An-Najah National University

Faculty of Engineering & Information Technology

Department of Computer Engineering

Hardware Project

Beat Me

Supervisor:

Dr. Ashraf Armoush

.

.

Student Name:

Ayham Omar Al-Duwairi

Ahmad Mohammed Nazzal

September 7, 2024

Acknowledgment

A special thanks to our supervisor, Dr. Ashraf Armoush. This is because, he was at the forefront in providing useful information and constructive criticism throughout the course of this project. Effectively, we have a supervisor who works with us, not for us.

We are also very thankful to the carpenter, Alaa Zorba who was responsible for putting together the items within our design to appreciate his effort. His work ethic, focus on quality and details were paramount in realizing the objectives of our project.

We finally appreciate everyone who has been there for us throughout the process and offered a helping hand whenever possible. Whether you were active or passive in helping us in completing this project, all of us received help to make the process easier. Thanks to all of you for your thoughtfulness and generosity.

Dedications

This work is dedicated to the martyrs of Gaza and the West Bank, especially those from Jenin and Tulkarm, whose courage and sacrifice inspire us every day. To our family, whose unwavering love and support are our constant source of strength and inspiration, and to everyone who has offered their help and guidance along the way,thank you for being our steadfast support in both challenging and joyous times. Your bravery, dedication, and kindness have made this project possible.

Abstract

This project of an air hockey robot has pretty great importance due to its wide range of applications, such as practicing for competitions, entertaining, and even earning some money. The main purpose during this project's construction was to design and create a high-tech robotic environment for the improvement of the engagement levels. The bot is designed to play air hockey autonomously, using a table that has holes drilled in it purposefully for friction reduction; the Oak-D camera for accurately and clearly detecting the puck; and the Raspberry Pi 4, useful for image processing and taking on quick decisions. The robot's movements are well-structured so that any game in defense or offense can be properly and promptly responsive. More importantly, an auto defense mode was added, where the robot is designed to keep the goal's defense in mind while doing high-speed gameplay. A joystick mode is also implemented in the system whereby the user can manually control the robot with the use of an app. The application does not only toggle the auto and joystick modes but controls, in real-time, the movements which the robot will achieve with the use of a joystick. The major objectives of the project are an autonomous air hockey robot, real-time image processing for tracking of the puck, highly responsive robotic system, and flexible app-level interface-based control.

This will involve the design of a low-friction table, setting up the camera and Raspberry Pi 4 of Oak-D, developing sophisticated algorithms that detect and control the movement, integrating hardware and software components into one system, and extended testing and optimization for performance and reliability. While several projects of air hockey robots have been made, this project differs from them in that it leverages recent technologies, like automatic defense mode and app-controlled joystick mode, to take the enjoyment and competitiveness of the game even further. This robust and adaptive solution caters to various user needs in disparate uses for an engaging experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The objective of the air hockey robot project is to combine technology and robotics in the development of an autonomous system that can play a game of air hockey. Thanks to high performance image capturing and advanced control of motion, the robot can be utilized for numerous purposes, including competition training, entertainment, and prospects for a business, thanks to the variety, offers making it in the first place soft ware able fast image process and decision making. For puck detection, the robot employs the use of an Oak-D camera and a Raspberry Pi 4 to enable quick processing of captured images as well as form a strategy of playing the game. The developmental phase of the project includes the construction of a low friction table surface along with clever software and hardware, with the more fast scene capture, small volume matrix baffling. The project intends to widen the boundaries of both horizons by addressing the issues of the competitive and recreational processes. It is expected that the results obtained during the reconstruction of the dynamics of air hockey robots can improve their design thanks to new technologies tested within the project.

1.2 Objective of the work

The main goal of this thesis is the functional design and implementation of a robotic system capable of interacting in different modes. Its first and foremost purpose is in order to develop the training of the players themselves, elevating a few of them into top gamers. It also has an element of entertainment where he user can play a game against the robot with just him and that bot, which acts as the highly intelligent advanced version opponent. Not just for training or entertainment, this project may also commercialize when installed in several places where active children and adults reach a critical mass so that the entire room is transformed into beneficial fun games against each other.

1.3 Scope of the work

The Air Hockey Robot project we did, focused on several important steps thus it was such a remarkable projecte specially a very detailed perspective was looked at in relation to the claims. It initiated with the air hockey table plan,emphasizing height and materials for constructing the table just as its primary functining was designed. After that, we 3D modeled the robot, and then proceeded to its essential parts which were designed from scratch with fringe parameters of this unique-to-be-built requirements for our UI. These systems were then installed into the table so that it fit snugly and worked seamlessly.

We then procured all the motors, drivers and electronics required in order to move things. Every component was chosen to help make the system operate more efficiently and be more responsive as a whole.

At the core of our project was a Trajectory algorithm, this is what we designed to control how the robot would move and make decisions. The technique was tested rigorously to maintain an accurate and fast-playing experience, employing many other situations for processing the algorithm.

1.4 Significance of our work

Our Air Hockey Robot project is not special just for the technical achievements but because it joins technology together with entertainment and practical uses. We have built an ultra-responsive, autonomous air hockey robot which can be used as: a training tool for players looking to hone their skills against challenging opposition; entertainment that casual players may enjoy playing; lucrative amusement options in the commercial sector.

Additionally, the project highlights the fusion of cutting-edge tech that includes on-the-fly image processing and high precision motor control to put robots at play in simple everyday leisure activities. Using the Trajectory algorithm to navigate movement of the robot shows our drive into creating advanced control systems that deliver an experience as real and engaging like no other.

In the end,our research really is pushing what can be done with autonomous robotics.

Chapter 2

Constraints and Earlier Coursework

2.1 Constraints

- High Cost of Electronic Components: The escalation of prices of electronic components gave us no choice, but to reduce the number of units required, thereby, not including some of the project's features and functionalities
- Power Supply Challenges: There was a great problem with this system because of the changed current demands at various places. So, we had the tests made of the power supply to be good for direct-current and voltage issues.
- Unexpected Failures: We had to find a solution to these problems by wasting time in repairing and diagnosing them. This, unfortunately, resulted in the late delivery of the project and had an adverse effect on the delivery status.
- Scarcity of Electronic Parts: The lack of some necessary electronic elements compelled us to look for substitutes that turned out to be not only more expensive but also less efficient. It also had a negative impact on the project budget and the performance of the whole project, too.
- Limited Design Experience: Our deficiency in design knowledge was the main reason we had to seek help from a different source. We allowed a designer to arrange the table and an engineer to execute the required part of the design, thus the project was competed in the way of our specifications.
- Time Constraints: The timetable of the project was affected by some local incidents like the strike action of workers and operation stop by the local regulators. These challenges led to significant delays, thus it is quite challenging to deliver the project on time.

2.2 Earlier Coursework

- Microcontroller and Its Applications: The main objectives of microcontroller applications, such as Arduino, sensor implementation, and hardware monitoring, were strengthened by this course.
- Electrical Circuits: We got hands-on in measuring current and voltage in series and parallel circuits, a very important aspect of designing and computing electrical systems.
- Image Processing: This course became a step to our project. So it trained object detection through computer vision techniques as well as the implementation of the Trajectory algorithm.
- Electronic Fundamentals: The class featured integrated circuits; we mainly discussed their operation, choice, resistance, and how to make them work better in different areas.

Chapter 3

Literature Review

The development of our Air Hockey Robot project is grounded in several key areas of research, particularly in trajectory algorithms and image processing techniques.

3.1 Trajectory Algorithm

The proper and timely movement of our air hockey robot depends on the correctness of trajectory prediction and assessment. The Trajectory algorithm we used is based on achieved research in trajectory completeness and prediction. Sharma and his colleagues in their study state that a pattern recog-nition algorithm that is robust is the one that first looks at overall trajectories [1]. Their study in thefield of transportation research underlines how accurate trajectory prediction in the dynamic systemsis the most essential thing. A concept that is the most suitable for our air hockey robot, wheresomeone has to make the correct and fast puck prediction for the game to be optimal. By utilizingand modifying these principles, we have successfully introduced an advanced algorithm that isable not only to analyze but also to make decisions based on real-time trajectory data, allowingthe robot to react well during air hockey matches.

3.2 Image Processing

Detection and tracking of the puck with heavy reliance on image processing is actually central to how our air hockey robot works, as suggested by what I just mentioned. Inspired by developments in image processing technologies such as the IMAGIC system utilized by Van [2]. The new generation of the IMAGIC image processing system where a very different nature of visual data is handled and analyzed evolved as they followed their earlier computational paths. Implementing such sophisticated image processing algorithms, our robot can detect and track the puck on the table with high accuracy in real time. This ability is crucial for the robot to

actively play with a puck, performing tactics based on real-time AI decision-making.

3.3 Conclusion

Our project seeks to bring the realms of what is possible in autonomous robotic gameplay by integrating these state-of-the-art algorithms and image processing techniques. The literature establishes the core; it provides us with resources to move ahead, duplicating current technologies along the way while also advancing new visions and breakthroughs

Chapter 4

Methodology

4.1 Design

4.1.1 Table Design

The table has a total length of 132 cm and a width of 65 cm, the playing surface is 100cm long and 60 cm wide as designed in the agreement.
The table's structure dimensions are specifically intended to help in the assembly of the 3D-printed elements and the aiming of the puck through the goals. The puck is deemed to have entered one of the two goals to mark the score.

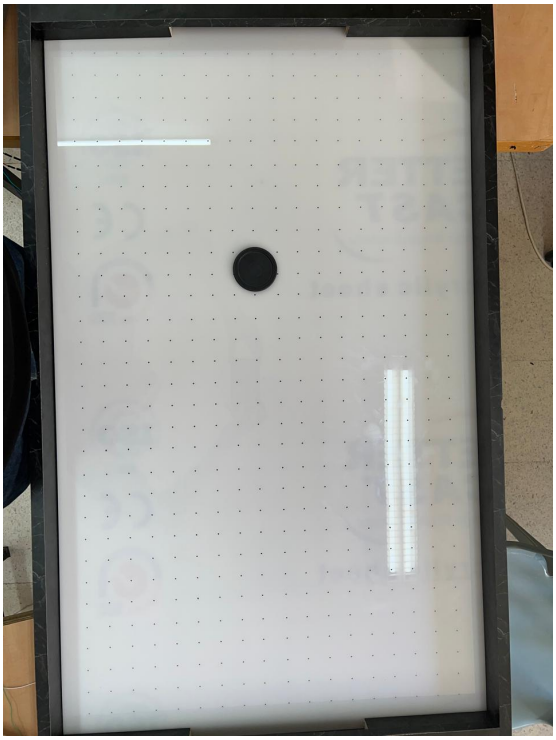

Figure 4.1: Design of the Table

The floor in the earlier picture was made with a CNC machine. Small holes were bored, each measuring two millimeters in diameter, to let air pass through and lessen friction between the puck and the surface. To help with this airflow, two fans were placed under the floor; they will be covered in more detail later.

Figure 4.2: Airflow Design of the Table

The table floor is made with tiny holes to let air flow through, as seen in picture 4.2. To make up for the small thickness of the surface, a bottom layer is added to the table to support the surface structurally and to disperse airflow equally throughout.

4.1.2 3D Printing Components

• X-axis motor carrier, that we use it to hold the motor that we have.

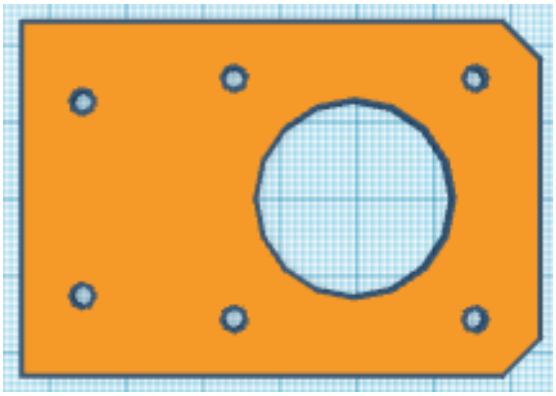

Figure 4.3: Component 1

• Y-axis motor carrier, this what we use to hold the motor that we have.

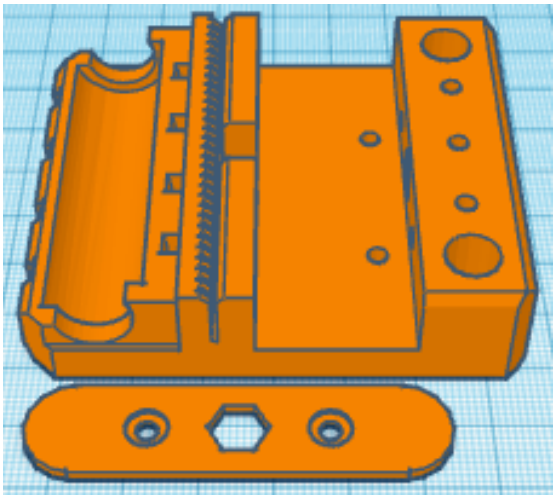

Figure 4.4: Component 2

• We use this piece to carry the motor on the y-axis, which is the motor for the x-axis, and it is also used to carry the time belt for the y-axis.

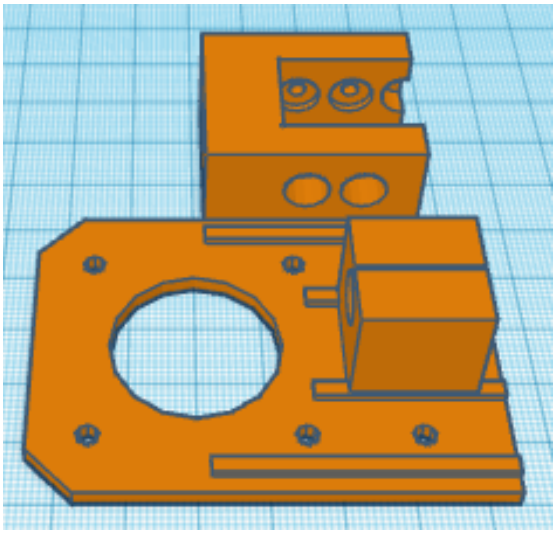

Figure 4.5: Component 3

• The racket that the robot uses to defend and attack.

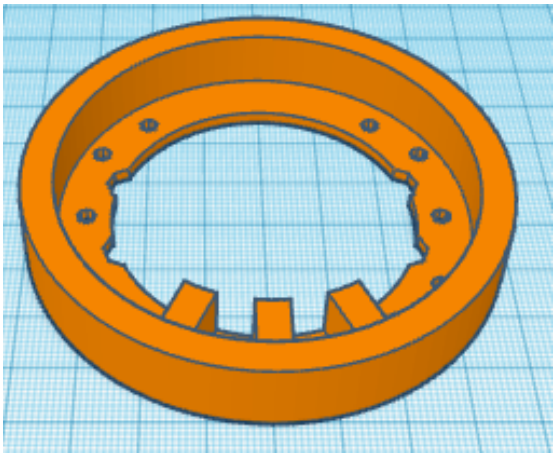

Figure 4.6: Component 4

• The remaining part of the bat that the robot uses to defend and attack.

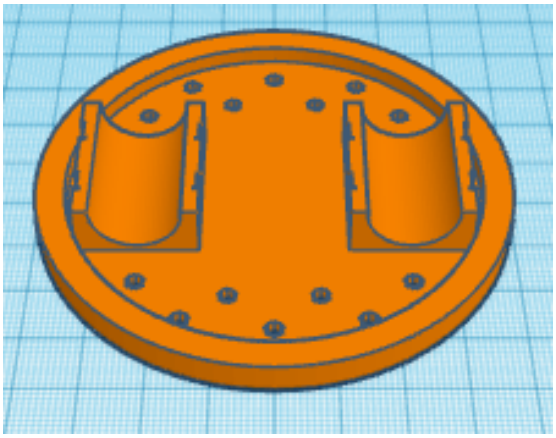

Figure 4.7: Component 5

• An essential piece in the project to place the timebelt on in the X-axis.

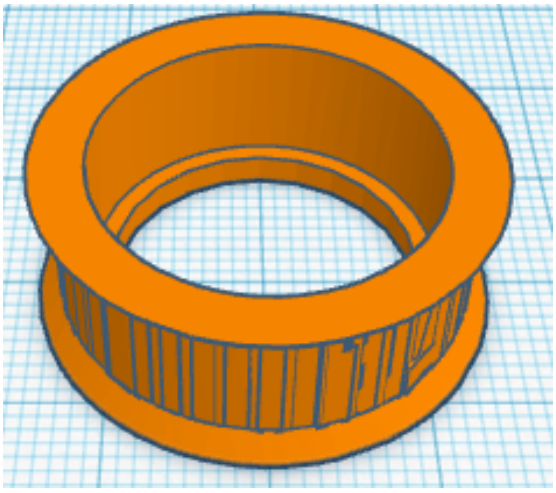

Figure 4.8: Component 6

• A piece that we install on the motor in order to place the time belt on it on the X axis.

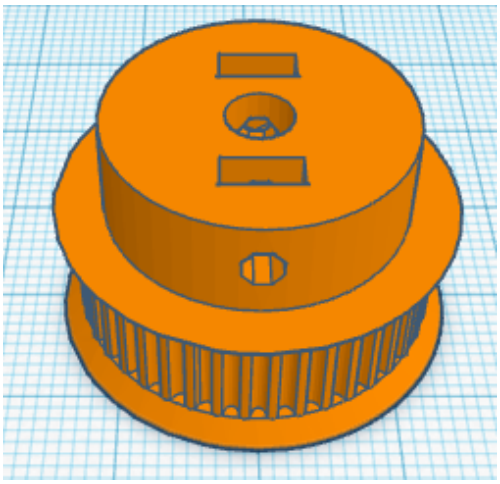

Figure 4.9: Component 7

4.2 Electronic Parts

4.2.1 Arduino UNO

An Arduino Uno board was used in the project to manage motors and sensors through the direct provision of precise control and the integration of different components. The setup allows the systems to connect to each other very easily and enables the sending of data and control of the two devices simultaneously. For maximizing the efficiency and having no latency, we partitioned the projects into two distinct sets of motors and sensors which allow more streamlining and less error. This method allows us to optimize our resources and take immediate action in the system by following this procedure.

Involving Arduino in this project is very critical for controlling the motors and the sensors based on the input signals from the Raspberry Pi. The communication between the Arduino and Raspberry Pi is handled via serial communication. These are the main processes that the Arduino deals with:

- Serial data communication: The Arduino, which is used for the control and monitoring system of the air-hockey game, is the one that picks up the signals coming from the Raspberry Pi through the serial port. These signals contain the movements of the motor and data for the sensors and through them, the system can make decisions in real-time for the hockey table.
- Receiving Commands: The Arduino receives a series of commands from the Raspberry Pi that involves elements such as the motor speed, direction, and sensor readings. It does this by first sending them through a parser that gets the relevant information
- Motor Control: The Arduino is responsible for converting the movement of the motors to match the commands that were parsed. These motors are the ones used in the movements

of the paddle to and fro the air hockey table's puck.

- Real-time Response: For quick response times, the Arduino lie with motor priority and then with sensor readings. Thus, the progress of the game is maintained smooth and much more reactive than usual. Also, most importantly, the machinery's response time is kept shorter, as long as it is in the actual game and in the real world.

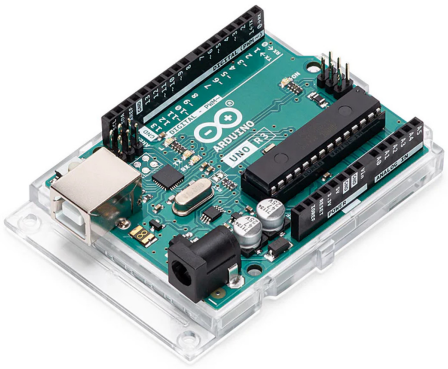

Figure 4.10: Arduino UNO

4.2.2 Raspberry PI

The Raspberry Pi acts as the brain of the project, responsible for, among other tasks, image processing, decision-making, and motor coordination within the air hockey table robot system. It collaborates also with the OAK-D camera and Arduino to determine where the puck is, and the motors are thus controlled.

The list of functions that the Raspberry Pi performs is long:

- Puck Detection and Tracking: The Raspberry Pi captures a real-time video feed of the air hockey table using the OAK-D camera. The camera specifically detects the green circular puck and the orange robot on the table. Through image processing algorithms, the OAK-D camera identifies these objects and tracks their movement across the table. It then returns the positions of both the puck and the robot in pixel coordinates relative to the frame of the video feed. These pixel positions are crucial for further calculations, such as converting pixel distances to centimeters and ultimately determining the motor steps required for movement.
- Trajectory Prediction: It predicts the path using a predefined algorithm on Raspberry Pi to calculate positions of forts. what the puck is going to do next, given its speed and position. This prediction allows best practice — the system will not be able to check for where it should move robot paddle, intercept the puck.

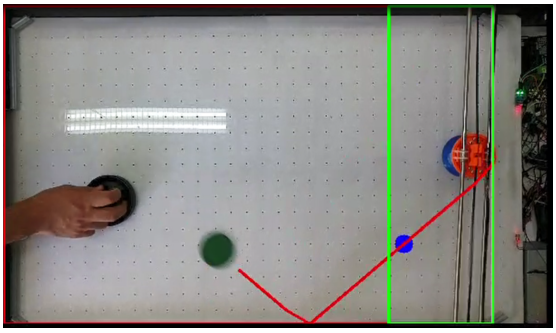

Figure 4.11: predection

- Motor Control through Step Calculation:

  The Raspberry Pi is responsible for determining the movement required to position the robot's paddle in response to the puck's movement. This process involves calculating the puck's position relative to the robot, converting these pixel values to centimeters, and then determining the motor steps needed to move the paddle. The steps involved are as follows:

  1. **Image Processing and Position Calculation:**
     - The first step involves obtaining the position of the puck and the robot in pixels.
     - To convert the pixel values to centimeters, the following equations are used:

$$\text{diffXPixels} = \text{puck\_Xposition} - \text{robot\_Xposition}$$

$$\text{pixelsXToXcm} = \text{round}\left(\frac{\text{diffXPixels}}{5.9}\right)$$

       where 5.9 is the number of pixels per centimeter on the X axis.
  2. **Step Conversion for X Axis:**
     - The distance in centimeters is then converted into the number of motor steps needed to move the paddle along the X axis:

$$\text{cmXToSteps} = \text{round}(\text{pixelsXToXcm} \times 192.3)$$

       where 192.3 is the number of motor steps required to move 1 cm along the X axis.
  3. **Step Conversion for Y Axis:**
     - For the Y axis, the pixel distance is converted to centimeters and then to motor steps using a different conversion factor:

$$\text{cmYToSteps} = \text{round}(\text{pixelsYToXcm} \times 200)$$

       where 200 is the number of motor steps required to move 1 cm along the Y axis, reflecting the difference in motor specifications.

4. **Calculate Direction and Speed:**
   - After determining the necessary steps for both axes, the Raspberry Pi calculates the direction and speed for each motor to ensure accurate and timely movement to the target position. This includes adjusting the speed to match the required movement precision.

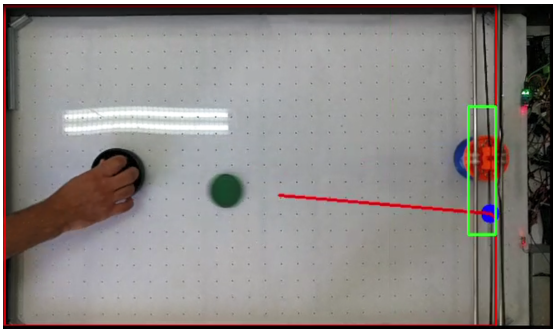

Figure 4.12: Defense Mode

This combination of image processing and motor control ensures that the robot can track the puck's movement and respond effectively during gameplay, enabling dynamic and intelligent interactions.

- Automatic Defense Mode: When the puck's velocity exceeds a certain threshold, the Raspberry Pi automatically switches the robot to a defense mode. In this mode, the robot focuses solely on defending the goal by limiting its movements along the X-axis. This ensures that the robot effectively blocks high-speed pucks from entering the goal, prioritizing defense over interception.
- Communication with Arduino: Once the steps and direction are calculated, the Raspberry Pi sends the corresponding commands to the Arduino over serial communication. These commands include the number of steps to move, the direction, and the speed of the movement. The Arduino then controls the motors based on these commands.
- Real-time Adjustments: As the game progresses, the Raspberry Pi continuously receives updates from the OAK-D camera, recalculating the puck's trajectory and updating the motor control instructions to ensure timely movements of the paddle.

Figure 4.13: Raspberry PI

4.2.3 OAK-D Camera

In the OAK-D camera, it is involved in capturing the motion of the puck on the table. This captures the 3D images in real time and works on the images to determine the position and movement of the puck. This information is then transmitted to the Raspberry Pi where it computes the appropriate movement of the motors to get closer to the puck. This way, the camera provides the necessary precision to track the game and give quick and efficient responses to the robot during the game.

Figure 4.14: OAK-D Camera

4.2.4 Stepper Motor

For the motion control of the robot along X and Y axis, we employed NEMA 17 stepper motors where two motors were assigned for the Y axis while one motor for the X axis. As for the motor, the NEMA 17 was selected because it can deal with loads up to five kilograms even though our project does not require the necessity to lift this amount of weight. But it is the smallest stepper motor that is currently available which has enough torque for our application.

A DC motor was not employed due to chain the requirement of achieving high accuracy with the number of steps, out of which the motor could be accurately positioned according to the step commands sent by the Arduino.

Figure 4.15: NEAM 17

4.2.5 Driver 3.5A

We used a 3. 5 amp driver to deliver the amount of current required by the motor and which is DC 1. 7 amps. This driver also gives us the ability to fine-tune the steps per cycle and as a result, has a motion control capability. The motor is of operational type and requires 12 volts for its proper functioning and it has pins for controlling the direction, operation and steps. The motor has four wires and they wire is associated with the coils of the motor are joined to the driver. The actuators built into the stepper motor are four coils and the full rotation of the motor is 200 steps in which each a step is 1. 8 degrees.

Figure 4.16: Driver 3.5

4.2.6 Fan

In the picture 4.2, you can see that we used 12V DC fans to blow air over the CNC machined surface. Two fans were operated at the bottom end of the air hockey table to constantly blow air over the table; this is beneficial for the puck's movements on the table because the air eliminates contact between the puck and the hockey table.

Figure 4.17: FAN

4.2.7 Joystick

For the next step, we added a joystick as the new feature in the project; thus, offering the new mode for the gameplay. This allows the system to operate in two modes: it is one where the entire play is done by the robot without much human intervention with the OAK-D camera for puck recognition and movement while in the other a real player can fully control the robot using the joystick for a more interactively enhanced gameplay.

Figure 4.18: Joystick

4.2.8 Laser

To identify a goal, we pointed at a laser towards the LDR sensor and then interrupting it using the position of the puck.

Figure 4.19: Laser

4.2.9 LDR

We positioned an LDR at the goal entrance with a laser placed on it on one wall. When the beam is in place, the LDR gives an equivalent HIGH digital output reading. When laser is broken, this means it detected a goal, the LDR reading goes back to LOW.

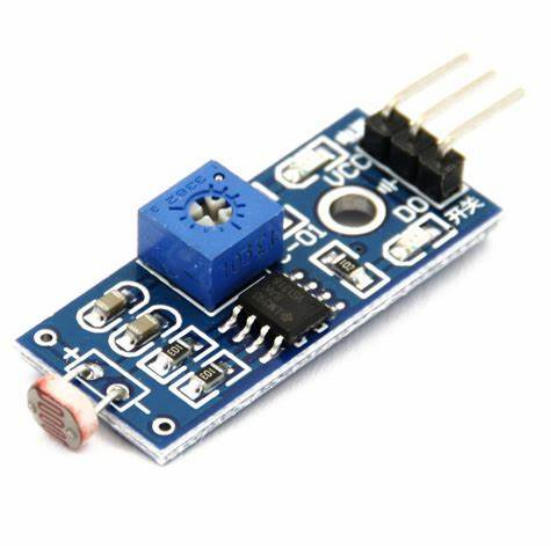

Figure 4.20: LDR Module

4.2.10 Power Supply

In powering the motors and fans, we used a 12 volt power supply with with 9 amps capacity. For sensors such as LDR, DFPlayer, laser, 5 volt wires were used and the ground was connected in unison to the arduino ground.

Figure 4.21: Power Supply

4.2.11 DF Player And Speaker

A voice commentary, just like those used in sporting events, has been added via the DFPlayer whereby music and recorded audio is stored on an SD card and sent to the speakers. When a fight begins, a voice introductory message is heard and each time a target is hit, some particular comments are made.

Figure 4.22: DF Player And Speaker

4.2.12 Seven Segment

A 4-digit seven-segment display was implemented to show the continuing score between the robot and the player. Two digits are assigned to each player to make the detection of each player's score easier. It is an automated process, with the detection of a goal by the LDR sensor automatically updating the score of the player concerned. This setup not only ensures

that the scores are current but also adds an easily visible score for the competitive nature of game plays.

Figure 4.23: Seven Segment

4.3 Hardware Components

4.3.1 Axis Rod

We took axis rod and connected our 3D-printed parts through it; this allowed our X-axis and Y-axis motors to move on it. This rod provides a critical structural element for proper movement with accuracy regarding the motors that are supposed to move on both axes.

Figure 4.24: Axis Rod

4.3.2 Linear Motion Ball Bearing

This component was used to allow the rod to pass through it, in order to enable parts depending on that rod to move. It ensures that the moving parts attached to the components can glide smoothly along the rod and that it is possible to guide and move them in a controlled way.

Figure 4.25: Linear Motion

4.3.3 Timing Belt

For the motion of the 3-D pieces, we used the timing belt, whose one end was attached to the motor and the other end to a pulley. This enables easy circular motions, hence giving us great control in moving our components.

Figure 4.26: Timing Belt

4.3.4 Wires

The most important part of the project involves the connection of all electronic components with the Raspberry Pi and Arduino. All these will ensure effective communication between sensors, motors, and other components to enable the Raspberry Pi to process data and send commands to the Arduino, which controls the hardware.

Figure 4.27: Wires

4.3.5 Sweep Window Seal

We used this part to bounce the puck quickly allowing for smooth and fun play.

Figure 4.28: Window Seal

4.4 Mobile Application

This control system included a React Native app, a Flask backend, Raspberry Pi, and Arduino for smooth transitions between the modes: robot mode and joystick mode. To this end, the

Raspberry Pi communicated with the Arduino to implement mode changes and control the stepper motors. The introduction of minimal delay in mode changes was quite easy for the users using the React Native app, as its interface is friendly and intuitive.

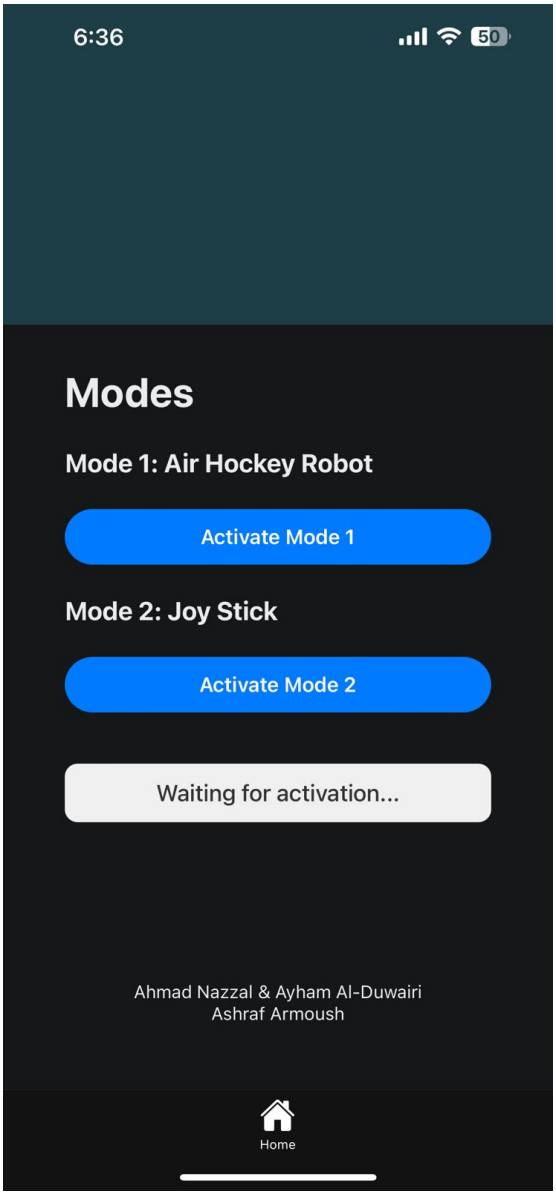

Figure 4.29: Wires

Chapter 5

Results and Discussion

This section evaluates the air hockey robot's performance and functionality across its two operating modes (robot mode and joystick mode), scoring system, sound feedback, and integrated control system comprising a React Native app, Flask backend, Raspberry Pi, and Arduino.

5.1 Robot Mode

In robot mode, the air hockey robot autonomously tracks and reacts to the puck's movement using an image processing system. During testing, the robot was able to anticipate and block puck movements and occasionally score when positioned advantageously.
However, during high-speed gameplay, the robot showed some limitations, with occasional delays in detecting and responding to faster puck movements. This points to a need for optimization in the image processing algorithms to enhance real-time performance and increase the robot's agility under faster conditions. Refining these algorithms could lead to significant improvements in overall performance during high-speed play.

5.2 Joystick Mode for Accessibility

The joystick mode allows for the manual control of the robot and was intended to make the game more accessible, especially to people with mobility impairments. Players reported that the joystick was responsive and easy to control, hence allowing a satisfying experience in playing.
On the other hand, some players' feedback indicated that the sensitivity of the joystick could be better adjusted so as to enhance the continuity with precision even in the most intense moment in the game. This would, in actual sense, make it quite easier for all players, particularly in highly competitive environments, to switch from rapid movements more fluently.

5.3 Scoring System with LDR and Laser Detection

The scoring system for the air hockey robot makes use of the LDR module and a laser to identify the intrusion of the puck in the goal. Working this out, it went perfectly during the games at moderate speed, changing the score on the 7-segment display without problems.
Although generally quite reliable at refular speeds, unfortunately, the system would sometimes fail to detect high-speed pucks due to the response time limitations of the LDR. This would affect the accuracy of score updates during faster gameplay. In future iterations, alternative detection technologies with faster response times could be explored in order to achieve greater reliability under all conditions.

5.4 Sound Feedback System

In the air hockey robot, there was designed a sound feedback mechanism with the voice of an Arabic-speaking soccer commentator when a player scores a goal. This added so much more to the overall feel and made for some exciting and culturally relevant audio experiences as players listened for a change in scores. The speaker turned out to be very reliable during the play and gave clear, impactful sound. This feature indeed brought an uptick in liveliness with regard to the scoring moments of the game because it made the experience of participating in such a tournament quite engaging and interesting.

5.5 Control System Integration (React Native, Flask, Raspberry Pi, and Arduino)

In detail, it is a control system consisting of a React Native app, a Flask backend, a Raspberry Pi, and Arduino together allowed seamless mode switching between the robot mode and the joystick mode. The Raspberry Pi Also, it connected Arduino to change modes and to control stepper motors. The user can easily go between modes with small latency with a super intuitive and user-friendly React Native app.
The system was quite reliable during testing: mode transitions were seamless. However, there were a few instances of time delays in communication between the Raspberry Pi and Arduino; this could be due to a number of factors, such as processing load or network latency. A prudent optimization of the protocols used in communication between these two devices would decrease latency further in order to improve system's responsiveness.

This air hockey robot really provided an interactive and easy gameplay experience by effectively integrating the React Native app, Flask backend image-processing system, and hardware. Mode switching between robot mode and joystick mode was seamless, and the performance of the system proved to be effective in both autonomous and manual control scenarios. The scoring system combined with sound feedback and a control interface for an immersive experience. This is shown by the combination of automation with user control new paradigm that balanced functionality with accessibility.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The air hockey robot project demonstrated the use of advanced technology and creative engineering in an engaging, fun overall product. Using the capabilities of the Raspberry Pi, the OAK-D camera, and the Arduino, it was possible to track the position of the puck, compute the predicted path, and control the motors and collimators to block and intercept the puck. A final feature that was coded was an automatic defense mode. If the gameplay was fast-paced and the robot was unable to hit the puck back to the opponent's side, the robot would immediately return to the defense position (using the position set by the user) and would only react to the opponent's puck. This made the robot really play intelligently. This project is a great example of how the capabilities of carefully designed hardware can be used to implement effective software, and vice versa, resulting in a truly interactive, responsive robotic system that behaves almost like a life-form.

6.2 Future Work

While the air hockey robot performed well in various scenarios, several areas for future improvement have been identified:

- Enhanced Puck Detection and Tracking: Future iterations of the project could explore the use of more advanced image processing techniques or machine learning models to further improve the accuracy and speed of puck detection and tracking, especially during high-speed gameplay.
- Refinement of Defense Mode: The automatic defense mode could be refined to dynamically adjust its sensitivity based on the puck's speed and direction, potentially incorporating predictive algorithms to better anticipate the puck's trajectory.
- Improved Joystick Sensitivity: Enhancing the sensitivity and responsiveness of the joy-

stick controls could provide a smoother and more precise manual control experience, particularly during fast-paced gameplay.

- Enhanced Sound Feedback: The sound feedback system could be expanded to include more varied and context-sensitive audio cues, possibly incorporating additional languages or customizable sounds to cater to different player preferences.
- Integration with AI for Strategic Play: Implementing AI-based strategies that allow the robot to learn from gameplay and adjust its tactics could further enhance the robot's performance, making it more challenging and engaging for human players.

These enhancements would not only improve the robot's overall performance but also broaden its applicability in educational, entertainment, and accessibility contexts. The project has laid a solid foundation, and with continued development, it holds the potential for even more sophisticated and innovative robotic systems.

Bibliographic

[1] Anshuman Sharma, Zuduo Zheng, and Ashish Bhaskar. “A pattern recognition algorithm for assessing trajectory completeness”. In: *Transportation research part C: emerging technologies* 96 (2018), pp. 432–457.

[2] Marin van Heel et al. “A new generation of the IMAGIC image processing system”. In: *Journal of structural biology* 116.1 (1996), pp. 17–24.

Appendix

A Main Code for image processing

```
import cv2
import numpy as np
import depthai as dai
import serial
import time
import cv2

# arduino = serial.Serial(port='/dev/ttyV1', baudrate=9600, timeout=1)
arduino = serial.Serial(port='/dev/ttyACM1', baudrate=9600, timeout=1)
# arduino = serial.Serial(port='/dev/ttyACM0', baudrate=9600, timeout=1)
time.sleep(2)
#HSV
lower_green = np.array([35, 50, 30])
upper_green = np.array([80, 255, 255])
lower_orange = np.array([0, 200, 150])
upper_orange = np.array([15, 255, 255])

# to start it imediatly
attackFlag = 2
returnFlag = 2
stableAttackFlag = 2

# def play_sound(file):
#     playsound(file)
```

```
# def play_sound_if_triggered():
#     if arduino.in_waiting > 0:
#         print("sound on")
#         data = arduino.readline().decode().strip()
#         if data == '1':
#             # playsound(r'D:\University\University_materials\Fourth Year\Summer\Graduation Pr
#             # Trigger sound asynchronously
#             with concurrent.futures.ThreadPoolExecutor() as executor:
#                 executor.submit(play_sound, r'D:\University\University_materials\Fourth Year\


def detect_green_circle(frame):
    frame_resized = cv2.resize(frame, (640, 360))

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2HSV)

    mask = cv2.inRange(hsv, lower_green, upper_green)
    masked_frame = cv2.bitwise_and(frame_resized, frame_resized, mask=mask)
    #cv2.imshow('Masked Frame (Green Color)', masked_frame)
    blurred = cv2.GaussianBlur(mask, (9, 9), 2)

    # detect circles
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1.2, minDist=50, param1=50, para

    if circles is not None:
        # Convert (x, y, radius) to integers
        circles = np.round(circles[0, :]).astype("int")
        return circles[0]

    return None

def detect_orange_circle(frame):
    frame_resized = cv2.resize(frame, (640, 360))

    hsv = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2HSV)
```

```
    mask = cv2.inRange(hsv, lower_orange, upper_orange)

    orange_regions = cv2.bitwise_and(frame_resized, frame_resized, mask=mask)

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Initialize variables to hold the x, y position
    x, y = None, None

    if contours:
        # Find the largest contour (assuming it's the target object)
        largest_contour = max(contours, key=cv2.contourArea)

        # Compute the centroid of the largest contour
        M = cv2.moments(largest_contour)
        if M["m00"] != 0:
            x = int(M["m10"] / M["m00"])
            y = int(M["m01"] / M["m00"])

            # Draw a circle at the centroid
            cv2.circle(frame_resized, (x, y), 5, (255, 0, 0), -1)

    #cv2.imshow("Orange Regions", orange_regions)
    #cv2.imshow("Detected Orange Centroid", frame_resized)
    return x, y
# Predict the puck's path (table borders, movement from left to right)
def predict_puck_path(puck_position, puck_velocity, border_limits, orange_puck_position = (0,0)
    global returnFlag
    predicted_path = []
    x, y, vx, vy = puck_position[0], puck_position[1], puck_velocity[0], puck_velocity[1]
    if( vx <= 0):
        returnFlag += 1
        if(returnFlag > 4):
            send_command(convertPixelsToSteps(180, orange_puck_position[1],570,orange_puck_posi
            print("**********************")
            print("orange_puck_positionY ",orange_puck_position[0])
            returnFlag = 0
```

```
            return predicted_path
        for _ in range(num_predictions):
            x += vx * time_step
            y += vy * time_step

            # collisions with table borders
            # if x <= border_limits[0] or x >= border_limits[2]:
            #     vx = -vx
            if y <= border_limits[1] or y >= border_limits[3]:
                vy = -vy

            # Update position
            x = max(border_limits[0], min(x, border_limits[2]))
            y = max(border_limits[1], min(y, border_limits[3]))

            # Only predict path if puck is moving from left to right
            # print(vx)
            predicted_path.append((x, y))



        return predicted_path

# capture video and detect green puck
def main():
    global attackFlag
    global returnFlag
    global stableAttackFlag
    pipeline = dai.Pipeline()

    camRgb = pipeline.create(dai.node.ColorCamera)
    camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
    camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
    camRgb.setInterleaved(False)
    camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

    # output
    xoutVideo = pipeline.create(dai.node.XLinkOut)
```

```
xoutVideo.setStreamName("video")
camRgb.video.link(xoutVideo.input)

puck_position = None
puck_velocity = (0, 0)
table_size = (640, 360)  # Define the actual table size
# from the camera view x left y top x right y bottom
border_limits = (40, 10, 580, 360)  # Red border limits start form the human place
attacking_area = (465, 10, 580, 360)  # Green attacking area limits
# attacking_area = (570, 10, 600, 360)  #  defends

previous_time = time.time()

with dai.Device(pipeline) as device:
    qVideo = device.getOutputQueue(name="video", maxSize=30, blocking=True)
    tempX=0
    flagX=False
    while True:
        # play_sound_if_triggered()
        inVideo = qVideo.get()
        frame = inVideo.getCvFrame()
        frame_resized = cv2.resize(frame, (640, 360))

        # Detect green puck and get its position x y radius
        detected_puck = detect_green_circle(frame)
        orange_puck_position = detect_orange_circle(frame)

        current_time = time.time()
        time_delta = current_time - previous_time
        previous_time = current_time
        # detected_puck is the new position
        # puck position is the previuos position
        if detected_puck is not None: # x y radius
            if puck_position is not None: # x y
                # Calculate velocity (simple difference for now, could be improved)
                # puck_velocity = (detected_puck[0] - puck_position[0], detected_puck[1] -
                puck_velocity = puck_velocity = ((detected_puck[0] - puck_position[0]) / ti
```

```
                                    (detected_puck[1] - puck_position[1]) / time_delta)
                    # print("puck velocity",puck_velocity[0],puck_velocity[1])
##################################################
                    #send when the puck is stopped
                    # if puck_position[1] == tempX:
                    #     if flagX:
                    #         print(puck_position[1])
                    #         send_command(convertPixelsToSteps(puck_position[1]))
                    #         flagX = False
                    # else:
                    #     tempX = puck_position[1]
                    #     flagX = True
                    distance_moved = np.sqrt((detected_puck[0] - puck_position[0])**2 + (detect
                    # Only update path prediction if the puck has moved more than 5 cm (30 pixe
                    if distance_moved > 30 * table_size[0] / 640:
                        puck_position = detected_puck[:2]
                        puck_position_scaled = (puck_position[0] * table_size[0] / 640, puck_po
                        puck_velocity_scaled = (puck_velocity[0] * table_size[0] / 640, puck_ve
                        predicted_path = predict_puck_path(puck_position_scaled, puck_velocity_
                        if(puck_velocity_scaled[0] > 1200):
                            # defend
                            attacking_area = (550, 120, 580, 260)
                        else:
                            # attack
                            attacking_area = (465, 10, 580, 360)

                        # Find the position in the predicted path that lies within the attackin
                        attack_position = None
                        for pos in predicted_path:
                            if attacking_area[0] <= pos[0] <= attacking_area[2] and attacking_a
                                attack_position = pos
                                break

                        if attack_position:
                            # Draw the attack position as a circle
                            attack_position_scaled = (int(attack_position[0] * 640 / table_size
                            cv2.circle(frame_resized, attack_position_scaled, 10, (255, 0, 0),
```

```
                            attackFlag+=1
                            if(attackFlag==3):
                                send_command(convertPixelsToSteps(attack_position_scaled[1], or
                                attackFlag=0



                        # Draw the predicted path as a solid red line
                        if len(predicted_path) > 1:
                            for i in range(1, len(predicted_path)):
                                pt1_scaled = (int(predicted_path[i-1][0] * 640 / table_size[0])
                                pt2_scaled = (int(predicted_path[i][0] * 640 / table_size[0]),
                                cv2.line(frame_resized, pt1_scaled, pt2_scaled, (0, 0, 255), 2)
                    # if the puck in the area of the robot
                    elif(detected_puck[0] > 400 and detected_puck[0] < 540):
                        stableAttackFlag+=1
                        if(stableAttackFlag==7):
                            send_command(convertPixelsToSteps(detected_puck[1], orange_puck_pos
                            stableAttackFlag=0

                else:
                    puck_position = detected_puck[:2]

            # Draw table borders
            cv2.rectangle(frame_resized, (border_limits[0], border_limits[1]), (border_limits[2
            # Draw attacking area
            cv2.rectangle(frame_resized, (attacking_area[0], attacking_area[1]), (attacking_are

            #cv2.imshow('Green Puck Detection and Path Prediction', frame_resized)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    cv2.destroyAllWindows()


def send_command(command):
```

```
    if isinstance(command, int):
        command = str(command)
    arduino.write((command + '\n').encode())
    arduino.flush()
    # time.sleep(0.05)

# XdefualtPostion= 180
def convertPixelsToSteps(puck_Xposition, orange_Xposition,puck_Yposition,orange_Yposition,sourc
    if puck_Xposition is None or orange_Xposition is None or puck_Yposition is None or orange_Y
        print("Error: One or both positions are None.")
        return 0
    if(puck_Xposition > 320):
        print("positions ///",orange_Xposition,puck_Xposition)
        puck_Xposition = 320
    elif(puck_Xposition < 50):
        print("positions ---",orange_Xposition,puck_Xposition)
        puck_Xposition = 50
    diffXPixels= puck_Xposition-orange_Xposition
    pixelsXToXcm = round(diffXPixels / 5.9)
    cmXToSteps = round( pixelsXToXcm * 192.3)

    diffYPixels= puck_Yposition-orange_Yposition
    pixelsYToXcm = round(diffYPixels / 5.9)
    cmYToSteps = round( pixelsYToXcm * 200)
    XY = str(cmXToSteps)+"*"+str(cmYToSteps)
    print(XY)
    return  XY

if __name__ == '__main__':
    main()
```

B Code for the Backend Server (flask)

```
from flask import Flask, jsonify
import RPi.GPIO as GPIO

app = Flask(__name__)
GPIO.setmode(GPIO.BCM)
pin = 23
GPIO.setup(pin, GPIO.OUT)

@app.route('/mode1', methods=['POST'])
def mode1():
    GPIO.output(pin, GPIO.HIGH)
    return jsonify({'status': 'Mode 1 activated'})

@app.route('/mode2', methods=['POST'])
def mode2():
    GPIO.output(pin, GPIO.LOW)
    return jsonify({'status': 'Mode 2 activated'})

    arduino.flush()
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

C Code for the react native app

```
import { StyleSheet, View, Button, TouchableOpacity, Text } from 'react-native';
import ParallaxScrollView from '@/components/ParallaxScrollView';
import { ThemedText } from '@/components/ThemedText';
import { ThemedView } from '@/components/ThemedView';
import React, { useState } from 'react';

export default function HomeScreen() {
  const [data, setData] = useState('');

  const handleMode1Press = () => {
    fetch('http://192.168.137.92:5000/mode1', {
      method: 'POST',
    })
      .then((response) => response.json())
      .then(() => {
        setData('Mode 1 activated');
        console.log('Mode 1 activated');
      })
      .catch((error) => {
        console.error('Error activating Mode 1:', error);
      });
  };

  const handleMode2Press = () => {
    fetch('http://192.168.137.92:5000/mode2', {
      method: 'POST',
    })
      .then((response) => response.json())
      .then(() => {
        setData('Mode 2 activated');
        console.log('Mode 2 activated');
      })
      .catch((error) => {
        console.error('Error activating Mode 2:', error);
      });
  };
```

```
  return (
    <ParallaxScrollView headerBackgroundColor={{ light: '#A1CEDC', dark: '#1D3D47' }}>
      <ThemedView style={styles.titleContainer}>
        <ThemedText type="title">Modes</ThemedText>
      </ThemedView>
      <ThemedView style={styles.stepContainer}>
        <ThemedText type="subtitle">Mode 1: Air Hockey Robot</ThemedText>
        <TouchableOpacity style={styles.button} onPress={handleMode1Press}>
          <Text style={styles.buttonText}>Activate Mode 1</Text>
        </TouchableOpacity>
        <ThemedText type="subtitle">Mode 2: Joy Stick</ThemedText>
        <TouchableOpacity style={styles.button} onPress={handleMode2Press}>
          <Text style={styles.buttonText}>Activate Mode 2</Text>
        </TouchableOpacity>
        <View style={styles.dataContainer}>
          {data ? (
            <Text style={styles.dataText}>{data}</Text>
          ) : (
            <Text style={styles.dataText}>Waiting for activation...</Text>
          )}
        </View>
      </ThemedView>
      <View style={styles.footer}>
        <Text style={styles.footerText}>Ahmad Nazzal & Ayham Al-Duwairi</Text>
        <Text style={styles.footerText}>Ashraf Armoush</Text>
      </View>
    </ParallaxScrollView>
  );
}

const styles = StyleSheet.create({
  titleContainer: {
    flexDirection: 'row',
    alignItems: 'center',
    gap: 8,
    paddingHorizontal: 16,
```

```
    paddingVertical: 8,
  },
  stepContainer: {
    gap: 16,
    marginBottom: 16,
    paddingHorizontal: 16,
  },
  button: {
    width: '100%',
    paddingVertical: 12,
    backgroundColor: '#007BFF',
    borderRadius: 25,
    alignItems: 'center',
    marginVertical: 8,
  },
  buttonText: {
    color: '#FFFFFF',
    fontSize: 16,
    fontWeight: '600',
  },
  dataContainer: {
    marginTop: 16,
    paddingHorizontal: 16,
    paddingVertical: 12,
    backgroundColor: '#f0f0f0',
    borderRadius: 10,
  },
  dataText: {
    fontSize: 18,
    textAlign: 'center',
    color: '#333333',
    fontWeight: '500',
  },
  footer: {
    marginTop: 80,
    paddingVertical: 12,
    paddingHorizontal: 16,
```

```
  },
  footerText: {
    color: '#ECF0F1',
    fontSize: 14,
    textAlign: 'center',
  },
});
```

D Arduino Code for controlling the motors and the modes

```
const int stepPinLeft = 2;  // Step pin for left motor
const int dirPinLeft = 3;   // Direction pin for left motor
const int dirPinRight = 4;  // Direction pin for right motor
const int stepPinRight = 5; // Step pin for right motor
const int dirPinX = 6;      // Direction pin for X-axis motor
const int stepPinX = 7;     // Step pin for X-axis motor
const int enablePinLeft=8;
const int enablePinRight=9;
const int enablePinX=10;
const int ldrSensor = 11;
const int modeFlag = 13;

volatile int XAxisSteps = 0;
volatile int YAxisSteps = 0;
#define VRX_PIN  A0
#define VRY_PIN  A1

int xValue = 0; // To store value of the X axis
int yValue = 0; // To store value of the Y axis

int joyStickCounter = 0; // to read analog every #

void setup() {
  // Set pin modes
  pinMode(dirPinLeft, OUTPUT);
  pinMode(stepPinLeft, OUTPUT);
```

```
  pinMode(dirPinRight, OUTPUT);
  pinMode(stepPinRight, OUTPUT);
  pinMode(dirPinX, OUTPUT);
  pinMode(stepPinX, OUTPUT);
  pinMode(enablePinLeft, OUTPUT);
  pinMode(enablePinRight, OUTPUT);
  pinMode(enablePinX, OUTPUT);
  pinMode(ldrSensor, INPUT);

  pinMode(VRX_PIN, INPUT);
  pinMode(VRX_PIN, INPUT);
  pinMode(modeFlag, INPUT);

  digitalWrite(dirPinLeft, LOW);
  digitalWrite(dirPinRight, LOW);
  digitalWrite(dirPinX, HIGH);

  digitalWrite(enablePinLeft, LOW);
  digitalWrite(enablePinRight, LOW);
  digitalWrite(enablePinX, LOW);


  Serial.begin(9600);
}
void loop() {

if(digitalRead(modeFlag) == LOW) {
  xValue = analogRead(VRX_PIN);
  yValue = analogRead(VRY_PIN);
  digitalWrite(enablePinLeft,LOW);
  digitalWrite(enablePinRight,LOW);
  digitalWrite(enablePinX,LOW);
  if (xValue > 550) {
    digitalWrite(dirPinX, LOW);
  }
  else {
    digitalWrite(dirPinX, HIGH);
```

```
}
if(yValue > 550) {
  digitalWrite(dirPinLeft, HIGH);
  digitalWrite(dirPinRight, HIGH);
}
else {
  digitalWrite(dirPinLeft, LOW);
  digitalWrite(dirPinRight, LOW);
}

if(!((xValue > 450 && xValue < 550) && (yValue > 450 && yValue < 550))) {

  if((yValue < 450 || yValue > 550) && (xValue < 450 || xValue > 550))
  {
    for(int i=0 ; i<200 ; i++)
    {
    digitalWrite(stepPinLeft, HIGH);
    delayMicroseconds(15);
    digitalWrite(stepPinLeft, LOW);
    delayMicroseconds(15);

    digitalWrite(stepPinRight, HIGH);
    delayMicroseconds(15);
    digitalWrite(stepPinRight, LOW);
    delayMicroseconds(15);

    digitalWrite(stepPinX, HIGH);
    delayMicroseconds(20);
    digitalWrite(stepPinX, LOW);
    delayMicroseconds(20);

    }

  }
  else if((yValue < 450 || yValue > 550)) {
    for(int i=0 ; i<200 ; i++)
    {
```

```
        digitalWrite(stepPinLeft, HIGH);
        delayMicroseconds(30);
        digitalWrite(stepPinLeft, LOW);
        delayMicroseconds(30);

        digitalWrite(stepPinRight, HIGH);
        delayMicroseconds(30);
        digitalWrite(stepPinRight, LOW);
        delayMicroseconds(30);

      }
    }

    else if((xValue < 450 || xValue > 550)) {
      for(int i=0 ; i<200 ; i++)
      {
        digitalWrite(stepPinX, HIGH);
        delayMicroseconds(45);
        digitalWrite(stepPinX, LOW);
        delayMicroseconds(45);

      }
    }
  }

}
else if(digitalRead(modeFlag) == HIGH) {
 if (Serial.available() > 0) {

    String command = Serial.readStringUntil('\n');
    int asteriskIndex = command.indexOf('*');
    String firstPart = command.substring(0, asteriskIndex);
    XAxisSteps = firstPart.toInt();

    String secondPart = command.substring(asteriskIndex + 1);
    YAxisSteps = secondPart.toInt();
      //10000 max steps on Xaxis
```

```
    //5600 max steps on Yaxis
    if( XAxisSteps < 0 && XAxisSteps > -10000) {
        digitalWrite(dirPinX, LOW);
        XAxisSteps *= -1;
    }
    else if(XAxisSteps > 0 && XAxisSteps <  10000) {
        digitalWrite(dirPinX, HIGH);
    }
    if( YAxisSteps < 0 && YAxisSteps > -5600) {
        digitalWrite(dirPinLeft, LOW);
        digitalWrite(dirPinRight, LOW);
        YAxisSteps *= -1;
    }
    else if(YAxisSteps > 0 && YAxisSteps <  5600) {
        digitalWrite(dirPinLeft, HIGH);
        digitalWrite(dirPinRight, HIGH);
    }


}

if (YAxisSteps > 0 && YAxisSteps <  5600 && XAxisSteps > 0 && XAxisSteps <  10000) {

    digitalWrite(enablePinLeft,LOW);
    digitalWrite(enablePinRight,LOW);
    digitalWrite(enablePinX,LOW);

    digitalWrite(stepPinLeft, HIGH);
    delayMicroseconds(10);
    digitalWrite(stepPinLeft, LOW);
    delayMicroseconds(10);

    digitalWrite(stepPinRight, HIGH);
    delayMicroseconds(10);
    digitalWrite(stepPinRight, LOW);
    delayMicroseconds(10);
```

```
    digitalWrite(stepPinX, HIGH);
    delayMicroseconds(20);

    digitalWrite(stepPinX, LOW);
    delayMicroseconds(20);

    YAxisSteps--;
    XAxisSteps--;
}
if ( XAxisSteps <= 0 && YAxisSteps > 0 && YAxisSteps <  5600) {
    digitalWrite(enablePinX,HIGH);
    digitalWrite(enablePinLeft,LOW);
    digitalWrite(enablePinRight,LOW);

    digitalWrite(stepPinLeft, HIGH);
    delayMicroseconds(10);
    digitalWrite(stepPinLeft, LOW);
    delayMicroseconds(10);

    digitalWrite(stepPinRight, HIGH);
    delayMicroseconds(10);
    digitalWrite(stepPinRight, LOW);
    delayMicroseconds(10);

    digitalWrite(stepPinX, HIGH);
    delayMicroseconds(20);
    digitalWrite(stepPinX, LOW);
    delayMicroseconds(20);

    YAxisSteps--;
}
if (YAxisSteps <= 0 && XAxisSteps > 0 && XAxisSteps <  10000) {
    digitalWrite(enablePinLeft,HIGH);
    digitalWrite(enablePinRight,HIGH);
    digitalWrite(enablePinX,LOW);

    digitalWrite(stepPinLeft, HIGH);
```

```
      delayMicroseconds(10);
      digitalWrite(stepPinLeft, LOW);
      delayMicroseconds(10);

      digitalWrite(stepPinRight, HIGH);
      delayMicroseconds(10);
      digitalWrite(stepPinRight, LOW);
      delayMicroseconds(10);

      digitalWrite(stepPinX, HIGH);
      delayMicroseconds(20);
      digitalWrite(stepPinX, LOW);
      delayMicroseconds(20);

      XAxisSteps--;
  }
}



}
```

E Arduino code for detecting the goals and playing sounds

```
#include <Wire.h>
#include <TM1637Display.h>
#include "mp3tf16p.h"

// Define the connection pins for the TM1637 (CLK and DIO)
#define CLK 6 // Clock pin (CLK)
#define DIO 7 // Data I/O pin (DIO)

// Create a display object with the defined pins
TM1637Display display(CLK, DIO);

// Define LDR pins
const int ldrPin1 = 2; // LDR sensor for the first person
const int ldrPin2 = 3; // LDR sensor for the second person
```

```
// Variables to track scores
int score1 = 0; // Score for the first person
int score2 = 0; // Score for the second person

// Goal detection variables
bool goalDetected1 = false; // Goal detection for the first person
bool goalDetected2 = false; // Goal detection for the second person

MP3Player mp3(10, 11);

void setup()
{
  // Initialize the display
  display.setBrightness(5); // Set brightness (0-7)
  display.clear();          // Clear the display

  // Set LDR pins as input
  pinMode(ldrPin1, INPUT);
  pinMode(ldrPin2, INPUT);

  Serial.begin(9600);
  mp3.initialize();
  mp3.playTrackNumber(5, 28);

  // Initialize random seed with an analog pin that is not connected to anything
  randomSeed(analogRead(0)); // Ensures different random numbers each time the board is reset
}

void loop()
{
  // Read the value from the LDR sensors
  int ldrValue1 = digitalRead(ldrPin1);
  int ldrValue2 = digitalRead(ldrPin2);

  // Check for goals from the first LDR sensor
  if (ldrValue1 == HIGH && !goalDetected1)
  { // Goal detected for the first person
```

```
    score1++;
    goalDetected1 = true;
    updateDisplay(); // Update the 7-segment display

    int randomTrack;
    do
    {
      randomTrack = random(4, 12); // Generate random number between 4 and 11 (inclusive)
    } while (randomTrack == 5); // Regenerate if the number is 5

    mp3.playTrackNumber(randomTrack, 28);
    Serial.println(randomTrack);
    delay(500);
  }
  else if (ldrValue1 == LOW)
  {
    goalDetected1 = false;
  }

  if (ldrValue2 == HIGH && !goalDetected2)
  {
    score2++;
    goalDetected2 = true;
    updateDisplay();

    int randomTrack;
    do
    {
      randomTrack = random(4, 12);
    } while (randomTrack == 5);

    mp3.playTrackNumber(randomTrack, 28);
    Serial.println(randomTrack);
    delay(500);
  }
  else if (ldrValue2 == LOW)
  {
```

```
    goalDetected2 = false;
  }

  delay(50);
}

void updateDisplay() {
  int displayValue = (score1 * 100) + score2;

  uint8_t separator = 0b01000000;

  display.showNumberDecEx(displayValue, separator, true);
}
```