An-Najah National University
Faculty Of Engineering And Information Technology
Computer Engineering Department

King's Guard

Prepared by:
Mohammad Aker
Yousef Abdulsalam

Supervised by:
Dr. Anas Toma

Presented in partial fulfillment of the requirements for Bachelor degree in Computer Engineering

Sep 8.2024

Acknowledgements

First and foremost, we would like to express our great appreciation to our supervisor; Dr. Anas Toma for his time, efforts, and great advice throughout working on this project. We would also like to thank all the academic staff in the Computer Engineering Department for all the cumulative knowledge we have gained from them over the years, and for their opinions, instructions, and enthusiasm without which we would not have completed this project.

Disclaimer

The following report has been authored by students Mohammad Aker and Yousef AbdulSalam from the Computer Engineering Department, Faculty of Engineering, An-Najah National University. The report has been through minimal modifications, limited to editorial corrections, and may still contain errors in language and content. It is important to mention that the opinions expressed within the report, including any conclusions and recommendations, solely belong to the students. An-Najah National University bears no responsibility or liability for any consequences arising from the utilization of this report for purposes other than its intended commission.

Table of Contents

List of Figures

Abstract

Chess is a two-player game with special rules to be followed. The project is primarily a 1 vs. 1 chessboard; player 1 is the human, and player 2 is a 4-Degrees Of Freedom (DOF) robotic arm capable of moving pieces across the table, capturing the opponent's (human's) pieces, and retrieving its pieces upon reaching the other end of the chessboard, requiring no human assistance.

We acknowledge that this project has been attempted before; however, ours is executed differently in several ways as follows: A custom Object Detection Model (ODM) is built for this project using nearly 10K images as a dataset. The model will be aided by the Luxonis OAK-D camera, in addition to an array of sensors (to be determined) specifically used for retrieving pieces that belong to the robotic arm if the pawn reaches the other end of the board. The arm will be composed of multiple stepper and servo motors, allowing it to move in three dimensions. Additionally, it will have obstacle detection sensors (such as ultrasonic, infrared transmitters, and receivers) to prevent interference with the camera. Pressure sensors will allow the system to shut down in the event of the player leaving the game. Finally, a dedicated control panel will give the player commands and control over the system. Some of these commands could be executed using voice control, aided by specific voice detection modules.

The main microcontroller will be a System On Chip (SOC) Raspberry Pi; it will be required to connect the camera and house the ODM; and a secondary microcontroller will be necessary to control the motors operating the robotic arm, modules, and the sensors that complement the project.

1. Introduction

In recent years, the integration of robotics and artificial intelligence has led to significant advancements in automation, human-machine interaction, and problem-solving. One particularly engaging application of this technology is in the field of automated gameplay, where machines can participate in complex, strategy-based games like chess.

This project focuses on the development of a robotic arm capable of autonomously playing chess against human opponents. The system is designed to physically move chess pieces on a standard chessboard, utilizing real-time image processing for board recognition and decision-making algorithms powered by the Stockfish chess engine.

1.1 Problem Statement

The main problem at hand is a requirement for real-world chess to have 2 players, marking unpaired individuals unable to play the game with experienced players who are familiar with the rules and do not hold back against the player.

1.2 Objective

King's Guard aims to be an easy-to-use, plug-and-play system that utilizes machine learning, robotics, and different components to deliver a high-end chess experience with nearly no features missing and tolerating mistakes that players may make during a standard chess game.

1.3 Scope of Work

1. **Develop a 3D-Printed Robotic Arm**: Design and construct a robotic arm capable of precise and accurate movement to autonomously manipulate chess pieces on a standard chessboard.
2. **Integrate machine learning for Real-Time Chess Decision-Making**: Implement a custom-built Object Detection Model (ODM) that analyzes the chessboard using real-time images from the Luxonis OAK-D camera and determines the optimal moves.
3. **Enhance User Interaction with Advanced Features**: Incorporate additional functionalities such as leaving game detection, audio command capabilities, and obstacle detection in the camera's Field of View (FoV), all to tolerate the user's behavior that may occur.
4. **Create a Web Interface for Remote Monitoring**: Develop a web interface that displays the chessboard using Forsyth-Edwards Notation (FEN), allowing users to track the game remotely in real time and fix any mistakes the ODM may make.

1.4 Significance

A way the project could be considered is to prepare chess players for chess tournaments, eliminating the need for a companion, and getting used to the real board, thus getting rid of smart devices that may pose a psychological factor that needs breaking. In addition, the inclusion of user-friendly features like voice commands and a web interface broadens accessibility, making the system more intuitive.

2. Constraints and Earlier Coursework

2.1 Constraints

1. **Mechanical limitations**: due to the limited torque of the servo motors, compromises had to be made regarding the joints, the gripper, and weights in general, making the system rather not customizable due to the fixed length of the arm.
2. **Lighting and surrounding**: due to using the camera, the system is not immune to light changes, it could interpret pieces falsely due to lighting changes, the pieces could be corrected via the built web interface, but it could be frustrating overtime.
3. **Processing power**: The system is power using the Raspberry pi, a powerful System on Chip (SOC), however, it appears that it handles the built model slowly, while on a standalone average personal computer (PC) takes nearly 4 times less than on the raspberry pi, it still gets the job done.

2.2 Earlier Coursework

1. **Electronics courses and electronic labs**: they were very useful for understanding the used electronic components and how they worked, specially the load cell sensor.
2. **Image processing course**: it was very important because it showed the process of how to deal with cameras and visualize the final image, thus, making the process much easier to deal with.
3. **Microcontroller course using PIC**: this course also was extremely important because it made us understand how microcontrollers work and how they should be dealt with and interface them with other components.
4. **Critical thinking course**: the ability to approach the project methodically and make wise conclusions was provided by the Critical Thinking course, which was crucial to the project's success. The course helped us develop critical thinking abilities that enabled us to recognize potential problems, examine them, and come up with workable solutions. This became even more crucial as the project progressed and when design and power-related challenges were encountered.

3. Literature Review

3.1 Mechanical chess

The concept of a mechanical robot playing chess dates back to 1770 when the Turk was all hype. The Turk is a life-sized mechanical robot that played chess against human opponents, though the concept later turned out to be fraudulent, due to various chess masters operating the Turk from under the table. The idea of eliminating the need for a second player was in mind way back. After the exposure of the Turk, several other attempts were made to create machines capable of playing chess independently, but none succeeded in reaching the level of autonomy and artificial intelligence that is seen today. These early attempts laid the foundation for later advancements, illustrating the long-standing interest in combining chess with mechanical or artificial players.

3.2 Virtual chess

In modern times, various applications are centered around the concept of chess, such as "chess.com," which incorporates bots with many difficulties to play against, featuring multiple chess engines with many strategies, including but not limited to Stockfish, which was used in this project. Chess.com represents the ultimate stage that virtual chess has reached in the modern era. Alongside chess.com, other platforms like Lichess and Fritz have also contributed to the growth of online and virtual chess, offering players the chance to compete against advanced engines or human opponents from across the globe. The evolution of virtual chess has made it accessible to players of all levels, providing tools for analysis, strategy improvement, and even historical game reviews. The use of engines like Stockfish, which constantly updates its algorithms, shows how far virtual chess has come from the days of simple AI opponents to highly advanced systems capable of calculating millions of moves ahead.

3.3 Similar projects

We are aware that this is not the first time this project has been attempted before, but this approach combines multiple unique aspects, is built differently, and most importantly, guarantees that the system works as much fewer flaws as possible.

4. Methodology

4.1 System Structure

4.1.1 Wooden Box

The main housing of the project is a wooden box, standing at 80*83*25, all sides filled, with multiple openings for easier wiring and more convenient component placing.

Figure (1): Wooden Box

4.1.2 Camera Stand

With the camera being a component of the project, a static stand is needed, a 84*86 stand was screwed to the wooden box, thus forming the main structure.

Figure (2): Camera Stand

4.1.3 Robotic Arm

Comprising of multiple 3D-printed joints, the arm extends fully to nearly 60cm, capable of moving in 4 Degrees-of-Freedom (DOF), placed on an elevated wooden piece, and at its end, a 3D-printed, 3 finger claw gripper capable of lifting all the chess pieces. (Thingiverse, n.d.)

Figure (3): Assembly of the Gripper

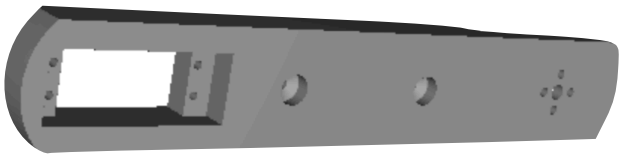

Figure (4): The Joint

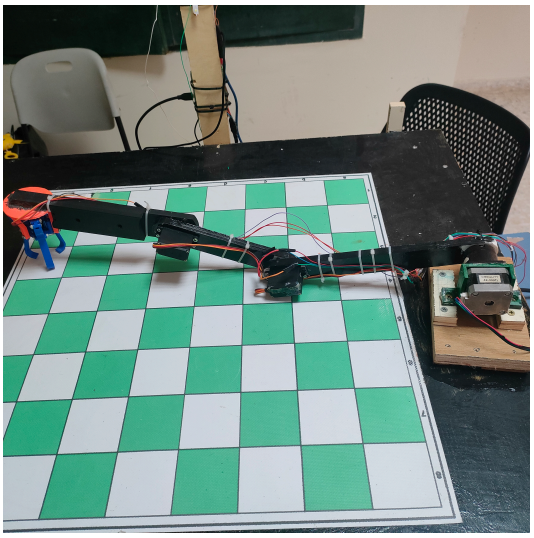

Figure (5): The Final Arm

4.2 Used Processing Units and Components

4.2.1 Raspberry pi 5

The Raspberry Pi 5, particularly the 4GB model, is a powerful upgrade in the Raspberry Pi lineup, designed for more demanding applications while maintaining the platform's signature compact form factor and affordability. It features a quad-core ARM Cortex-A76 processor running at 2.4 GHz, providing significantly better performance than its predecessors, making it suitable for multitasking, software development, and handling more intensive workloads like machine learning or video processing. With 4GB of LPDDR4X RAM, the 5th generation Raspberry Pi ensures smooth performance for a variety of applications, including running multiple Docker containers, emulating retro games, or building complex IoT systems.

The Raspberry Pi 5 offers dual 4K HDMI outputs, enabling high-resolution display setups for media centers or digital signage. It also includes PCIe support via a custom slot for faster storage options or network cards, expanding the range of potential uses. Connectivity options include faster USB 3.0 ports, Gigabit Ethernet, and built-in Bluetooth 5.0 and Wi-Fi 6, ensuring high-speed data transfer and stable wireless connections. Additionally, the improved GPU allows for hardware-accelerated video decoding, enabling smooth playback of 4K videos. The Raspberry Pi 5, with its enhanced technical features, is ideal for enthusiasts, developers, and engineers looking for a compact yet robust platform for advanced computing projects.

The raspberry pi 5 was used as the “Master” controller in King’s Guard, it managed the microphone, OAK-D camera, communication between other microcontrollers, the logic the chess game needs, and finally it ran the ODM. (Raspberry Pi Foundation, n.d.)

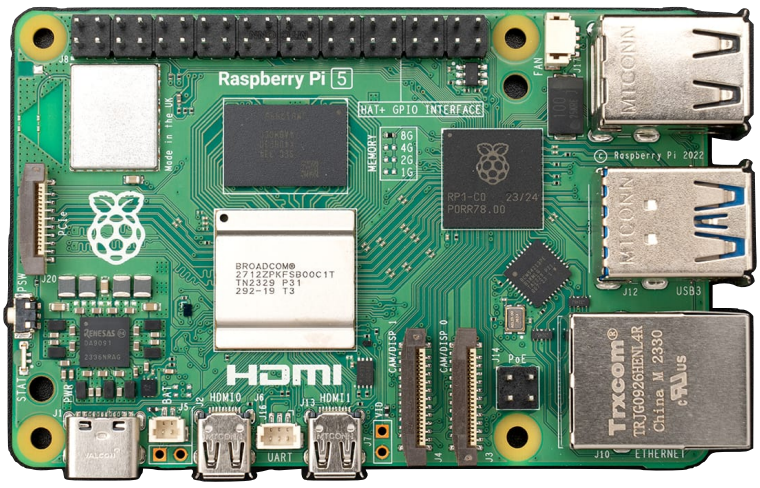

Figure (6): Raspberry pi 5

4.2.2 Luxonis OAK-D camera

The OAK-D is an advanced camera designed for robotic vision, offering human-like perception by combining a stereo depth camera and a high-resolution color camera with on-board neural network inferencing and computer vision capabilities. Powered via USB-C for both power and USB 3.0 connectivity, the OAK-D provides a smooth user experience, particularly when following the USB deployment guide. Built on the RVC2 architecture, it delivers 4 TOPS of processing power, with 1.4 TOPS dedicated to AI performance, enabling the use of any AI model, including custom-built ones (after conversion). It supports H.264, H.265, and MJPEG encoding, with up to 4K at 30 FPS or 1080p at 60 FPS. The camera's computer vision capabilities include warp (undistortion), resizing, cropping, edge detection, and feature tracking, with the flexibility to run custom CV functions. Stereo depth perception includes post-processing, RGB-depth alignment, and high configurability, while object tracking is supported in both 2D and 3D via the ObjectTracker node. With a baseline of 75mm and an ideal depth range of 70cm to 12m, the OAK-D is housed in industrial-grade aluminum with a front Gorilla Glass for durability. It features a 1/4" tripod mount, a 75mm M4 VESA mount, and has a power consumption of up to 5W. The compact design measures 110x54.5x33 mm and weighs 115g, making it a powerful yet portable solution for advanced robotic vision applications.

The OAK-D camera was responsible for capturing the frames that needed to be divided and analyzed by the ODM

Figure (7): Luxonis OAK-D camera

4.2.3 Touch Screen

The used touch screen is an off-brand 7-inch touch screen with a resolution of 1024*600, the touchscreen requires 2 connections, an HDMI to transfer the image, and a power/touch to transfer touch inputs to the raspberry. The screen is used as a display for the web interface and the interactive chessboard.

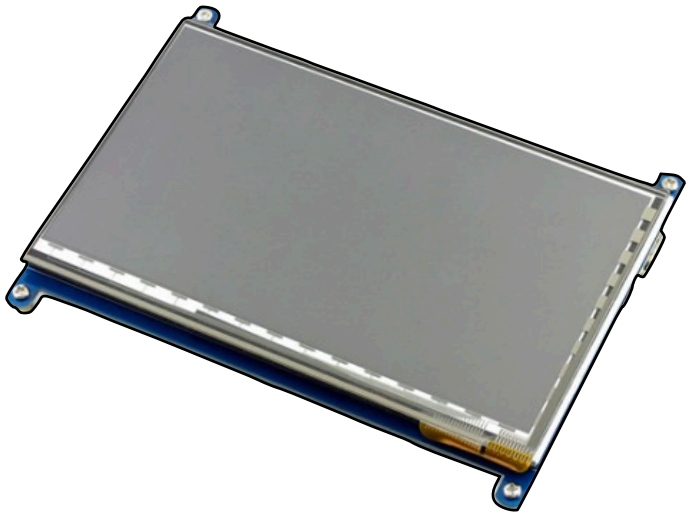

Figure (8): Touch Screen

4.2.4 4 Channel I2C Logic Level Shifter Converter Bi-Directional Module 5V To 3.3V

The 4 Channel I2C Logic Level Shifter Converter Bi-Directional Module allows seamless communication between devices operating at different voltage levels, such as 5V and 3.3V. It is in this project used to convert voltage levels from the Arduino Mega level (5V logic) to the Raspberry Pi level (3.3V logic).

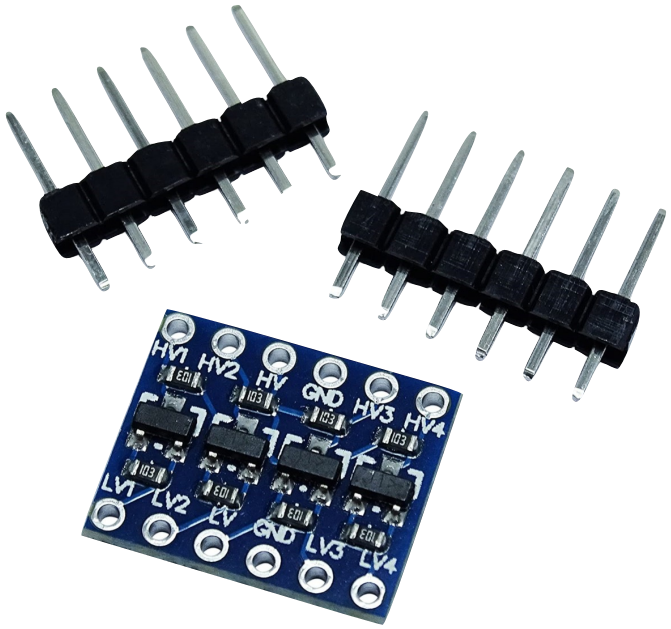

Figure (9): 4 Channel I2C Logic Level Shifter Converter

4.2.5 Arduino Mega

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller. (Arduino Mega 2560 Rev3, n.d.)

The Mega was utilized as the controller for all motors, getting the generated move and moving the pieces around the board according to special square mapping.

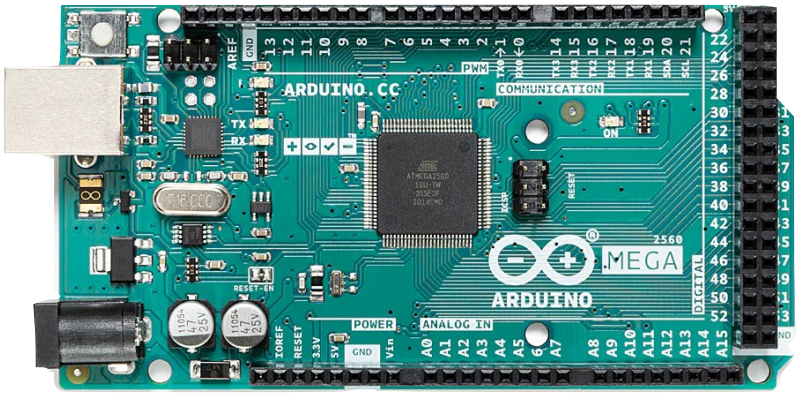

Figure (10): Arduino Mega

4.2.6 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller, allowing it to be easily connected to a computer via USB for programming or powered via an external battery or AC-to-DC adapter. (Arduino, n.d.)

The uno was used to power and get the readings of all used sensors in the system, and supply it to the raspberry pi.

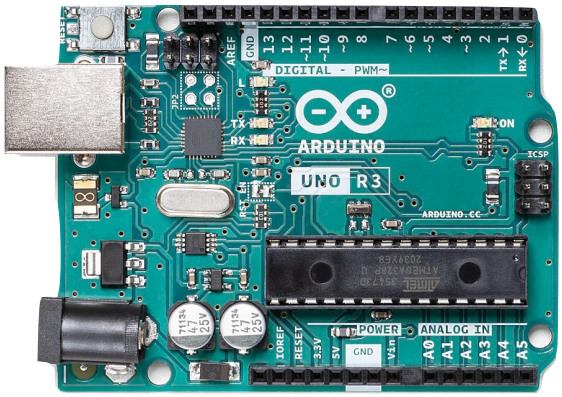

Figure (11): Arduino Uno

4.2.7 Power Supply

An ATX12v switching power supply was retrieved from an old computer and used in this project to supply the necessary voltage and current to the motors and components.

Figure (12): Power Supply

4.2.8 Microphone

A standard issue USB microphone capable of inputting audio to the connected device, was used to input the voice command (pause and resume) in the system.

Figure (13): Microphone

4.2.9 Load Cell and HX711 ADC

A single-point load cell is a type of force sensor designed to measure weight or pressure applied at a single point, often used in industrial and consumer applications such as scales or load measurement systems. It works by converting mechanical force into an electrical signal using strain gauges bonded to a metal body that deforms under load. The electrical signal generated by the load cell is very small, typically in the millivolt range, which requires amplification before it can be accurately measured. This is where an Analog-to-Digital Converter (ADC) like the HX711 comes in. The HX711 is a popular 24-bit ADC specifically designed for weight scales and load cell applications. It amplifies the small signal from the load cell and converts it into a digital value that can be processed by a microcontroller or computer. The HX711 is easy to integrate, requiring minimal external components, and provides accurate and stable measurements, making it a widely used solution for load cell systems.

The load cell was used in the pawn promotion mechanic, it weighs the piece upon it, if the weight matches that of a queen, then it proceeds.

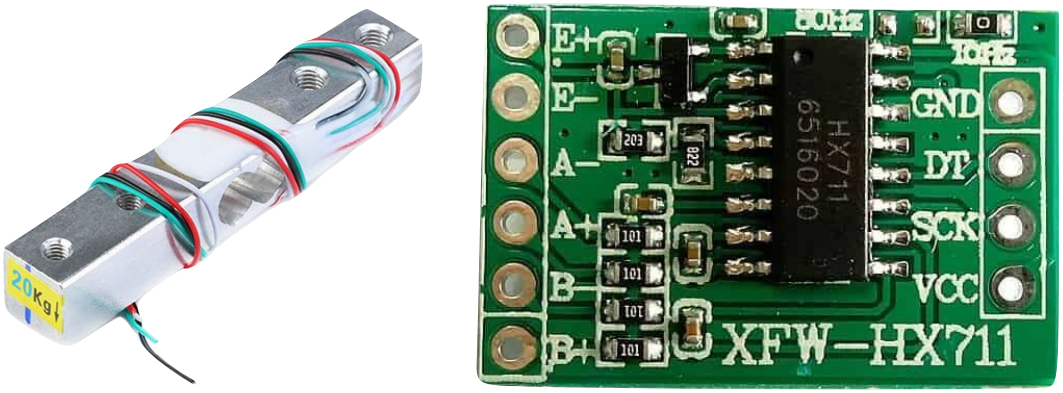

Figure (14): Load Cell and HX711 ADC

4.2.10 PIR Motion Sensor

PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensor range. They are small, inexpensive, low-power, easy to use, and don't wear out. For that reason, they are commonly found in appliances and gadgets used in homes or businesses. They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.

PIRs are made of pyroelectric sensors, which can detect levels of infrared radiation. Everything emits some low-level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is split in two halves. The reason for that is that we are looking to detect motion (change) not average IR levels. The two halves are wired up so that they cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low.

new PIRs have more adjustable settings and have a header installed in the 3-pin ground/ out/power pads.

PIR was used to detect whether the player had left the game or not, if the player left and didn’t come back in 2 minutes and the game was not paused, then the system will shut down.

Figure (15): PIR Motion Sensor

4.2.11 Ultrasonic Sensors

The HC-SR04 is an ultrasonic sensor used to measure distances accurately by emitting sound waves and detecting their reflection from objects. It is widely used in robotics, obstacle avoidance systems, and distance measurement projects. The sensor consists of a transmitter and receiver and works by sending an ultrasonic pulse and calculating the time it takes for the sound to bounce back, providing a distance reading. It has a range of 2 cm to 400 cm, with a typical accuracy of around 3 mm. In this project, it is used to detect obstacles in the camera's FOV.

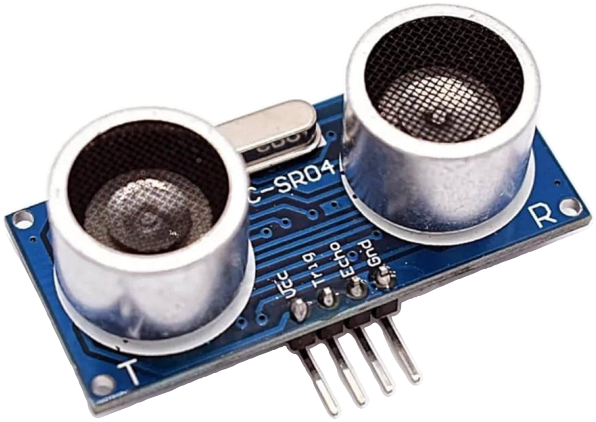

Figure (16): Ultrasonic Sensor

4.2.12 Nema 17 Stepper Motor (Creality 42-34)

The Nema 17 Stepper Motor (Creality 42-34) is a compact and reliable motor widely used in 3D printers, such as those from Creality, and other precision-driven applications like robotics and small CNC machines. With a holding torque of around 0.45 Nm and a 1.8° step angle, it provides smooth and accurate control over movement. it is used to lift the robotic arm up and down, screwed on a wooden base.

Figure (17): Nema 17 Stepper Motor

4.2.13 Nema 23 Stepper Motor (Zhengji - J-5718HB2401)

The Nema 23 Stepper Motor (Zhengji - J-5718HB2401) is a high-performance motor commonly used in precision applications such as CNC machines, 3D printers, and robotics. It offers a holding torque of 1.89 Nm, making it suitable for tasks requiring reliable and strong rotational force. The motor operates with a 1.8° step angle, providing accurate control over positioning, which is essential for applications like robotic arms or precise movements in automation systems. It is used to move the whole robotic arm including the base of it.

Figure (18): Nema 23 Stepper Motor

4.2.14 YS-DIV268N-5A and Microstep 3.5A Drivers

Designed to provide precise control over two-phase stepper motors. It allows for micro-stepping, which divides a motor's full step into smaller steps, improving the smoothness and accuracy of the motor's movements. They are used to power both stepper motors mentioned above.

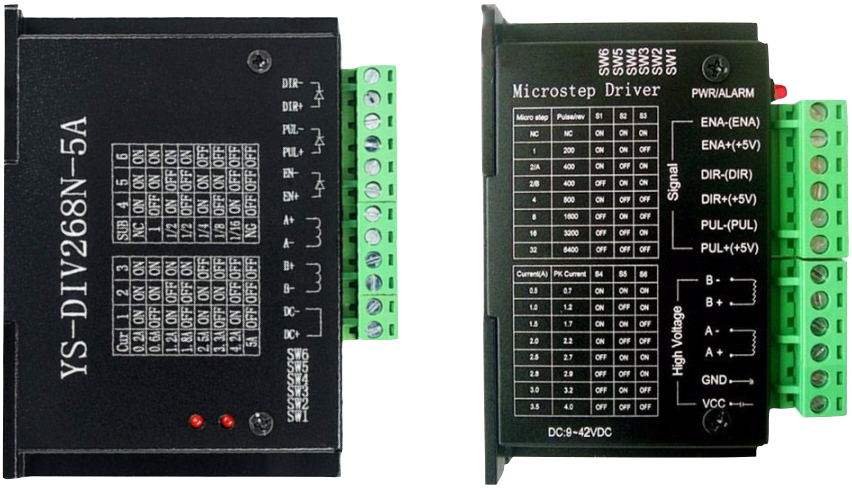

Figure (19): Stepper Drivers

4.2.15 Servo Motor MG946R

The MG946R is a high-torque digital servo motor commonly used in robotics, RC cars, and other hobbyist projects. It provides strong, reliable motion control with a torque of up to 12 kg/cm, making it suitable for applications that require lifting or precise movement of heavier components. The servo features metal gears, which improve its durability and lifespan compared to plastic-geared servos. It operates at 4.8-7.2V, and its digital circuitry ensures smooth and accurate positioning, making it a popular choice for projects requiring both power and precision. 2 are used for the system, each screwed on a different joint to achieve a degree of freedom, and thus, achieve more areas to reach.

Figure (20): Servo Motor MG946R

4.2.16 Servo Motor 9g

The 9g servo is a lightweight, compact micro servo motor commonly used in small electronics projects, such as Arduino and RC planes. It is known for its low weight (around 9 grams) and small size, making it ideal for applications where space and weight are limited. Despite its size, the 9g servo provides reliable control with a torque of around 1.8 kg/cm, enough for light-duty tasks like moving small arms or controlling lightweight mechanisms. In this project, it is to operate the gripper, close and open it at 2 angles to hold the pieces firmly.

Figure (21): Servo Motor 9g

4.3 Utilized Software

4.3.1 ODM

To achieve the element of uniqueness and to follow current trends, a direction was taken towards machine learning and built an ODM that consists of nearly 7500 images as a dataset, divided into train, test, and validation.

The process in mind was to create as many positions for as many pieces as possible considering the time limit. At first, an attempt was made to include all black pieces as separate classes, but the results were not promising at all due to the nature of black colors and the not being able to identify the details in black pieces, so the approach followed was to sum all black pieces in a single class, identify blank squares, and identify every single white piece, thus the total was 8 classes, the logic used was that the robotic arm will play with black pieces, so it is not cared about where it moves since it can be tracked it in-code, the problem lied in the white pieces and classifying every single one that occupied a square to identify whether the player has switched up the pieces, played an illegal move, or played more than one move via a turn, all of this was achieved using the ODM. The model was trained using YOLO technology provided by Ultralytics, and using Adam optimizer. (Ultralytics, 2024)

Shown below are some metrics and graphs that the model was able to achieve.

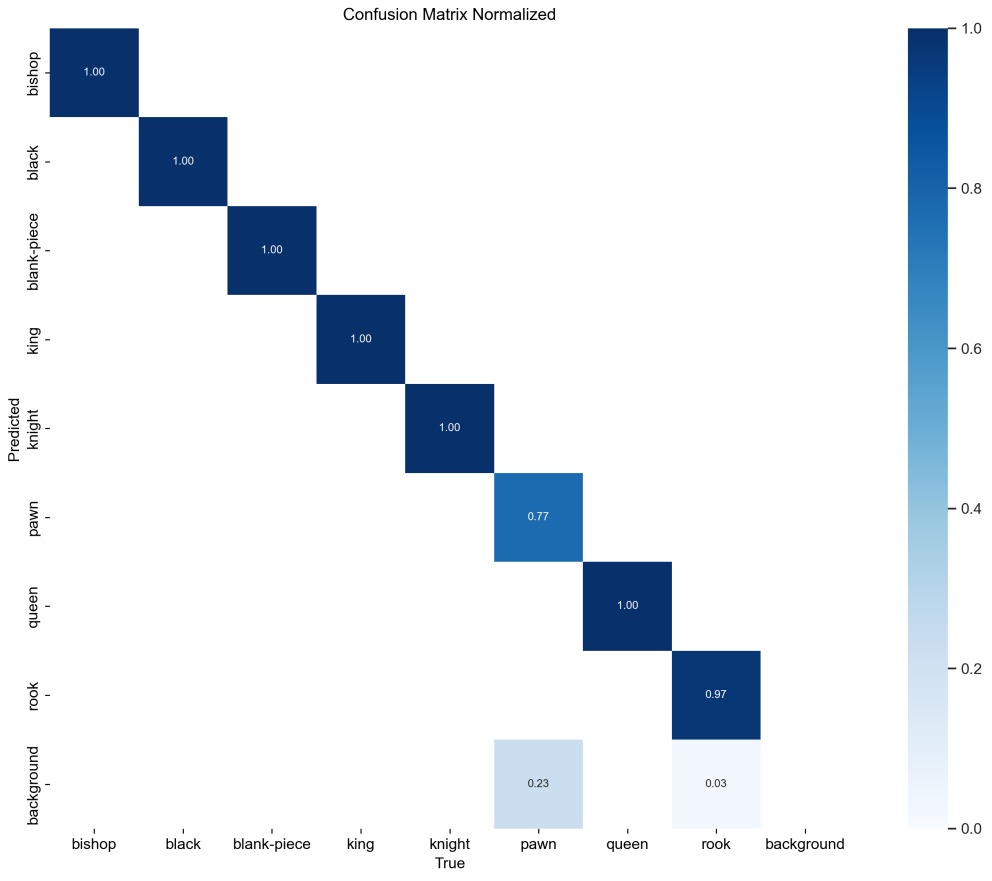

Figure (22): Confusion Matrix Normalized

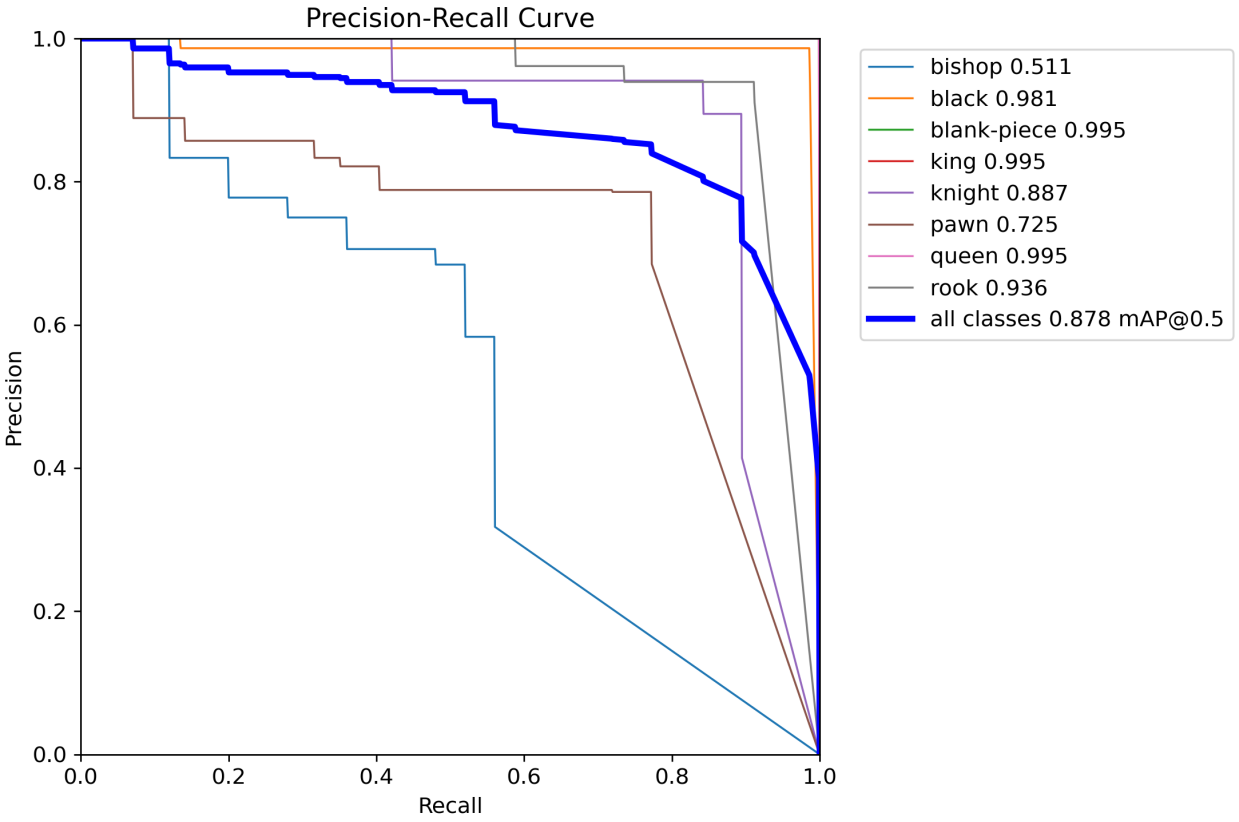

Figure (23): Precision-Recall Curve

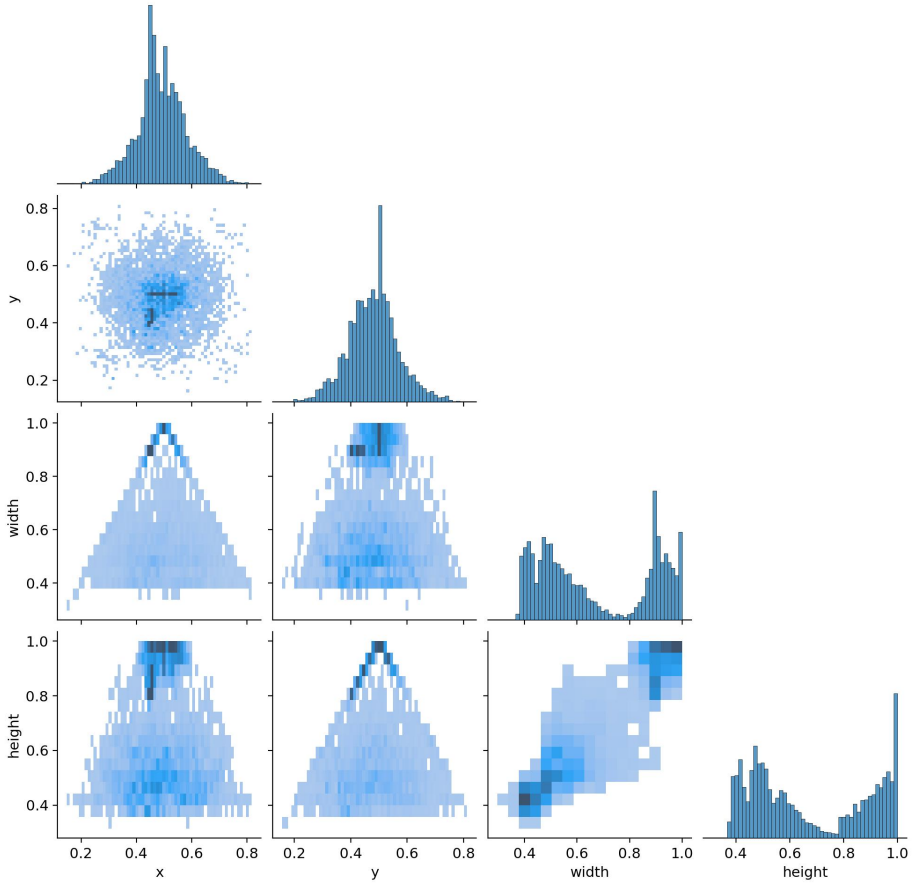

Figure (24): Labels Correlogram

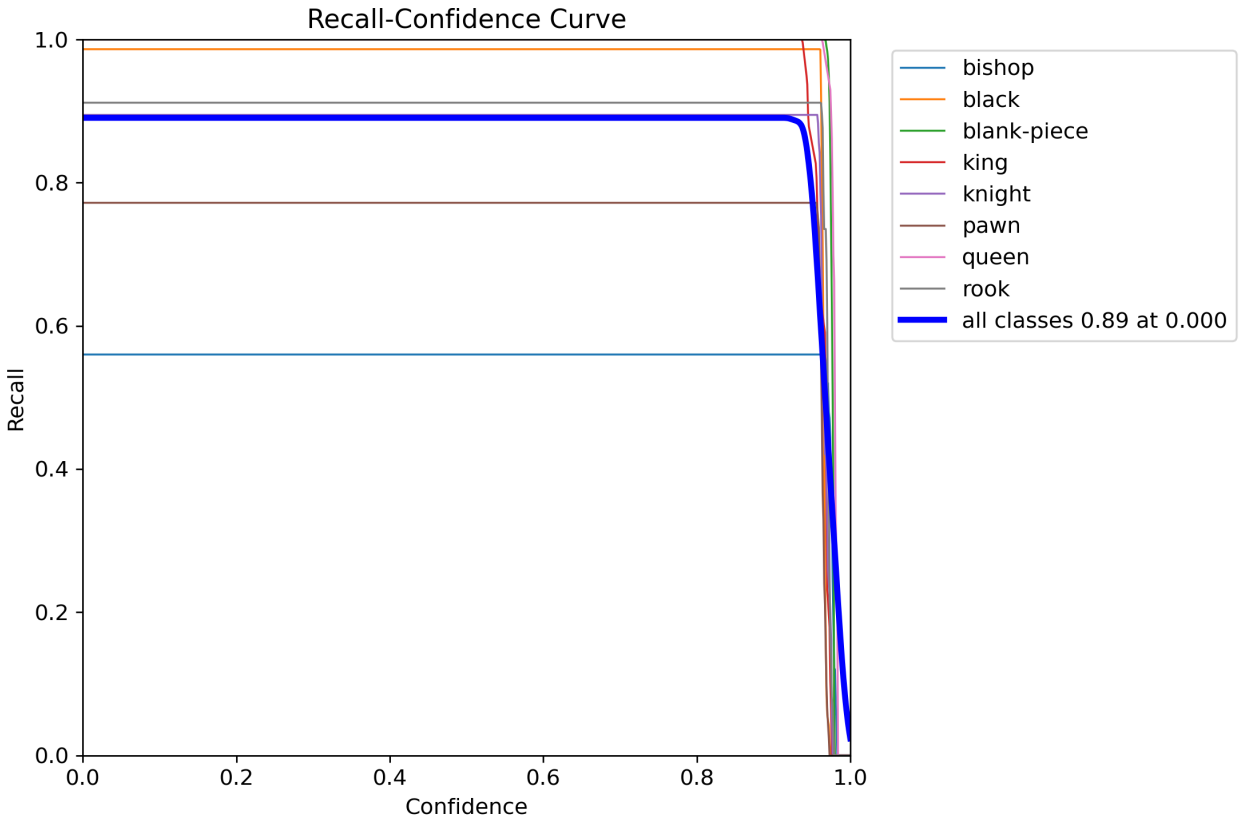

Figure (25): Recall-Confidence Curve

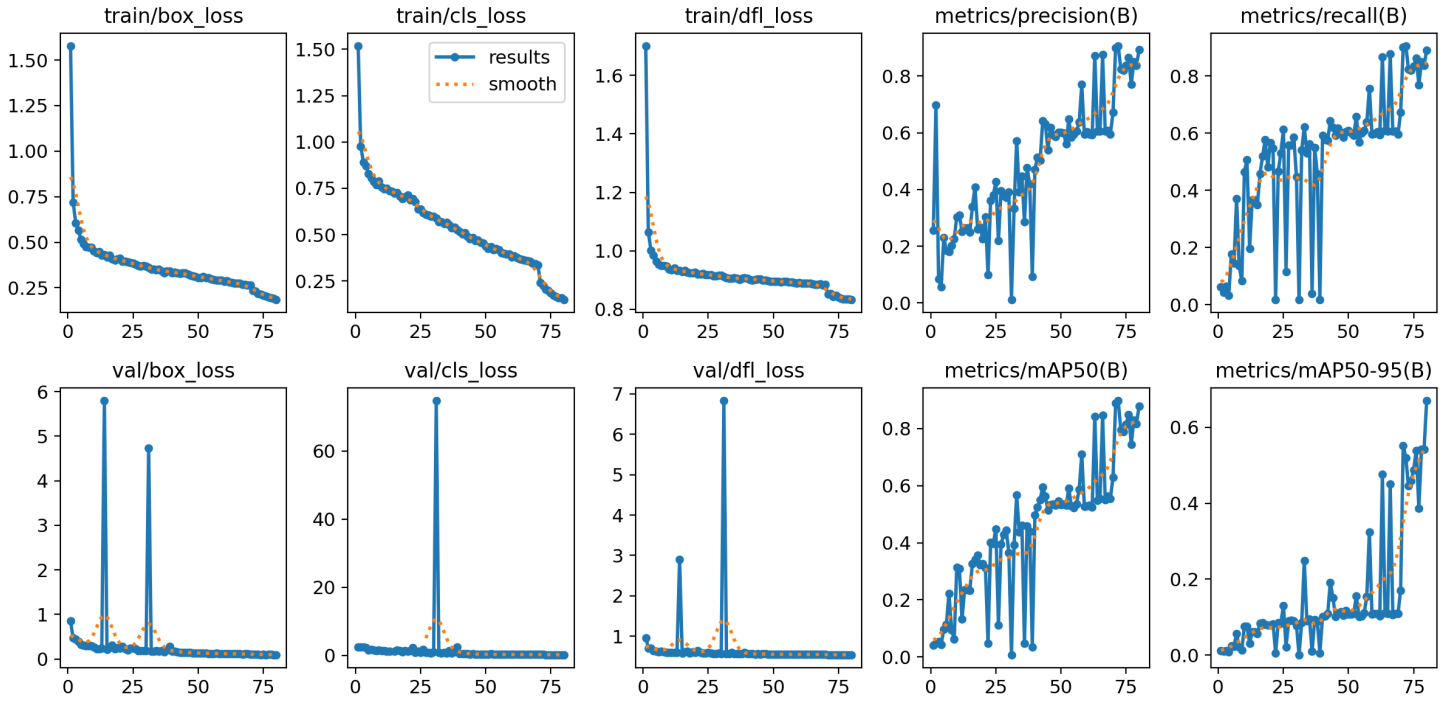

Figure (26): Results

4.3.2 Stockfish

stockfish.py is a Python library that provides an interface to interact with Stockfish, a powerful open-source chess engine known for its advanced analysis and gameplay strength. By using stockfish.py, developers can easily integrate Stockfish's capabilities into their Python projects, enabling the program to evaluate board positions, suggest moves, or even play chess games at various difficulty levels. This library allows for sending commands to the Stockfish engine and receiving responses in a structured manner, making it useful for building applications such as chess bots, analysis tools, or interactive learning platforms. The library simplifies the process of communicating with Stockfish, making it accessible even to those with limited knowledge of chess engine protocols. This powerful chess engine was used to assist in setting up the board, visualizing it in-code, manipulating pieces, and getting the best moves for the robotic arm to apply.

4.4 How The System Works

4.4.1 Connections

The system utilizes multiple connection techniques, keeping in mind unifying the ground for all components and supplying as much current and voltage as the components need, the figure below shows all connections of the system:

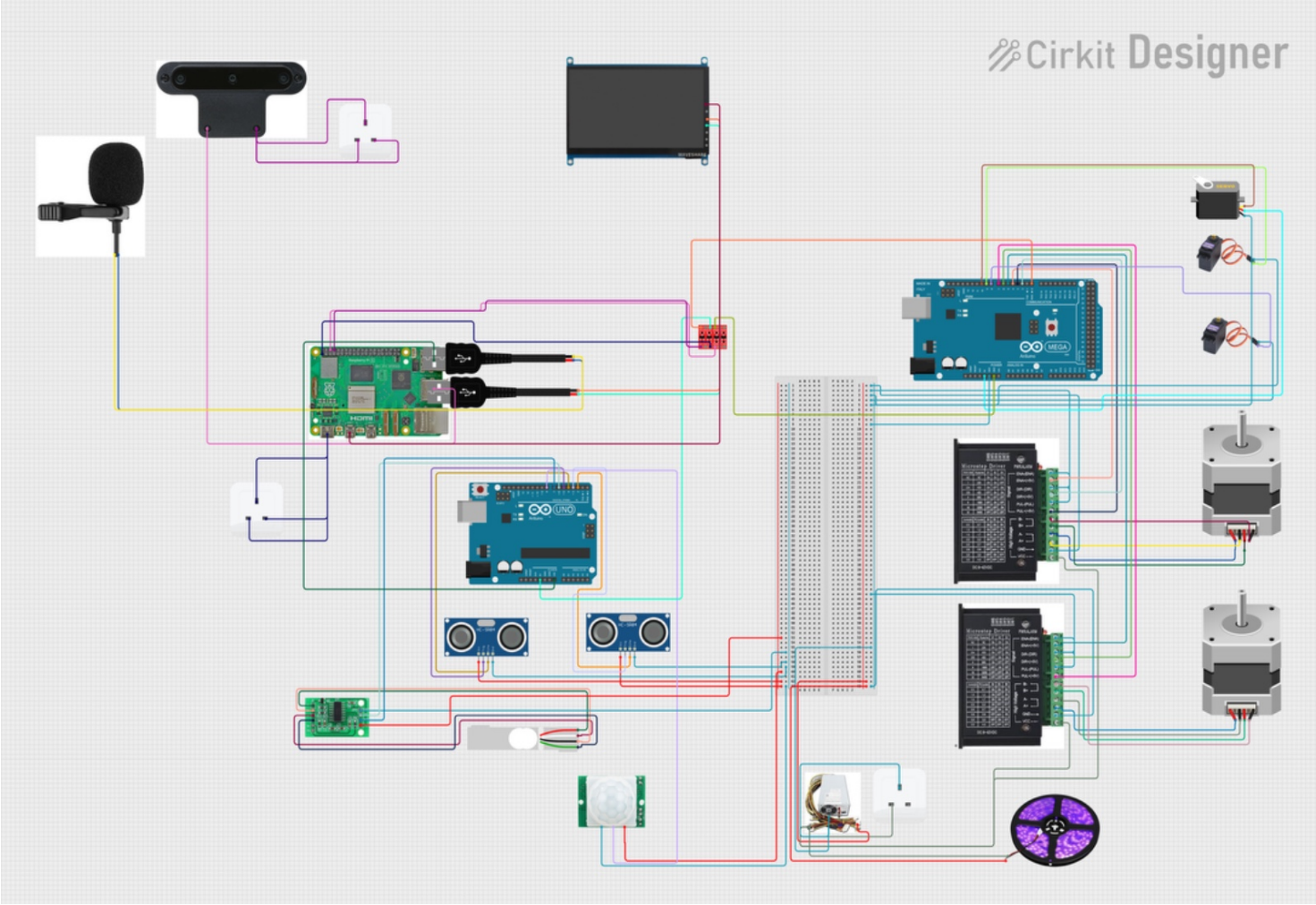

Figure (27): Wired Connections

As shown, we can derive that there 3 sections of the connections that require manually plugging it in:

1. **The Power Supply**

   All motors (except the 9g servo), drivers, and LED strips are connected individually to the power supply due to the required 12 volts and higher amps to operate compared to what the other components supply.

2. **The Raspberry Pi**

   This System On Chip (SOC) houses multiple peripherals connected to the USB ports, starting with the Arduino UNO that supplies the Raspberry with the necessary sensor reading, moving on to the USB microphone that is used for a couple of voice commands, and next the USB port for OAK-D camera, and finally the touch input for the used touchscreen. It is worth noting that the Arduino Mega is connected to the Raspberry Pi via TX/RX pins, which supply the necessary coordinates to the Mega when it is time to move for black.

3. **OAK-D power supply**

   Due to a "low voltage" warning on the Raspberry Pi that appeared when dealing with the OAK-D camera when getting its power from the Raspberry Pi USB port, an approach was made to supply the camera with its power using the proprietary power supply.

4.4.2 Booting Up

Booting up the system is rather simple, first, you need to plug in the power brick that powers the Raspberry Pi and the Arduino Mega, then plug in the power supply for the OAK-D camera, then you need to activate the Flask server and the frontend interface that are providing logic for how the game of chess is managed (refer to appendix B), then, you need to set up the robotic arm at its marked starting position, otherwise, it won't move properly to its initial position, and finally plug in the ATX power.

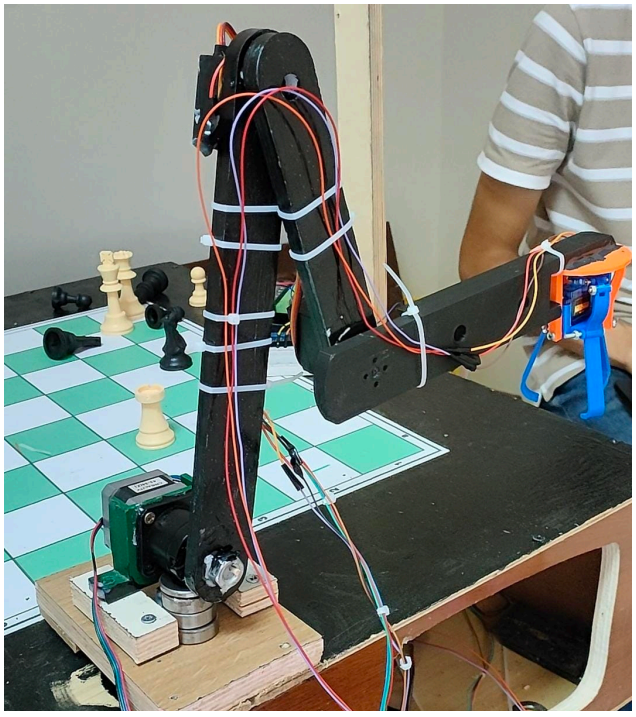

Figure (28): The Initial position for The Robotic Arm

4.4.3 Playing The Chess Game

The game starts with white's (the player's) turn, it makes its move and presses the "Done playing" button on the web interface, this sends a "process" request to the backend to capture a frame of the chessboard to analyze, but before capturing, it checks the ultrasonic sensors whether there's an obstacle in range of the camera's FOV and returns that there's an obstacle detected by either sensors, checks the load cell to see if there's a black queen available if Stockfish decides to promote a pawn and returns "No pieces detected for retrieval" if there's not a black queen or an object other than it, and finally checks if the game is paused (voice command "pause" was lastly spoken) and returns that the game is paused if it is, all these criteria are shown in the flowchart below, if they are all met, the camera takes the frame, splits the board into 64 images, passes each into the ODM and returns a single detection per image, these 64 detections are used to generate the new Forsyth-Edwards Notation (FEN, a string that represent all pieces on the board) by comparing the detections to the old FEN and checking if there any pieces that got switched, or if an illegal move that has been made, if any of these checks out, then it returns the fault and does not proceed with the game. (Chess.com, n.d.)

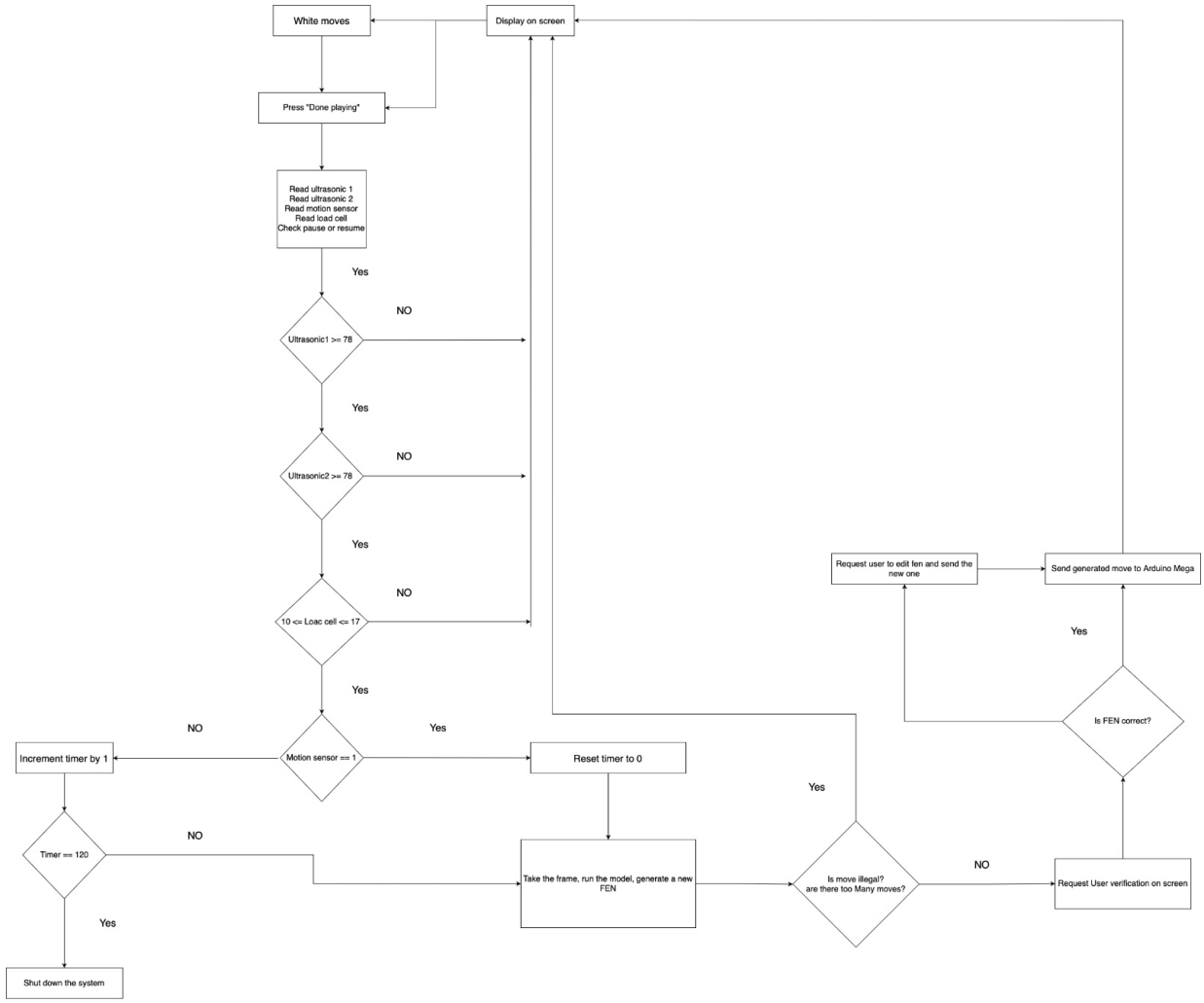

Figure (29): General System Flowchart

After all these checks, the new FEN is returned to the interface with a prompt to the player if it is all interpreted correctly, if yes then black's (the robotic arm's) move is generated using Stockfish, and passed to Arduino Mega to move, capture, or promote, if the returned FEN is incorrect then the system prompts the player to send the correct one by drag-and-dropping chess pieces in the interactive board on the screen and sending the correct FEN to the backend, and then it's back to generating the move for black and passing it to the Mega.

Figure (30): Web Interface

Lastly, after receiving the move in the format of "e7e5m" as in the first square is e7, the second square is s5, and more type is "move", in addition, it could be c as in "capture" or r as in"retrieve", this affects how the arm moves, to elaborate, if the move type is "m" it moves the piece from square 1 to square 2, if the move type is "c" it picks up the piece from square 2, returns it to the initial, and picks up piece on square 1 and places it back on square 2, and finally, if the move type is "r", it picks up the piece on square 1, moves it to the initial and picks up the Queen on the load cell and places it on square 2.

5. Results & Discussion

5.1 Results

The system provides an efficient and fun game of chess that smartly detects multiple conditions. Additionally, the system is equipped with a web interface that serves as a way to tolerate errors that may be produced by the model, thus, achieving an advancement in the game of chess

5.2 Discussion

The system introduces new and innovative features compared to traditional ideas and methods. While it is still in the deployment phase, it offers a solution that has the potential to provide smart ways to pass time for many people.

6. Conclusions & Future Work

6.1 Conclusions

In conclusion, we believe that this idea has the potential to have a significant impact on the entertainment industry and could become a highly sought-after product in the future. Its simplicity and user-friendly nature will make it accessible to a wide range of individuals.

This has been the primary goal and motivation behind our project from the outset.

6.2 Future Work

- Make the system customizable-friendly by adding more boards and pieces.
- Optimize the system for less delay time.
- Train the model more and add more images to the dataset to make it less susceptible to false truths.
- Develop an app to control the system more.
- Add game analytics to analyze the player's moves and give comments.
- Use special materials instead of wood to make the system more robust and portable.

A References

Ultralytics. (2024, September 7). *Predict*. Ultralytics YOLO Docs. https://docs.ultralytics.com/modes/predict/#inference-sources

Arduino. (n.d.). *Arduino Uno Rev3*. Arduino Documentation. Retrieved September 7, 2024, from https://docs.arduino.cc/hardware/uno-rev3/

Arduino. (n.d.). *Arduino Mega 2560 Rev3*. Arduino Store. Retrieved September 7, 2024, from https://store.arduino.cc/products/arduino-mega-2560-rev3

Raspberry Pi Foundation. (n.d.). Documentation. Raspberry Pi. Retrieved September 7, 2024, from https://www.raspberrypi.com/documentation/

Chess.com. (n.d.). FEN chess notation. Chess.com. Retrieved September 7, 2024, from https://www.chess.com/terms/fen-chess

Thingiverse. (n.d.). *Robot gripper with three fingers for chess robot by maquina_pensante* [Files]. Thingiverse. Retrieved September 7, 2024, from https://www.thingiverse.com/thing:1322867/files

B Arduino Uno Code for Sensor Reading

```
#define TRIG_PIN1 2
#define ECHO_PIN1 3
#define TRIG_PIN2 5
#define ECHO_PIN2 4
#define PIR_PIN 8
#define LOADCELL_DT 6
#define LOADCELL_SCK 7

#include <HX711_ADC.h>

// Load Cell initialization
HX711_ADC LoadCell(LOADCELL_DT, LOADCELL_SCK);

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

  // Initialize Ultrasonic Sensors
  pinMode(TRIG_PIN1, OUTPUT);
  pinMode(ECHO_PIN1, INPUT);
  pinMode(TRIG_PIN2, OUTPUT);
  pinMode(ECHO_PIN2, INPUT);
  pinMode(PIR_PIN, INPUT);

  // Initialize Load Cell
  LoadCell.begin();
  float calibrationValue = 101.38; // Set the calibration value
  unsigned long stabilizingtime = 2000; // Stabilizing time after power-
up
  boolean _tare = true; // Perform tare operation after start
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag()) {
    Serial.println("Timeout, check MCU>HX711 wiring and pin
designations");
    while (1);
  }
  else {
    LoadCell.setCalFactor(calibrationValue); // Set calibration value
    Serial.println("Load Cell Startup Complete");
  }
}

void loop() {
  long duration1, distance1, duration2, distance2;
  static boolean newDataReady = 0;
  const int serialPrintInterval = 0; // Increase to slow down serial
print activity
  static unsigned long t = 0;

  // Ultrasonic Sensor 1
  digitalWrite(TRIG_PIN1, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN1, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN1, LOW);
```

```
  duration1 = pulseIn(ECHO_PIN1, HIGH);
  distance1 = (duration1 * 0.034) / 2;

  // Ultrasonic Sensor 2
  digitalWrite(TRIG_PIN2, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN2, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN2, LOW);
  duration2 = pulseIn(ECHO_PIN2, HIGH);
  distance2 = (duration2 * 0.034) / 2;

  // PIR Sensor
  int motion = digitalRead(PIR_PIN);

  // Load Cell Reading
  if (LoadCell.update()) newDataReady = true;
  float weight = 0;
  if (newDataReady) {
    if (millis() > t + serialPrintInterval) {
      weight = LoadCell.getData();
      newDataReady = 0;
      t = millis();
    }
  }

  // Send data over serial with commas separating the values
  Serial.print(distance1);
  Serial.print(",");
  Serial.print(distance2);
  Serial.print(",");
  Serial.print(weight);
  Serial.print(",");
  Serial.println(motion);

  delay(1000);  // Delay before next loop
}
```

C Flask API for Chess Logic

C.1 main.py

```
from flask_cors import CORS  # Import CORS
from flask import Flask, jsonify, request
import Image_capture
import threading
from stockfish import Stockfish
from generator import generate_fen, \
    generate_move_and_update_fen
import time
import serial
import speechTest
import serial
import time
import logging
import warnings
logging.basicConfig(level=logging.CRITICAL)
app = Flask(_name_)
CORS(app)

# Initialize the serial connection to the Arduino
SERIAL_PORT = '/dev/ttyACM0'  # Update this to your actual serial port
BAUD_RATE = 9600
SERIAL_PORT2 = '/dev/ttyAMA0'

ser = serial.Serial(SERIAL_PORT, BAUD_RATE)
ser2 = serial.Serial(SERIAL_PORT2, BAUD_RATE)
warnings.filterwarnings("ignore", message="ALSA lib pcm.c:")
warnings.filterwarnings("ignore", message="Cannot connect to server
socket")
# Start the camera thread
camera_thread = threading.Thread(target=Image_capture.initialize_camera)
camera_thread.daemon = True
camera_thread.start()

# Game-related variables
current_fen = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w - - 0 1"
move = ""
next_fen = ""

# Motion inactivity tracker
last_motion_time = time.time()  # Initialize with the current time
MOTION_TIMEOUT = 120  # Timeout in seconds (2 minutes)

def read_sensor_data():
    ser.flushInput()
    line = ser.readline().decode('utf-8').strip()
    return map(float, line.split(','))


def update_fen_after_move(fen, move):
    try:
        print(f"Original FEN: {fen}")
        print(f"Move to apply: {move}")
```

```
        board_part, turn, castling, en_passant, halfmove_clock,
fullmove_number = fen.split()

        # Map board rows to a list of lists
        board_rows = [list(row.replace('8', '11111111').replace('7',
'1111111').replace('6', '111111')
                           .replace('5', '11111').replace('4',
'1111').replace('3', '111').replace('2', '11')
                           .replace('1', '1')) for row in
board_part.split('/')]

        # Parse move (e.g., e2e4 or e7e5)
        start_square = move[:2]
        end_square = move[2:]

        start_rank = 8 - int(start_square[1])
        start_file = ord(start_square[0]) - ord('a')

        end_rank = 8 - int(end_square[1])
        end_file = ord(end_square[0]) - ord('a')

        # Move the piece
        piece = board_rows[start_rank][start_file]
        board_rows[start_rank][start_file] = '1'  # Clear the start
square
        board_rows[end_rank][end_file] = piece  # Place the piece on the
end square

        # Convert the board back to FEN format
        new_board_part = '/'.join([''.join(row).replace('11111111',
'8').replace('1111111', '7')
                                   .replace('111111',
'6').replace('11111', '5').replace('1111', '4')
                                   .replace('111', '3').replace('11',
'2').replace('1', '1')
                                    for row in board_rows])

        # Switch the turn
        new_turn = 'w' if turn == 'b' else 'b'

        # Update halfmove clock and fullmove number
        if piece.lower() == 'p' or board_rows[end_rank][end_file] != '1':
# pawn move or capture
            new_halfmove_clock = '0'
        else:
            new_halfmove_clock = str(int(halfmove_clock) + 1)

        if turn == 'b':
            new_fullmove_number = str(int(fullmove_number) + 1)
        else:
            new_fullmove_number = fullmove_number

        # Construct the new FEN
        new_fen = f"{new_board_part} {new_turn} {castling} {en_passant}
{new_halfmove_clock} {new_fullmove_number}"
```

```
        print(f"Updated FEN: {new_fen}")

        return new_fen

    except Exception as e:
        print(f"Error in update_fen_after_move: {e}")
        raise


@app.route('/process', methods=['GET'])
def process_request():
    global current_fen
    global next_fen
    global last_motion_time

    try:
        # Check if the game is paused
        if speechTest.get_game_status() == 0:
            return jsonify({"error": "Game is paused, resume it to
play"}), 400

        print(f"current fen: {current_fen} \n")

        # Read sensor data
        distance1, distance2, weight, motion = read_sensor_data()

        # Check for motion inactivity
        if motion == 0:
            # If no motion, check how long it's been since motion was
last detected
            if time.time() - last_motion_time > MOTION_TIMEOUT:
                return jsonify({"error": "Player left the game"}), 400
        else:
            # Motion detected, reset the last motion time
            last_motion_time = time.time()

        # Define acceptable range for sensor data
        MIN_DISTANCE = 0  # Example minimum distance in cm
        MAX_DISTANCE = 1000  # Example maximum distance in cm
        MIN_DISTANCE2 = 75  # Example minimum distance in cm
        MAX_DISTANCE2 = 83  # Example maximum distance in cm
        MIN_WEIGHT = 10  # Example minimum weight
        MAX_WEIGHT = 1000  # Example maximum weight

        # Check if sensor data is within the acceptable range
        if not (MIN_DISTANCE2 <= distance2 <= MAX_DISTANCE2):
            return jsonify("Obstacles detected by sensor 2"), 400

        elif not (MIN_DISTANCE <= distance1 <= MAX_DISTANCE):
            return jsonify("Obstacles detected by sensor 1"), 400

        elif not (MIN_WEIGHT <= weight <= MAX_WEIGHT):
            return jsonify("Piece not detected for retrieval"), 400

        else:
```

```
            save_dir = "Full_boards"
            Image_capture.capture_and_process_image(save_dir)  # Capture
the image

            # Generate the new FEN, changes, and validate the move
            next_fen, changes, extracted_move, is_legal =
generate_fen(current_fen)
            print(f"next_fen: {next_fen} \n"
                  f"changes: {changes} \n"
                  f"extracted move: {extracted_move} \n"
                  f"is legal: {is_legal} \n")

            if changes == 2:
                if is_legal:
                    return jsonify(next_fen), 200
                else:
                    return jsonify("Illegal move"), 400

            elif changes < 5:
                return jsonify(next_fen), 200

            else:
                return jsonify({"error": "Too many changes"}), 400

    except Exception as e:
        return jsonify({"error": str(e)}), 500
@app.route('/setFen', methods=['POST'])
def set_fen():
    global current_fen
    global next_fen

    try:
        data = request.json
        if not data:
            return jsonify({"error": "No data received"}), 400

        fen = data.get('fen')
        if not fen:
            return jsonify({"error": "FEN is missing in the request
data"}), 400

        # Add missing FEN components
        # Assuming it's Black's turn and we're at the start of the game
        fen_complete = f"{fen} b KQkq - 0 1"

        print(f"Received FEN: {fen}")
        print(f"Complete FEN: {fen_complete}")

        # Set the received FEN as the current FEN
        current_fen = fen_complete

        # Generate the best move against the current FEN (assuming it's
Black's turn)
        result = generate_move_and_update_fen(current_fen)

        # Ensure the result is correctly unpacked
```

```
        if result and len(result) == 4:
            updated_fen, move, move_type, is_checkmate = result
            command = f"{move}{move_type}".upper()

        # Send the command to the Arduino over serial
            ser2.write(command.encode())
 print(f"Updated FEN: {updated_fen}"
                  f"checkmate?: {is_checkmate}")

        else:
            return jsonify({"error": "Failed to generate move"}), 500

        # Set the updated FEN as the new current FEN (White to move now)
        current_fen = updated_fen

        # Send the updated FEN and move back to the frontend
        return jsonify({"updated_fen": updated_fen}), 200
    except Exception as e:
        print(f"Error occurred: {e}")  # Debugging: Print the exception
message
        return jsonify({"error": str(e)}), 500

@app.route('/confirmFen', methods=['POST'])
def confirm_fen():
    global current_fen
    global next_fen

    try:

        # Apply the move to the FEN to generate the new FEN
        updated_fen, done_move, move_type, is_checkmate =
generate_move_and_update_fen(next_fen)

        print(f"Updated FEN: {updated_fen}"
              f"checkmate?: {is_checkmate}")

        # Update current FEN to the new one
        current_fen = updated_fen
        print(f"cuurent fen : {current_fen} ")

        # Clear next_fen and move since they've been applied
        next_fen = ""
        command = f"{done_move}{move_type}".upper()

        # Send the command to the Arduino over serial
        ser2.write(command.encode())

        # Send the new FEN after the move back to the frontend
        return jsonify(current_fen), 200

    except Exception as e:
        print(f"Error occurred: {e}")
        return jsonify({"error": str(e)}), 500

if _name_ == '_main_':
    app.run(host='0.0.0.0', port=5050)
```

C.2 Image_divide_preprocess.py

```
import os
import time
from stockfish import Stockfish
import cv2
import numpy as np

def rotate_and_save_image(image_path, initial_angle, output_filename):
    image = cv2.imread(image_path)

    (h, w) = image.shape[:2]

    center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, initial_angle, 1.0)
    cos_theta = np.abs(M[0, 0])
    sin_theta = np.abs(M[0, 1])
    new_w = int((h * sin_theta) + (w * cos_theta))
    new_h = int((h * cos_theta) + (w * sin_theta))
    M[0, 2] += (new_w / 2) - center[0]
    M[1, 2] += (new_h / 2) - center[1]
    rotated_image = cv2.warpAffine(image, M, (new_w, new_h))

    # Save the resultant image
    cv2.imwrite(output_filename, rotated_image)
  print(f"Resultant image saved as '{output_filename}'.")


def divide_static_chessboard(image_path, num_rows, num_cols, output_dir):
    corners = [(1078, 172), (2934, 172), (2934, 2028), (1078, 2028)]
    image = cv2.imread(image_path)

    # Define the destination points for perspective transform
    width, height = 1856, 1856
    dst_pts = np.float32([
        [0, 0],
        [width - 1, 0],
        [width - 1, height - 1],
        [0, height - 1]
    ])

    M = cv2.getPerspectiveTransform(np.float32(corners), dst_pts)
    warped_image = cv2.warpPerspective(image, M, (width, height))

    cell_height = height // num_rows
    cell_width = width // num_cols

    os.makedirs(output_dir, exist_ok=True)

    # Prepare the file names in the correct order (row by row from top to
bottom)
    file_names = ['h8', 'h7', 'h6', 'h5', 'h4', 'h3', 'h2', 'h1',
                  'g8', 'g7', 'g6', 'g5', 'g4', 'g3', 'g2', 'g1',
                   'f8', 'f7', 'f6', 'f5', 'f4', 'f3', 'f2', 'f1',
                   'e8', 'e7', 'e6', 'e5', 'e4', 'e3', 'e2', 'e1',
```

```
                  'd8', 'd7', 'd6', 'd5', 'd4', 'd3', 'd2', 'd1',
                  'c8', 'c7', 'c6', 'c5', 'c4', 'c3', 'c2', 'c1',
                  'b8', 'b7', 'b6', 'b5', 'b4', 'b3', 'b2', 'b1',
                  'a8', 'a7', 'a6', 'a5', 'a4', 'a3', 'a2', 'a1']

    for i in range(num_rows):
        for j in range(num_cols):
            x_start = j * cell_width
            y_start = i * cell_height
            cell = warped_image[y_start:y_start + cell_height,
x_start:x_start + cell_width]
            cell_filename = os.path.join(output_dir, f'{file_names[i *
num_cols + j]}.jpg')
            cv2.imwrite(cell_filename, cell)

    print(f"Chessboard divided into {num_rows * num_cols} cells and saved
in '{output_dir}'.")

    # Draw grid lines on the warped image for visualization
    for i in range(1, num_rows):
        y = i * cell_height
        cv2.line(warped_image, (0, y), (width, y), (0, 255, 0), 2)  #
Horizontal lines

    for j in range(1, num_cols):
        x = j * cell_width
        cv2.line(warped_image, (x, 0), (x, height), (0, 255, 0), 2)  #
Vertical lines

    # Save the image with grid lines
    # cv2.imwrite(os.path.join(output_dir,
'DividedChessboardWithLines.jpg'), warped_image)

    # Optionally show the image
    # cv2.imshow("Divided Chessboard with Lines", warped_image)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    #print(file_names)

def board_divider(image_path):
    initial_angle = -1  # Angle to rotate the image initially
    output_filename = 'resultant_image.jpg'  # Output file name
    rotate_and_save_image(image_path, initial_angle, output_filename)
    divide_static_chessboard(output_filename, 8, 8, 'divided_output')
```

C.3 Image_capture.py

```
import os
import cv2 as cv
import numpy as np
import depthai as dai
import datetime
import threading
import Image_divide_preprocess

latest_frame = None
frame_lock = threading.Lock()

def initialize_camera():
    global latest_frame, frame_lock
    try:
        print("Setting up pipeline...")
        pipeline = dai.Pipeline()

        # Define a source - color camera
        cam_rgb = pipeline.createColorCamera()
        cam_rgb.setBoardSocket(dai.CameraBoardSocket.RGB)

cam_rgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_13_M
P)

        # Adjust camera settings for better quality
        cam_rgb.setIspScale(1, 1)  # Full resolution

        # Create output
        xout_video = pipeline.createXLinkOut()
        xout_video.setStreamName("video")
        cam_rgb.video.link(xout_video.input)

        # Create an input for camera control
        cam_control = pipeline.createXLinkIn()
        cam_control.setStreamName("control")
        cam_control.out.link(cam_rgb.inputControl)

        # Connect to device and start pipeline
        with dai.Device(pipeline) as device:
            print("Pipeline started, allowing camera to focus...")
            q_rgb = device.getOutputQueue(name="video", maxSize=4,
blocking=True)
            q_control = device.getInputQueue(name="control")

            # Configure initial manual settings
            control = dai.CameraControl()
            control.setManualExposure(20000, 100)  # Exposure time (  s),
ISO

control.setAutoFocusMode(dai.CameraControl.AutoFocusMode.CONTINUOUS_VIDEO
)

control.setAutoWhiteBalanceMode(dai.CameraControl.AutoWhiteBalanceMode.AU
TO)
```

```
control.setAntiBandingMode(dai.CameraControl.AntiBandingMode.MAINS_60_HZ)
            control.setBrightness(1)
            control.setContrast(2)
            control.setSaturation(1)
            control.setSharpness(1)
            control.setLumaDenoise(1)
            control.setChromaDenoise(0)
            # control.setSceneMode(dai.CameraControl.SceneMode.LANDSCAPE)

            # Apply initial controls
            q_control.send(control)

            while True:
                in_rgb = q_rgb.get()
                frame = in_rgb.getCvFrame()

                with frame_lock:
                    latest_frame = frame.copy()

                try:
                    # Uncomment this line if you want to see the video
feed
                    # cv.imshow("video", frame)
                    pass
                except cv.error as e:
                    print(f"OpenCV error: {e}")

                if cv.waitKey(1) == ord('q'):
                    break
    except Exception as e:
        print(f"Error capturing image: {e}")
        raise

def capture_and_process_image(save_dir):
    global latest_frame, frame_lock
    try:
        with frame_lock:
            if latest_frame is None:
                raise Exception("No frame captured yet")

            frame = latest_frame.copy()
            #timestamp =
datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            image_name = f"full_board.png"
            if not os.path.exists(save_dir):
                os.makedirs(save_dir)
            cv.imwrite(os.path.join(save_dir, image_name), frame)
            print(f"Image saved as {os.path.join(save_dir, image_name)}")
            cv.destroyAllWindows()
            Image_divide_preprocess.board_divider(os.path.join(save_dir,
image_name))

    except Exception as e:
        print(f"Error capturing image: {e}")
        raise
```

C.4 modelTest.py

```
import cv2
from ultralytics import YOLO

# Load the model
model = YOLO('../model/weights/best.pt')
class_names = ['bishop', 'black', 'blank-piece', 'king', 'knight',
'pawn', 'queen', 'rook']

def warm_up_model():
    # Create a small dummy image
    dummy_image = cv2.imread('divided_output/g7.jpg')  # Replace with a
valid small image path
    if dummy_image is None:
        print("Dummy image not found, skipping warm-up.")
        return

    # Run a dummy prediction
    model.predict(dummy_image, conf=0.5, iou=0.7, max_det=1, imgsz=96)
    print("Model warm-up completed.")

# Call the warm-up function
warm_up_model()
def detect_obj(image_path_cropped):
    image_path = image_path_cropped
    results = model.predict(image_path, conf=0.5, iou=0.7, max_det=1,
save_conf=True, imgsz=96)

    # Load the image using OpenCV
    image = cv2.imread(image_path)

    detections = []
    if isinstance(results, list):
        for result in results:
            if hasattr(result, 'boxes') and result.boxes:
                for box in result.boxes:
                    if hasattr(box, 'cls') and hasattr(box, 'conf'):
                        detections.append({
                            "class": class_names[int(box.cls)],
                            "confidence": float(box.conf)
                        })
    else:
        print("Unexpected results type:", type(results))

    # Debug: Check the contents of detections
    if not detections:
        detections.append({
            "class": "rook",
            "confidence": 1.0  # Assuming a high confidence for the
default class
        })
    else:
        print(f"Detections found for {image_path}: {detections}")

    return detections
```

C.5 modelInference.py

```
import os
import modelTest
from concurrent.futures import ThreadPoolExecutor, as_completed
import time

def process_image(image_path):
    try:
        # Detect objects in the image
        detections = modelTest.detect_obj(image_path)

        # Initialize detected_class and conf as None
        detected_class = None
        conf = None

        # Check if detections is a non-empty list
        if detections and isinstance(detections, list) and
len(detections) > 0:
            # Safely access the first element of the list
            first_detection = detections[0]
            detected_class = first_detection.get('class')
            conf = first_detection.get('confidence')

        # Return the results
        return {
            'image': os.path.basename(image_path),
            'detected_class': detected_class,
            'conf': conf
        }

    except Exception as e:
        # Handle errors in detection
        print(f"Error processing image {image_path}: {e}")
        return {
            'image': os.path.basename(image_path),
            'detected_class': 'error',
            'conf': None
        }

def process_images_detections(directory):
    # Get the list of image files sorted alphabetically
    image_files = [os.path.join(directory, f) for f in
sorted(os.listdir(directory)) if
                   f.endswith(('.jpg', '.jpeg', '.png'))]

    # Check if the directory contains exactly 64 images
    if len(image_files) != 64:
        raise ValueError("The directory should contain exactly 64
images.")

    detections_list = []

    # Use ThreadPoolExecutor for multithreading
    with ThreadPoolExecutor() as executor:
        # Submit tasks to the executor
```

```
        future_to_image = {executor.submit(process_image, image_path):
image_path for image_path in image_files}

        for future in as_completed(future_to_image):
            image_path = future_to_image[future]
            try:
                result = future.result()
                detections_list.append(result)
            except Exception as e:
                print(f"Error processing image {image_path}: {e}")
                detections_list.append({
                    'image': os.path.basename(image_path),
                    'detected_class': 'error',
                    'conf': None
                })
    sorted_detections_list = sorted(detections_list, key=lambda x:
x['image'] or '')

    return sorted_detections_list
```

C.6 generator.py

```
import itertools
import modelInference
from stockfish import Stockfish

def update_fen_square(fen: str, square: str, piece: str) -> str:
    piece_map = {
        'pawn': 'P',
        'queen': 'Q',
        'king': 'K',
        'rook': 'R',
        'bishop': 'B',
        'knight': 'N',
        'blank-piece': '1'
    }

    piece = piece_map[piece.lower()]
    rows = fen.split(' ')[0].split('/')
    col_map = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f': 5, 'g': 6,
'h': 7}
    rank_map = {'8': 0, '7': 1, '6': 2, '5': 3, '4': 4, '3': 5, '2': 6,
'1': 7}

    col = col_map[square[0]]
    rank = rank_map[square[1]]

    rank_list = []
    for ch in rows[rank]:
        if ch.isdigit():
            rank_list.extend(['1'] * int(ch))
        else:
            rank_list.append(ch)

    rank_list[col] = piece

    new_rank = []
    empty_count = 0
    for ch in rank_list:
        if ch == '1':
            empty_count += 1
        else:
            if empty_count > 0:
                new_rank.append(str(empty_count))
                empty_count = 0
            new_rank.append(ch)
    if empty_count > 0:
        new_rank.append(str(empty_count))

    rows[rank] = ''.join(new_rank)

    updated_fen = '/'.join(rows) + ' ' + ' '.join(fen.split(' ')[1:])
    return updated_fen

def extract_move_from_fens(current_fen, new_fen):
    # Initialize Stockfish instances
```

```
    stockfish_current = Stockfish()
    stockfish_new = Stockfish()

    # Set FENs into each instance
    stockfish_current.set_fen_position(current_fen)
    stockfish_new.set_fen_position(new_fen)

    move_from = None
    move_to = None

    # Iterate over all squares on the board
    for file in 'abcdefgh':
        for rank in '12345678':
            square = file + rank

            piece_current =
stockfish_current.get_what_is_on_square(square)
            piece_new = stockfish_new.get_what_is_on_square(square)

            if piece_current is not None and piece_new is None:
                # The piece moved from this square (it was occupied and
now it's empty)
                move_from = square
            elif piece_current is None and piece_new is not None:
                # The piece moved to this square (it was empty and now
it's occupied)
                move_to = square
            elif piece_current is not None and piece_new is not None and
piece_current != piece_new:
                # A capture occurred (the square was occupied by a piece
and now it's occupied by another piece)
                # move_from = square
                move_to = square

    if move_from and move_to:
        return move_from + move_to
    else:
        raise ValueError("Could not extract a move from the FEN
comparison.")

def expand_fen_row(fen_row):
    """Expands a compressed FEN row into its full representation."""
    expanded_row = ""
    for char in fen_row:
        if char.isdigit():
            expanded_row += '1' * int(char)
        else:
            expanded_row += char
    return expanded_row

def generate_fen(current_fen):
    fen_parts = current_fen.strip().split()
    board_part = fen_parts[0]
    turn = 'b'  # Ensure it's Black's turn
    castling_availability = fen_parts[2]
    en_passant_target = fen_parts[3]
```

```
    halfmove_clock = fen_parts[4]
    fullmove_number = fen_parts[5]

    file_names = [
        'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8',
        'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8',
        'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8',
        'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8',
        'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8',
        'f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8',
        'g1', 'g2', 'g3', 'g4', 'g5', 'g6', 'g7', 'g8',
        'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8'
    ]

    detections =
modelInference.process_images_detections('divided_output')

    original_board_part = board_part
    changes = 0

    for i, detection in enumerate(detections):
        detected_class = detection['detected_class']
        square = file_names[i]

        if detected_class == 'black':
            continue
        elif detected_class == 'blank-piece':
            updated_fen = update_fen_square(board_part, square, 'blank-
piece')
        else:
            updated_fen = update_fen_square(board_part, square,
detected_class)

        updated_board_part = updated_fen.split(' ')[0]

        if original_board_part != updated_board_part:
            changes += 1
            board_part = updated_fen
            original_board_part = updated_board_part
 new_fen = ' '.join([board_part, turn, castling_availability,
en_passant_target, halfmove_clock, fullmove_number])

    # Extract the move from the FEN comparison
    try:
        move = extract_move_from_fens(current_fen, new_fen)

    except ValueError as e:

        move = None

    # Validate the move with Stockfish
    stockfish = Stockfish()
    stockfish.set_fen_position(current_fen)

    is_legal = stockfish.is_move_correct(move) if move else False
```

```
    return new_fen, changes, move, is_legal


def update_fen_after_move(fen, move):
    board_part, turn, castling, en_passant, halfmove_clock,
fullmove_number = fen.split()

    # Map board rows to a list of lists
    board_rows = [list(row.replace('8', '1' * 8).replace('7', '1' *
7).replace('6', '1' * 6)
                       .replace('5', '1' * 5).replace('4', '1' *
4).replace('3', '1' * 3)
                       .replace('2', '1' * 2)) for row in
board_part.split('/')]

    # Parse move (e.g., e2e4 or e7e5)
    start_square = move[:2]
    end_square = move[2:]

    start_rank = 8 - int(start_square[1])
    start_file = ord(start_square[0]) - ord('a')

    end_rank = 8 - int(end_square[1])
    end_file = ord(end_square[0]) - ord('a')

    # Move the piece
    piece = board_rows[start_rank][start_file]
    board_rows[start_rank][start_file] = '1'
    board_rows[end_rank][end_file] = piece

    # Convert the board back to FEN format
    new_board_part = '/'.join([''.join(row).replace('11111111',
'8').replace('1111111', '7')
                               .replace('111111', '6').replace('11111',
'5').replace('1111', '4')
                               .replace('111', '3').replace('11',
'2').replace('1', '1')
                                for row in board_rows])

    # Switch the turn
    new_turn = 'w' if turn == 'b' else 'b'

    # Update halfmove clock and fullmove number
    if piece.lower() == 'p' or board_rows[end_rank][end_file] != '1':  #
pawn move or capture
        new_halfmove_clock = '0'
    else:
        new_halfmove_clock = str(int(halfmove_clock) + 1)

    if turn == 'b':
        new_fullmove_number = str(int(fullmove_number) + 1)
    else:
        new_fullmove_number = fullmove_number

    # Construct the new FEN
```

```
    new_fen = f"{new_board_part} {new_turn} {castling} {en_passant}
{new_halfmove_clock} {new_fullmove_number}"

    return new_fen

def generate_move_and_update_fen(fen):
    # Initialize Stockfish
    stockfish = Stockfish()
  # Set the FEN position
    stockfish.set_fen_position(fen)
    print(f"Generating move for FEN: {fen}")

    # Get the best move
    move = stockfish.get_best_move()
    if not move:
        raise ValueError("No valid move could be generated.")

    print(f"Generated move: {move}")

    try:
        # Parse the FEN to determine the board state
        board_part, turn, castling, en_passant, halfmove_clock,
fullmove_number = fen.split()
    except ValueError as e:
        print(f"Error parsing FEN: {e}")
        print(f"Received FEN: {fen}")
        raise

    # Create a mapping of board positions
    board_rows = [list(row.replace('8', '1'*8).replace('7',
'1'*7).replace('6', '1'*6)
                       .replace('5', '1'*5).replace('4',
'1'*4).replace('3', '1'*3)
                       .replace('2', '1'*2)) for row in
board_part.split('/')]

    # Parse the move
    start_square = move[:2]
    end_square = move[2:]

    start_rank = 8 - int(start_square[1])
    start_file = ord(start_square[0]) - ord('a')

    end_rank = 8 - int(end_square[1])
    end_file = ord(end_square[0]) - ord('a')

    # Check if the destination square was occupied
    destination_occupied = board_rows[end_rank][end_file] != '1'

    # Manually update the FEN with the generated move
    updated_fen = update_fen_after_move(fen, move)
    print(f"Updated FEN after move: {updated_fen}")

    # Determine move type
    move_type = 'm'  # Default to regular move
    if len(move) == 5:
```

```
            move_type = 'r'
            move = move[:4]  # Castling
        elif destination_occupied:
            move_type = 'c'  # Capture

        # Check if the move results in checkmate
        stockfish.set_fen_position(updated_fen)
        evaluation_after_move = stockfish.get_evaluation()
        is_checkmate = evaluation_after_move['type'] == 'mate'

        return updated_fen, move, move_type, is_checkmate
```

C.7 speechTest.py

```
import speech_recognition as sr
import os
import sys
import threading
import warnings
import contextlib
import os
import logging
logging.basicConfig(level=logging.CRITICAL)
os.environ['ALSA_QUIET'] = '1'
# Suppress ALSA lib errors
sys.stderr = open(os.devnull, 'w')
sys.stdout = open(os.devnull, 'w')

warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", message="ALSA")
warnings.filterwarnings("ignore", message="JackShmReadWritePtr")
warnings.filterwarnings("ignore", message="Cannot connect to server")
warnings.filterwarnings("ignore", message="Cannot connect to server
socket")
# Initialize the recognizer
recognizer = sr.Recognizer()

# Initialize game status
game_status = 1  # 0 for paused, 1 for resumed

def listen_for_command():
    global game_status

    try:
        with sr.Microphone() as source:
            print("Listening for 'resume' or 'pause'...")

            # Adjust for ambient noise and listen
            recognizer.adjust_for_ambient_noise(source)
            audio = recognizer.listen(source)

            # Recognize speech using Google's Speech Recognition
            command = recognizer.recognize_google(audio).lower()
            print(f"Detected command: {command}")
```

```
            if "resume" in command:
                game_status = 1
                print("Game status set to: Resume")
            elif "pause" in command:
                game_status = 0
                print("Game status set to: Pause")

    except sr.UnknownValueError:
        print("Could not understand the audio.")
    except sr.RequestError as e:
        print(f"Could not request results; {e}")

def get_game_status():
    return game_status

def start_listening():
    while True:
        listen_for_command()

speech_thread = threading.Thread(target=start_listening)
speech_thread.daemon = True  # This makes sure the thread exits when the
main program does
speech_thread.start()
```

D Arduino Mega (Motors Controler)

```
#include <Servo.h>
#include <Stepper.h>

// Servo motors
Servo servo1;
Servo servo2;
Servo gripper; // Declare the gripper servo

int i=0;

// Stepper motor control pins for first stepper
const int stepper1_enablePin = 4;
const int stepper1_dirPin = 2;
const int stepper1_pulsePin = 3;

// Stepper motor control pins for second stepper
const int stepper2_enablePin = 5;
const int stepper2_dirPin = 6;
const int stepper2_pulsePin = 7;

// Define steps per revolution for your stepper motors
const int stepsPerRevolution = 200;

// Initialize the stepper library on the stepper motor control pins
Stepper stepper1(stepsPerRevolution, stepper1_pulsePin, stepper1_dirPin);
Stepper stepper2(stepsPerRevolution, stepper2_pulsePin, stepper2_dirPin);

// Initialize arrays for servo and stepper positions
int servo1_steps[64] = {170, /B1/170, /C1/170, /D1/170, /E1/170, /F1/170,
/G1/170, /H1/170,/A2/ 145, /B2/110, /C2/105, /D2/98, /E2/96, /F2/98, /
G2/102, /H2/108,
                        /A3/90, /B3/83, /C3/62, /D3/60, /E3/57, /F3/55, /
G3/65, /H3/75, /A4/60, /B4/45, /C4/38, /D4/34,/E4/ 34, /F4/34, /G4/40, /
H4/45,
                        /A5/40,/B5/ 28, /C5/21, /D5/18, /E5/15, /F5/18, /
G5/22, /H5/27, /A6/30, /B6/15, /C6/7, /D6/2, /E6/0, /F6/1, /G6/7, /H6/16,
                        /A7/16, /B7/4, /C7/0, /D7/0, /E7/0, /F7/0, /G7/1,
/H7/ 4, /A8/ 8, /B8/0, /C8/0, /D8/0, /E8/0, /F8/0, /G8/0, /H8/ 3};

int servo2_steps[64] = {80, /B1/86, /C1/84, /D1/85, /E1/87, /F1/85, /
G1/83, /H1/87, /A2/96, /B2/130, /C2/130, /D2/130, /E2/133, /F2/132, /
G2/132, /H2/128,
                        /A3/140,/B3/ 142, /C3/170, /D3/160, /E3/165, /
F3/168, /G3/158, /H3/152, /A4/180, /B4/180, /C4/180, /D4/180, /E4/180, /
F4/180, /G4/180, /H4/180,
                        /A5/180, /B5/180, /C5/180, /D5/180, /E5/180, /
F5/180, /G5/180, /H5/180, /A6/180, /B6/180, /C6/180, /D6/180, /E6/180, /
F6/180, /G6/180, /H6/180,
                        /A7/180, /B7/180,/C7/163, /D7/153, /E7/145, /
F7/147,/G7/ 160, /H7/180, /A8/180, /B8/160, /C8/137, /D8/109, /E8/107, /
F8/118, /G8/140, /H8/ 160};
```

```
int stepper1_steps[64] = {515, /B1/515, /C1/515, /D1/515, /E1/515, /
F1/515, /G1/515, /H1/515, /A2/420, /B2/350, /C2/340, /D2/315, /E2/315, /
F2/315, /G2/315, /H2/335,
                          /A3/295, /B3/275, /C3/275, /D3/235, /E3/235, /
F3/235, /G3/245,/H3/ 265, /A4/295, /B4/235, /C4/210, /D4/190, /E4/190, /
F4/193, /G4/205, /H4/235,
                          /A5/225, /B5/160, /C5/125, /D5/125, /E5/105,  /
F5/125, /G5/140, /H5/160, /A6/185, /B6/105, /C6/80, /D6/50, /E6/50, /
F6/65, /G6/85, /H6/115,
                          /A7/120, /B7/65, /C7/-35, /D7/-125, /E7/-155, /
F7/-145, /G7/-55, /H7/55,/A8/ 90, /B8/-60, /C8/-200, /D8/-380, /E8/-380,/
F8/ -330, /G8/-200,/H8/ -50};

int stepper2_steps[64] = {-520, /B1/-560, /C1/-620, /D1/-670, /E1/-735, /
F1/-785, /G1/-840, /H1/-900, /A2/-490, /B2/-540, /C2/-605, /D2/-675, /
E2/-745, /F2/-805, /G2/-865, /H2/-920,
                          /A3/-455 , /B3/-525, /C3/-600, /D3/-675, /
E3/-740, /F3/-815,/G3/ -885,/H3/ -945, /A4/-435, /B4/-500, /C4/-575, /
D4/-665, /E4/-755, /F4/-840, /G4/-910, /H4/-985,
                          /A5/-405, /B5/-480, /C5/-570, /D5/-670, /
E5/-775,  /F5/-878, /G5/-975, /H5/-1040, /A6/-355, /B6/-440, /C6/-545, /
D6/-680, /E6/-820, /F6/-935, /G6/-1040, /H6/-1133,
                          /A7/-290, /B7/ -390, /C7/-515, /D7/-655,/E7/
-835, /F7/-1010, /G7/-1140, /H7/-1210, /A8/-220, /B8/-315, /C8/-445, /
D8/-705, /E8/-980, /F8/-1150, /G8/-1280, /H8/-1330};

int pos1 = 180; // Initial position for servo 1
int pos2 = 110; // Initial position for servo 2
int gripperPos = 170; // Initial position for gripper servo (open)
int initialStepper1Steps = 0; // Initial steps for first stepper motor
int initialStepper2Steps = 0; // Initial steps for second stepper motor
int servoDelay = 15; // Delay in milliseconds between each step for
smoother movement

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

  // Attach the servos to their respective pins
  servo1.attach(8);
  servo2.attach(9);
  gripper.attach(10); // Attach the gripper servo

  // Set initial positions
  servo1.write(pos1);
  servo2.write(pos2);
  gripper.write(gripperPos); // Set gripper to initial position (open)

  // Set stepper motor speed
  stepper1.setSpeed(200);
  stepper2.setSpeed(200);

  // Set stepper motor control pins as outputs
  pinMode(stepper1_enablePin, OUTPUT);
  pinMode(stepper1_dirPin, OUTPUT);
  pinMode(stepper1_pulsePin, OUTPUT);
  pinMode(stepper2_enablePin, OUTPUT);
```

```
  pinMode(stepper2_dirPin, OUTPUT);
  pinMode(stepper2_pulsePin, OUTPUT);

  // Enable the stepper motors
  digitalWrite(stepper1_enablePin, LOW);
  digitalWrite(stepper2_enablePin, LOW);
  delay(2000);
  // Move the stepper motors to the initial positions
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, initialStepper1Steps);
  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, initialStepper2Steps);
  delay(2000);
  // Perform initial movements
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -700);
  delay(650);

  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(650);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(650);
}

void loop() {
  if (Serial.available() > 0) {
    String input = Serial.readString();
    input.trim();

    String start = input.substring(0, 2);
    String end = input.substring(2, 4);
    char action = input.charAt(4);

    int startIndex = chessBoardIndex(start);
    int endIndex = chessBoardIndex(end);

    if (startIndex >= 0 && startIndex < 64 && endIndex >= 0 && endIndex <
64) {
      if (isSpecialPosition(start) || isSpecialPosition(end)) {
        if (action == 'M') {
          movePieceSpecial(startIndex, endIndex);
        } else if (action == 'C') {
          capturePieceSpecial(startIndex, endIndex);
        }
        else if (action == 'R'){
          moveRetrieveSpecial(startIndex, endIndex);
        }
      } else {
        if (action == 'M') {
          movePiece(startIndex, endIndex);
        } else if (action == 'C') {
          capturePiece(startIndex, endIndex);
        } else {
```

```
          Serial.println("Invalid action. Use 'M' for move or 'C' for
capture.");
        }
      }
    } else {
      Serial.println("Invalid chess board position.");
    }
  }
}

bool isSpecialPosition(String pos) {
  return pos.equalsIgnoreCase("B8") || pos.equalsIgnoreCase("C8") ||
pos.equalsIgnoreCase("D8") ||
         pos.equalsIgnoreCase("E8") || pos.equalsIgnoreCase("F8") ||
pos.equalsIgnoreCase("G8") ||
         pos.equalsIgnoreCase("H8") || pos.equalsIgnoreCase("C7") ||
pos.equalsIgnoreCase("D7") ||
         pos.equalsIgnoreCase("E7") || pos.equalsIgnoreCase("F7") ||
pos.equalsIgnoreCase("G7") ||
         pos.equalsIgnoreCase("A3") || pos.equalsIgnoreCase("A2") ||
pos.equalsIgnoreCase("B3") ||
         pos.equalsIgnoreCase("B2") || pos.equalsIgnoreCase("D3") ||
pos.equalsIgnoreCase("E3") ||
         pos.equalsIgnoreCase("F3") || pos.equalsIgnoreCase("G3") ||
pos.equalsIgnoreCase("H3") ||
         pos.equalsIgnoreCase("C2") || pos.equalsIgnoreCase("D2") ||
pos.equalsIgnoreCase("E2") ||
         pos.equalsIgnoreCase("F2") || pos.equalsIgnoreCase("G2") ||
pos.equalsIgnoreCase("H2") ||
         pos.equalsIgnoreCase("A1") || pos.equalsIgnoreCase("B1") ||
pos.equalsIgnoreCase("C1") ||
         pos.equalsIgnoreCase("D1") || pos.equalsIgnoreCase("E1") ||
pos.equalsIgnoreCase("F1") ||
         pos.equalsIgnoreCase("G1") || pos.equalsIgnoreCase("H1");
}

void movePiece(int startIndex, int endIndex) {
  moveMotorsToPosition(startIndex); // Move to the start position and
grab the piece
  delay(650);

  moveMotorsToPosition(endIndex); // Move to the end position and release
the piece
}

void capturePiece(int startIndex, int endIndex) {


// Move second stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[endIndex]);
  delay(650);
```

```
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[endIndex]);
  pos2 = servo2_steps[endIndex];
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[endIndex]);
  pos1 = servo1_steps[endIndex];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[endIndex]);
  delay(650);

  // Close the gripper
  gripper.write(20);
  delay(650);

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
  Serial.println(servo1_steps[endIndex]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[endIndex]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[endIndex]);
  Serial.print("Stepper 2 steps: ");
  Serial.println(stepper2_steps[endIndex]);

  // Return to initial positions
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[endIndex]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(650);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[endIndex]);
  delay(650);
```

```
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  gripper.write(160); // Open the gripper to release the enemy piece
  delay(650);

  movePiece(startIndex, endIndex); // Then proceed with moving the
original piece
}

void movePieceSpecial(int startIndex, int endIndex) {
  moveMotorsSpecial(startIndex); // Move to the start position and grab
the piece
  delay(650);

  moveMotorsSpecial(endIndex); // Move to the end position and release
the piece
}

void capturePieceSpecial(int startIndex, int endIndex) {
 // Move second stepper motor

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[endIndex]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[endIndex]);
  pos1 = servo1_steps[endIndex];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[endIndex]);
  delay(650);

  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[endIndex]);
  pos2 = servo2_steps[endIndex];
  delay(650);

  // Close the gripper
  gripper.write(20);
  delay(650);

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
```

```
  Serial.println(servo1_steps[endIndex]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[endIndex]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[endIndex]);
  Serial.print("Stepper 2 steps: ");
  Serial.println(stepper2_steps[endIndex]);

  if (   endIndex == 50 || endIndex == 51 || endIndex == 52||
         endIndex == 53 || endIndex == 54 || endIndex == 57 ||
         endIndex == 58 || endIndex == 59 || endIndex == 60||
         endIndex == 61 || endIndex == 62 || endIndex == 63){

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
  delay(900);

  smoothServoMove(servo1, pos1, 45);
  pos1 = 45;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[endIndex]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[endIndex]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  // OPEN the gripper
  gripper.write(160);
  delay(650);

        }

  else{

    moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
```

```
  delay(900);

  smoothServoMove(servo1, pos1, 175);
  pos1 = 175;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[endIndex]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[endIndex]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  // OPEN the gripper
  gripper.write(160);
  delay(650);

  }
  movePieceSpecial(startIndex, endIndex); // Then proceed with moving the
original piece
}

void moveRetrieveSpecial(int startIndex, int endIndex){

// Move second stepper motor
  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[startIndex]);
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[startIndex]);
  pos1 = servo1_steps[startIndex];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[startIndex]);
  delay(650);
```

```
  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[startIndex]);
  pos2 = servo2_steps[startIndex];
  delay(650);

  // Close the gripper
  gripper.write(20);
  delay(650);

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
  Serial.println(servo1_steps[endIndex]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[endIndex]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[endIndex]);
  Serial.print("Stepper 2 steps: ");
  Serial.println(stepper2_steps[endIndex]);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
  delay(900);

  smoothServoMove(servo1, pos1, 175);
  pos1 = 175;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[startIndex]+200);
  delay(500);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[startIndex]);
  delay(650);

  // OPEN the gripper
  gripper.write(160);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -1370);
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, 35);
  pos1 = 35;
  delay(650);
```

```
  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 35);
  delay(650);

  // Move second servo
  smoothServoMove(servo2, pos2, 170);
  pos2 = 170;
  delay(650);

  // Close the gripper
  gripper.write(20);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -45);
  delay(650);

  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(650);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, 1370);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 110);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[endIndex]);
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[endIndex]);
  pos1 = servo1_steps[endIndex];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[endIndex]);
  delay(650);
```

```
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[endIndex]);
  pos2 = servo2_steps[endIndex];
  delay(650);

  // OPEN the gripper
  gripper.write(160);
  delay(650);

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
  Serial.println(servo1_steps[endIndex]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[endIndex]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[endIndex]);
  Serial.print("Stepper 2 steps: ");
  Serial.println(stepper2_steps[endIndex]);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -450);
  delay(900);

  smoothServoMove(servo1, pos1, 175);
  pos1 = 175;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[endIndex]+450);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[endIndex]);
  delay(900);

  // OPEN the gripper
  gripper.write(160);
  delay(650);

}

void moveMotorsToPosition(int index) {
```

```
  i=i+1;
  // Move second stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);
  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[index]);
  delay(650);
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);
  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[index]);
  pos2 = servo2_steps[index];
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[index]);
  pos1 = servo1_steps[index];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[index]);
  delay(650);

  if(i==1){
  // Close the gripper
  gripper.write(20);
  delay(650);
  }

  if(i==2){
  // Open the gripper
  gripper.write(160);
  i=0;
  delay(650);
  }

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
  Serial.println(servo1_steps[index]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[index]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[index]);
  Serial.print("Stepper 2 steps: ");
  Serial.println(stepper2_steps[index]);

  // Return to initial positions
  if (   index == 50 || index == 51 || index == 52||
         index == 53 || index == 54 || index == 57 ||
         index == 58 || index == 59 || index == 60||
         index == 61 || index == 62 || index == 63){
```

```
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
  delay(900);

  smoothServoMove(servo1, pos1, 45);
  pos1 = 45;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

        }
        else if( index == 0 || index == 1|| index == 2||
        index == 3 || index == 4 || index == 5 ||
        index == 6 || index == 7 || index == 8||
        index == 9 || index == 10 || index == 11 ||
        index == 12 || index == 13 || index == 14 ||
        index == 15 || index == 16 || index == 18||
        index == 19 || index == 20 || index == 21 ||
        index == 22){

          moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
          delay(900);

  smoothServoMove(servo1, pos1, 175);
  pos1 = 175;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);
```

```
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);


  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

        }

        else{

moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]);
  delay(650);

  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(650);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);
        }


}

// Function to move motors in a specific sequence for special positions
void moveMotorsSpecial(int index) {
  i=i+1;
```

```
  // Move second stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);
  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, stepper2_steps[index]);
  delay(650);
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);
  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -125);
  delay(650);

  // Move first servo
  smoothServoMove(servo1, pos1, servo1_steps[index]);
  pos1 = servo1_steps[index];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, stepper1_steps[index]);
  delay(650);

  // Move second servo
  smoothServoMove(servo2, pos2, servo2_steps[index]);
  pos2 = servo2_steps[index];
  delay(650);

  // Move first stepper motor
  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 125);
  delay(650);

  if(i==1){
  // Close the gripper
  gripper.write(20);
  delay(650);
  }

  if(i==2){
  // Open the gripper
  gripper.write(160);
  i=0;
  delay(650);
  }

  // Print the positions of each motor
  Serial.print("Servo 1 angle: ");
  Serial.println(servo1_steps[index]);
  Serial.print("Servo 2 angle: ");
  Serial.println(servo2_steps[index]);
  Serial.print("Stepper 1 steps: ");
  Serial.println(stepper1_steps[index]);
  Serial.print("Stepper 2 steps: ");
```

```
  Serial.println(stepper2_steps[index]);

  if (   index == 50 || index == 51 || index == 52||
         index == 53 || index == 54 || index == 57 ||
         index == 58 || index == 59 || index == 60||
         index == 61 || index == 62 || index == 63){

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
  delay(900);

  smoothServoMove(servo1, pos1, 45);
  pos1 = 45;
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

         }
         else if( index == 0 || index == 1|| index == 2||
         index == 3 || index == 4 || index == 5 ||
         index == 6 || index == 7 || index == 8||
         index == 9 || index == 10 || index == 11 ||
         index == 12 || index == 13 || index == 14 ||
         index == 15 || index == 16 || index == 18||
         index == 19 || index == 20 || index == 21 ||
         index == 22){

           moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -200);
           delay(900);

  smoothServoMove(servo1, pos1, 175);
  pos1 = 175;
```

```
  delay(1200);

  // Return to initial positions in reverse order
  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(900);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]+200);
  delay(900);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(900);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);

        }

        else{

moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -stepper1_steps[index]);
  delay(650);

  smoothServoMove(servo2, pos2, 180);
  pos2 = 180;
  delay(650);

  smoothServoMove(servo1, pos1, 20);
  pos1 = 20;
  delay(650);

  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, -100);
  delay(650);

  moveStepper(stepper2, stepper2_enablePin, stepper2_dirPin,
stepper2_pulsePin, -stepper2_steps[index]);
  delay(650);


  moveStepper(stepper1, stepper1_enablePin, stepper1_dirPin,
stepper1_pulsePin, 100);
  delay(650);

        }

}

// Function to convert chess board position to array index
int chessBoardIndex(String pos) {
  char column = pos.charAt(0);
  char row = pos.charAt(1);
```

```
  if (column < 'A' || column > 'H' || row < '1' || row > '8') {
    return -1;
  }

  int columnIndex = column - 'A';
  int rowIndex = row - '1';

  return rowIndex * 8 + columnIndex;
}

// Function to move the stepper motor a given number of steps
void moveStepper(Stepper &stepper, int enablePin, int dirPin, int
pulsePin, int steps) {
  digitalWrite(enablePin, LOW);  // Ensure the stepper motor is enabled
  digitalWrite(dirPin, steps > 0 ? HIGH : LOW);  // Set direction
  steps = abs(steps);

  for (int i = 0; i < steps; i++) {
    digitalWrite(pulsePin, HIGH);
    delayMicroseconds(2000);  // Adjust the delay to slow down the
stepper motor
    digitalWrite(pulsePin, LOW);
    delayMicroseconds(2000);  // Adjust the delay to slow down the
stepper motor
  }
}

// Function to smoothly move a servo from current position to target
position
void smoothServoMove(Servo &servo, int currentPos, int targetPos) {
  int step = (targetPos > currentPos) ? 1 : -1;
  for (int pos = currentPos; pos != targetPos; pos += step) {
    servo.write(pos);
    delay(14);  // Increase delay for slower movement
  }
  servo.write(targetPos);  // Ensure it reaches the exact target position
}
```

E Code for Web Interface

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chess Interface</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
chessboard-js/1.0.0/chessboard-1.0.0.min.css">
  <style>
    body {
      margin: 0;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background-color: #1a1a2e;
      color: white;
      display: flex;
      height: 100vh;
      overflow: hidden;
    }
    .container {
      display: flex;
      width: 100%;
    }
    .left-section, .right-section {
      height: 100vh;
    }
    .left-section {
      width: 70%;
      background-color: #162447;
      display: flex;
      align-items: center;
      justify-content: center;
      padding: 20px;
      box-shadow: inset 0 0 10px rgba(0, 0, 0, 0.5);
    }
    .right-section {
      width: 30%;
      background-color: #1f4068;
      display: flex;
      flex-direction: column;
      align-items: center;
      padding: 20px;
      box-shadow: inset 0 0 10px rgba(0, 0, 0, 0.5);
    }
    .done_button {
      display: flex;
      justify-content: center;
      width: 100%;
      margin-bottom: 20px;
    }
    .content {
      text-align: center;
      width: 100%;
    }
    .button {
```

```
      background-color: #e43f5a;
      color: white;
      border: none;
      padding: 10px;
      cursor: pointer;
      font-size: 1.2em;
      border-radius: 5px;
      transition: background-color 0.3s;
      width: 100%;
    }
    .button:hover {
      background-color: #b9344a;
    }
    #board {
      width: 100%;
      height: 100%;
      max-width: 100%;
      max-height: 100%;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.5);
      border-radius: 10px;
      transform: scale(0.65) translateY(-225px); /* Scale the board to
70% and lift it up by 30px */
      transform-origin: center; /* Center the scale transformation */
    }
    #fenInput {
      padding: 10px;
      font-size: 16px;
      width: 80%;
      margin: 10px 0;
      border: none;
      border-radius: 5px;
      display: none;
    }
    #fenLabel {
      margin-top: 20px;
      font-size: 14px;
    }

    /* Custom confirmation dialog styles */
    #confirmationDialog {
      display: none;
      position: fixed;
      left: 50%;
      top: 50%;
      transform: translate(-50%, -50%);
      background-color: #1f4068;
      color: white;
      padding: 20px;
      border-radius: 10px;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.5);
      z-index: 1000;
      text-align: center;
      width: 400px;
    }

    #confirmationDialog .message {
```

```
      font-size: 1.5em; /* Bigger font size */
      margin-bottom: 20px; /* Space between message and buttons */
    }

    #confirmationDialog .button-container {
      display: flex;
      justify-content: center;
      gap: 20px; /* Space between buttons */
    }

    #confirmationDialog button {
      background-color: #e43f5a;
      color: white;
      border: none;
      padding: 10px 20px;
      cursor: pointer;
      font-size: 1.2em;
      border-radius: 5px;
      transition: background-color 0.3s;
    }

    #confirmationDialog button:hover {
      background-color: #b9344a;
    }
  </style>
</head>
<body>
<div class="container">
  <div class="left-section">
    <div id="board"></div>
  </div>

  <div class="right-section">
    <div class="done_button">
      <button class="button" id="doneButton">Done Playing</button>
    </div>
    <div class="content">
      <input type="text" id="fenInput" placeholder="Enter FEN notation">
      <button class="button" id="setFenButton" style="display:none;">Set
FEN</button>
      <div id="fenLabel">Current FEN: </div>
    </div>

  </div>
</div>

<!-- Custom Confirmation Dialog -->
<div id="confirmationDialog">
  <div class="message">Is the updated FEN correct?</div>
  <div class="button-container">
    <button id="confirmYes">Yes</button>
    <button id="confirmNo">No</button>
  </div>
</div>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/
jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/chessboard-js/1.0.0/
chessboard-1.0.0.min.js"></script>
<script>
  document.addEventListener('DOMContentLoaded', (event) => {
    var board = Chessboard('board', {
      position: 'start',
      pieceTheme: 'https://chessboardjs.com/img/chesspieces/wikipedia/
{piece}.png',
      draggable: true,
      dropOffBoard: 'trash',
      sparePieces: true
    });

    function updateBoardSize() {
      var container = document.getElementById('board').parentNode;
      board.resize(container.clientWidth, container.clientHeight);
    }

    window.addEventListener('resize', updateBoardSize);
    updateBoardSize();

    function updateFenLabel() {
      var fen = board.fen();
      document.getElementById('fenLabel').textContent = 'Current FEN: ' +
fen;
    }

    function setFen(fen) {
      try {
        board.position(fen);
        updateFenLabel();
      } catch (error) {
        console.error("Invalid FEN notation:", error);
      }
    }

    document.getElementById('doneButton').addEventListener('click', () =>
{
      $.get('http://127.0.0.1:5050/process', (response) => {
        if (response.error) {
          alert(response.error);
        } else {
          setFen(response);  // Update the board immediately
          showConfirmationDialog(response);  // Show the custom
confirmation dialog
        }
      }).fail(() => {
        alert("Failed to process the board.");
      });
    });

    function showConfirmationDialog(fen) {
      var dialog = document.getElementById('confirmationDialog');
      dialog.style.display = 'block';
```

```
      document.getElementById('confirmYes').onclick = function () {
        dialog.style.display = 'none';
        $.ajax({
          url: 'http://127.0.0.1:5050/confirmFen',
          type: 'POST',
          contentType: 'application/json',  // Set content type to JSON
          data: JSON.stringify({}),  // No data needed, just confirming
          success: function(response) {
            setFen(response);  // Set the updated FEN after move is
applied
          },
          error: function(xhr, status, error) {
            alert("Failed to apply move. Error: " + error);
          }
        });
      };

      document.getElementById('confirmNo').onclick = function () {
        dialog.style.display = 'none';
        // If not confirmed, show the "Set FEN" button
        document.getElementById('setFenButton').style.display = 'block';
      };
    }

    document.getElementById('setFenButton').addEventListener('click', ()
=> {
      var fen = board.fen();  // Get the current FEN from the board
      $.ajax({
        url: 'http://127.0.0.1:5050/setFen',
        type: 'POST',
        contentType: 'application/json',  // Set content type to JSON
        data: JSON.stringify({ fen: fen }),  // Send the FEN as JSON
        success: function(response) {
          setFen(response.updated_fen);  // Update the board with the
response FEN
          alert("FEN sent to server and move generated. Updated FEN: " +
JSON.stringify(response.updated_fen));
          document.getElementById('setFenButton').style.display = 'none';
// Hide the "Set FEN" button after sending the FEN
        },
        error: function(xhr, status, error) {
          alert("Failed to send FEN. Error: " + error);
        }
      });
    });

    updateFenLabel();
  });
</script>
</body>
</html>
```