick ≈ 512
  if (abs(diff) > threshold) {
    if (diff > 0) {
      digitalWrite(motorA1, HIGH);
      digitalWrite(motorA2, LOW);
      analogWrite(enableA, motorSpeed);
    } else {
      digitalWrite(motorA1, LOW);
      digitalWrite(motorA2, HIGH);
      analogWrite(enableA, motorSpeed);
    }
  } else {
    digitalWrite(motorA1, LOW);
    digitalWrite(motorA2, LOW);
    analogWrite(enableA, 0);
  }
  delay(10);
}
```

<u>Passing Score to Wi-Fi Module</u>

```
// Arduino Mega
void sendScoresToESP32() {
  // Send scores in format: "P1:score1,P2:score2"
  String scoreData = "P1:" + String(player1Score) + ",P2:" + String(player2Score);
  Serial3.println(scoreData);                    // Send to ESP32 (connect TX3/RX3)
  Serial.print("Sent to ESP32: ");
  Serial.println(scoreData);
}
```

<u>Code for Undo Button</u>

```
const int undoButtonPin = 46;
bool undoBtnPrevState = HIGH;
int player1Score = 0, prevPlayer1Score = 0;

void setup() {
  pinMode(undoButtonPin, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  // Imagine 'player1Score' gets incremented elsewhere
  bool undoBtnNow = digitalRead(undoButtonPin);
  if (undoBtnPrevState == HIGH && undoBtnNow == LOW) { // Button pressed
    player1Score = prevPlayer1Score;
    Serial.print("Undo: player1Score restored to "); Serial.println(player1Score);
    delay(200); // debounce
  }
  undoBtnPrevState = undoBtnNow;
}
```

<u>Code for Pause Button</u>

```
const int pauseButtonPin = 44;
bool paused = false;
bool pauseBtnPrevState = HIGH;

void setup() {
  pinMode(pauseButtonPin, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  bool pauseBtnNow = digitalRead(pauseButtonPin);
  if (pauseBtnPrevState == HIGH && pauseBtnNow == LOW) {
    paused = !paused; // Toggle pause
    Serial.println(paused ? "PAUSED" : "RESUMED");
    delay(150); // debounce
  }
  pauseBtnPrevState = pauseBtnNow;

  if (paused) {
    // All gameplay code pauses (return or skip main logic)
    return;
  }

  // ...normal game code here...
}
```

<u>Code for Light Strip Blink(For the Winner)</u>

```
const int ledStripPin = 48;

void setup() {
  pinMode(ledStripPin, OUTPUT);
  digitalWrite(ledStripPin, HIGH); // Default ON
}

void loop() {
  // Call this function when you want to blink
  ledStripScoreBlink();
  delay(3000); // Wait before next test
}

void ledStripScoreBlink() {
  for (int i = 0; i < 2; i++) {
    digitalWrite(ledStripPin, LOW); delay(150);
    digitalWrite(ledStripPin, HIGH); if (i < 1) delay(150);
  }
}
```

Code for the Buzzer

```
const int scoreBuzzerPin = 42;

void setup() {
  pinMode(scoreBuzzerPin, OUTPUT);
}

void loop() {
  playScoreSound();
  delay(2000); // Wait a bit before next buzz
}

void playScoreSound() {
  tone(scoreBuzzerPin, 1000, 300); // Frequency, duration(ms)
}
```

Code for Start/Splash Menu on OLED

```
#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64

#define OLED_RESET -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);


const int joyA1_SW = 35;

bool waitingForPlayButton = true;


void setup() {

  pinMode(joyA1_SW, INPUT_PULLUP);

  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

}


void loop() {

  if (waitingForPlayButton) {

    splashScoreDisplay();

    if (digitalRead(joyA1_SW) == LOW) {

      waitingForPlayButton = false;

      delay(200); // debounce

    }

    delay(50);

    return;
}

void splashScoreDisplay() {
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 8);
  display.println("ROBORAIL");
  display.setCursor(10, 30);
  display.println("FOOTBALL");
  display.setTextSize(1);
  display.setCursor(34, 54);
  display.println("> PLAY <");
  display.display();
}
```

Difficulty Selection Menu Using Joystick and OLED

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

const int joyA1_X = A0;
const int joyA1_SW = 35;
enum Difficulty { EASY = 0, MEDIUM = 1, HARD = 2 };
Difficulty selectedDifficulty = EASY;
const char* DIFFICULTY_NAMES[3] = {"EASY", "MEDIUM", "HARD"};
bool difficultySelected = false;

void setup() {
  pinMode(joyA1_SW, INPUT_PULLUP);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  pinMode(joyA1_X, INPUT);
}

void loop() {
  if (!difficultySelected) {
    difficultySelectionMenu();
    delay(50);
  }
}

void difficultySelectionMenu() {
  static unsigned long lastUpdate = 0;
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(15, 0);
  display.println("Select");
  display.setTextSize(3);
  display.setCursor(14, 28);
  display.print(DIFFICULTY_NAMES[selectedDifficulty]);
  display.display();

  int joyX1 = analogRead(joyA1_X);

  if (millis() - lastUpdate > 200) {
    if ((joyX1 > 700) && selectedDifficulty < HARD) {
      selectedDifficulty = (Difficulty)(selectedDifficulty + 1);
      lastUpdate = millis();
    }
    if ((joyX1 < 300) && selectedDifficulty > EASY) {
      selectedDifficulty = (Difficulty)(selectedDifficulty - 1);
      lastUpdate = millis();
    }
    if (digitalRead(joyA1_SW) == LOW) {
      difficultySelected = true;
      display.clearDisplay(); display.setTextSize(2);
      display.setCursor(14,24);
      display.println(DIFFICULTY_NAMES[selectedDifficulty]);
      display.display();
      delay(1000);
    }
  }
}
```

7.5 235031P - W.P Imasha Dilshani

In the Robo Rail Football Game project, my main responsibility was to design and implement the automatic ball detection and kicking mechanism, which is a core part of the robot's interactive gameplay system. This mechanism was built using an Infrared (IR) sensor and a servo motor.

I began by carefully placing the IR sensor at the front of the robot, in a position where it could accurately detect when the ball entered the kicking zone. The IR sensor works by sending out infrared light and checking for reflections. When the ball comes close to the sensor, it reflects the IR signal back, and the sensor detects its presence. This signal is then sent to the Arduino microcontroller, which acts as the brain of the robot.

Once the ball is detected, the microcontroller immediately activates the servo motor, which is connected to the kicking arm. I programmed the servo to rotate the arm to 90 degrees quickly and precisely, delivering a strong and controlled kick to push the ball forward toward the opponent’s side. After completing the kick, the servo then returns the arm to its original position (0 degrees), so it's ready to kick again when the next ball is detected.

To ensure that this system performs reliably during gameplay, I wrote and tested the full Arduino code for the module. I focused on:

- Making the detection responsive, so the kick happens instantly after the ball enters the zone.
- Ensuring timing accuracy, to avoid multiple or false kicks if the ball stays too long in front of the sensor.
- Maintaining stability, so the servo operates consistently without jerking or lagging.

This subsystem plays a crucial role in the game, as it allows the robot to react automatically to the ball without needing any manual control from the player during Level 1 of the game. It enhances the speed and flow of gameplay, making the robot feel more intelligent and interactive, and contributes significantly to the overall functionality and success of the Robo Rail Football Game.

Code for IR sensor

```
#include <Servo.h>

const int IR1_PIN = 32;
const int IR2_PIN = 30;

const int SERVO1_PIN = 38;
const int SERVO2_PIN = 40;

Servo servo1, servo2;

// Difficulty enum and variable
enum Difficulty { EASY = 0, MEDIUM = 1, HARD = 2 };
Difficulty selectedDifficulty = EASY;

void setup() {
  pinMode(IR1_PIN, INPUT);
  pinMode(IR2_PIN, INPUT);

  servo1.attach(SERVO1_PIN);
  servo2.attach(SERVO2_PIN);

  servo1.write(0);
  servo2.write(0);
}

void loop() {
  // IR SENSOR SERVO CONTROL: ONLY IN EASY MODE
  if (selectedDifficulty == EASY) {
    if (digitalRead(IR1_PIN) == LOW) {
      servo1.write(90);
      delay(200);
      servo1.write(0);
      delay(200);
    }
    if (digitalRead(IR2_PIN) == LOW) {
      servo2.write(90);
      delay(200);
      servo2.write(0);
      delay(200);
    }
  }

  // Add any other logic or delay if needed
}
```

8.References

Arduino Mega 2560
Documentation
https://docs.arduino.cc/hardware/mega-2560/

Datasheet:
https://www.mouser.com/catalog/specsheets/ArduinoBoardMega2560.pdf

Wi-Fi Module (ESP32)
Documentation
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html

LDR (Light Dependent Sensor)
https://robocraze.com/blogs/post/how-ldr-sensor-works

https://youtu.be/wcLeXXATCR4

https://youtu.be/OLFjdCk2nS8