
**An-Najah National University**
**Faculty of Engineering & Information Technology**
**Computer Engineering Department**

# Ball-E

Prepared By:
**Sohaib Arafat**
**Osama Dweikat**

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

Supervised By:
**Dr. Samer Arandi**

January 31, 2025

# Acknowledgment

First of all, we would like to take this opportunity to thank our supervisor, Dr. Samer Arandi, for his time, effort, and valuable guidance that he has given to us during the course of this project. We would also like to thank the academic staff of the Computer Engineering Department for their knowledge and support over the years. We appreciate their guidance and encouragement in the completion of this project.

# Disclaimer

The following report has been authored by students Sohaib Arafat and Osama Dweikat 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

# Contents

# List of Figures

# Abstract

The project involves a robotic system for ball collection and sorting according to specified characteristics, such as color, size, or type. For example, tennis balls, ping pong balls, and beach balls. This robotic mobile arm will be able to move around the playing area, picking up the balls and sorting them without human interference.

The thing that differentiates our project from the others is the approach. We are trying to develop a custom Object Detection Model (ODM), for the detection and classification of the balls, which was trained on almost 30,000 images, supported with an OpenMV H7 or Luxonis OAK-D camera for the best outcome of the job. The arm will be equipped with various sensors-still to be defined-ensuring retrieval and sorting in an easy way, even for those balls whose size or type is different. It will be able to move in all directions with great efficiency due to the use of a stepper and servo motor.

The biggest challenge in this project will be the design of a gripping mechanism that will be able grasp different kinds of balls without damaging them or the robot rollover. The gripping arm should be adjusted to variations in ball size and material to ensure a secure but light grip. The system will also contain sensors for obstacle detection, such as ultrasonic or infrared sensors, with no interference from movement with the camera. A control panel gives the user commands for the robot. It will have a Raspberry Pi main controller for the ODM and camera, while another secondary microcontrollers are used to control the motors and manage sensors for the robotic arm. This ensures a fully automated process in the sorting and retrieval of the balls.

# 1 Introduction

The use of robotics and artificial intelligence has greatly improve automation and human machine collaboration and problem solving. These technologies are applied to autonomous sports equipment to help with time and effort in training through machines. This project aims to create an autonomous ball collecting robot that can work on sports fields and help players and coaches. The system uses an Oak-D camera and a custom ODM for real time image processing to accurately locate and pick up the ball. By letting the robot to navigate through the court and pick up the balls, the solution is trying to minimize the number of interruptions in the training sessions and the physical workload of players and coaches, with the hope of creating a better training environment.

## 1.1 Problem Statement

One of the biggest issues in sports training and recreational activities is the manual collection of balls which is time and energy consuming. Current practices are manual which means they are inefficient during training sessions and game play. This inefficiency results in practice time being completely wasted and an excess of physical strain being placed on players or members of staff who have to chase after balls. There is a clear need for an autonomous solution to this process to increase productivity.

## 1.2 Objective

BALL-E is an attempt at a simple, plug and play system that combines machine learning, robotics, and other components to provide an efficient, high performance ball collection experience. This system is therefore designed to run with minimal human intervention, in order to guarantee that balls are collected without interruption.

## 1.3 Scope of Work

1. **Optical Ball Detection System:** Use sophisticated computer vision capabilities through the Oak-D camera platform and implement a specialized ODM that can precisely and instantly identify and track balls all across the operational area.

2. **Intelligent Operation Features:** Develop a set of intelligent features that include environmental hazard detection, performance analytics, and control through wireless, with the aim of improving the operational efficiency and user experience.

3. **Advanced Collection Mechanism:** Design a sophisticated ball collection device with an intelligently engineered gripper that can fit different sizes of balls and is gentle on the balls and structurally sound even after repeated use.

4. **Centralized Management Platform:** Develop a web-based interface to manage all aspects of the system in a simple and effective manner, with real-time operational analytics and performance metrics and remote management capabilities to facilitate easy system administration.

## 1.4 Significance

Through the automation of the tedious and time-consuming activity of ball collection, the BALL-E project comes up as a significant contribution to sports training and recreational activities. It improves efficiency by cutting down on interruptions that occur during practice sessions thereby enabling athletes and coaches to devote their time to skill development instead of menial tasks.

# 2 Constraints and Earlier Coursework

## 2.1 Constrains

1. **Mechanical Constraints:** The system cannot grip or lift larger balls, including footballs, due to the high cost of 3D printing and the limitations of the geared DC motor, limiting its adaptability to different sports equipment.

2. **Computational Limitations:** The restrictions on the complexity of software features that can be implemented are imposed by the processing capabilities of the Raspberry Pi 3 and the Oak-D camera, making tasks like real-time live streaming of the playground unfeasible.

3. **Environmental Challenges:** Obstacles to system performance include lighting variations, surface irregularities, and weather conditions, which can affect the accuracy of ball detection and navigation.

4. **Power Supply Constraints:** The system's operation is limited by the battery capacity and power efficiency. Frequent recharging or battery replacement is required, which can be troublesome for continuous use during a long training session.

## 2.2 Earlier Coursework

1. **Image Processing:** Equipped us with essential knowledge on handling visual data, enabling the development of ball detection algorithms using real-time image analysis from the Oak-D camera.

2. **Critical Thinking:** Helped us develop problem-solving skills, allowing for effective troubleshooting, decision-making, and optimization of the robot's functionality under various conditions.

3. **Artificial Intelligence Course:** Gave us insights into machine learning techniques, which were important in training and fine-tuning the ODM used for accurate ball recognition.

4. **Microcontrollers Using PIC Course:** Built a solid foundation in working with microcontrollers, which was important for designing the control system of the robot and managing sensor interactions.

5. **Microcontrollers Lab:** Offered practical experience in programming and interfacing microcontrollers with different components, enabling efficient integration of hardware and software.

6. **Distributed Operating Systems Course:** Introduced concepts related to parallel processing and system resource management, which contributed to optimizing the robot's real-time operations. Additionally, this course enabled us to implement a mechanism that allows the robot to be operated remotely.

# 3 Literature Review

## 3.1 Autonomous Navigation in Sports Fields

Autonomous robots are increasingly being used in sports environments for example in ball picking, player tracking, and field demarcation. Different navigation strategies including path planning methods such as A* and Dijkstra's have been employed to find the most efficient path on the field. The use of ultrasonic, LiDAR, and vision-based SLAM sensors enhances the ability to detect and avoid obstacles. However, issues such as rough terrain, dynamic obstacles, and real-time decision-making remain an area of interest and need further development.

## 3.2 Computer Vision for Object Detection

Computer vision is an important part of object detection and tracking in sports. The ball can be detected very efficiently using techniques such as CNN and deep learning models like You Only Look Once (YOLO). The Oak-D camera, with its depth sensing capability, provides real-time information for localization and detection of the ball on the field. Despite these advances, the accuracy of detection is restricted by light variation, occlusion, and other environmental factors, thus the need for adaptive models and preprocessing techniques.

## 3.3 Similar Works

There are several ball-collecting robots for various sports such as tennis, golf, and soccer. These systems usually use vision-based detection and mechanical pickup. Some commercial products, such as robotic tennis ball collectors, use vacuum or rolling mechanisms to pick up the balls effectively. Nevertheless, most of the methods that are described above are not adaptable to different sizes and types of the balls, do not use dynamic navigation, and are quite expensive. Therefore, they are not available to private and small sports establishments. BALL-E tries to fill these gaps with an affordable, adaptive, and autonomous solution.

# 4 Methodology

## 4.1 System Structure

### 4.1.1 Wooden Cart

The cart used in the project has dimensions of 42x42x36 cm which provides a stable and compact platform to house essential components like motors, sensors and power supply. The cart has two 12cm in diameter wheels as the primary driving force and ensures that movement is smooth and controlled. Further two swivel ball casters are placed at the back of the cart to increase the stability and can be used to rotate the robot easily.

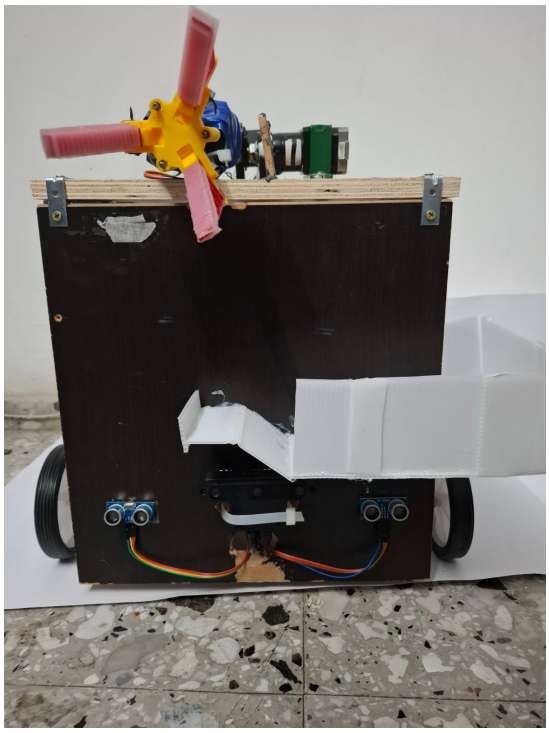

Figure 4.1: Wooden Cart Front-view

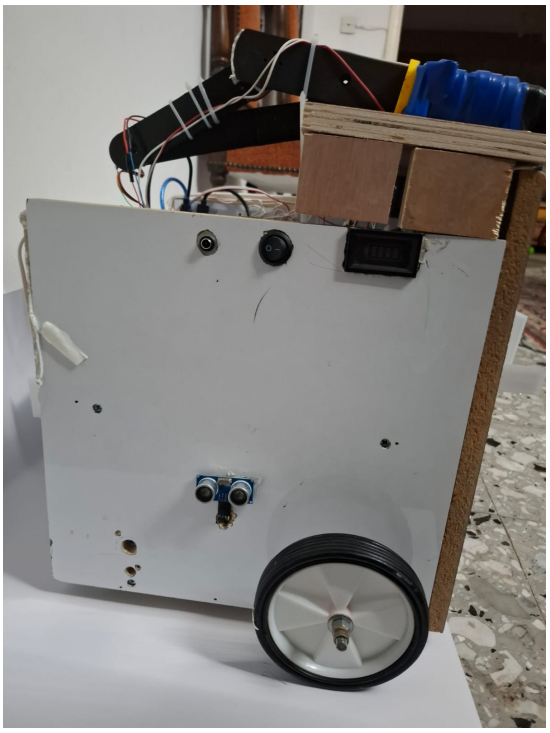

Figure 4.2: Wooden Cart Side-view

### 4.1.2 Ball Ramp

The ball ramp is made from a flexible and lightweight material, designed to guide collected balls smoothly into the cart for storage.

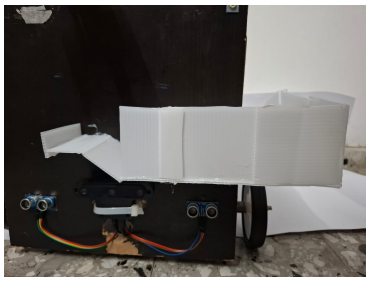

Figure 4.3: Ramp Front-view

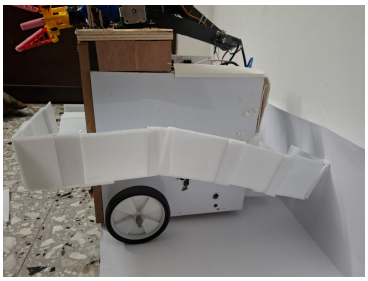

Figure 4.4: Ramp Side-view

Figure 4.5: Ramp Isometric-view

### 4.1.3 Robotic Arm

The DOF robotic arm mounted on a wooden platform has a maximum reach of 60cm and is purpose built to accurately collect and manoeuvre balls. It has two 3D printed joints which move smoothly to grasp and transfer balls easily into the cart.

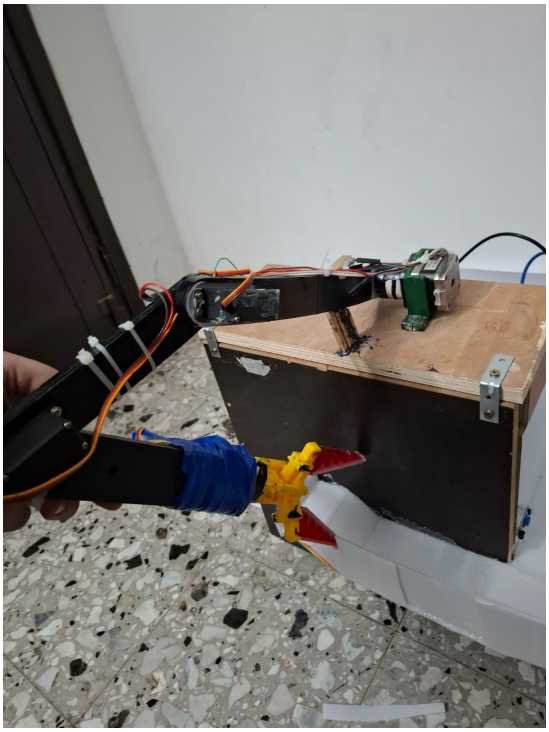

Figure 4.6: Default Arm Position

Figure 4.7: Fully Extended Arm

Figure 4.8: Arm joint

### 4.1.4 Adaptive Gripper

The adaptive gripper is a 3D printed part designed to grasp balls effectively. It has a three finger design, which has the fingers made of flexible TPU material to adapt to the size and shape of the ball for a better grip. The main body of the gripper is printed with durable PLA filament which gives a solid frame to support the fingers in the course of use. The TPU finger of each finger is covered with a layer of silicone rubber to increase the friction on the surface of the ball to prevent slippage. The flexible finger and rigid body combination of the gripper makes it suitable to work with different ball textures and weights, giving it a high efficiency when use in different sports environments. The system's lightweight and long lasting structure makes it suitable for easy use without causing damage to the balls, thus being an important part of the autonomous ball collection system.(Cults3D, 2022)

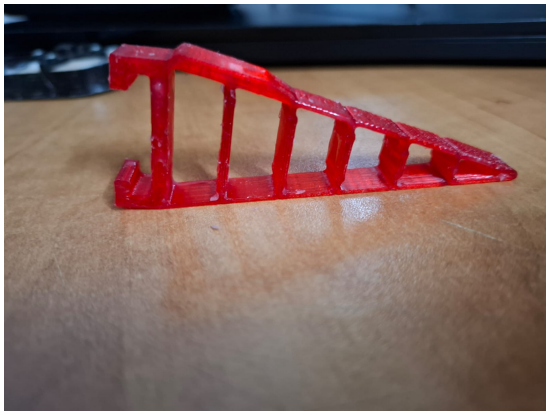

Figure 4.9: TPU Finger

Figure 4.10: TPU Finger - Silicon Coated

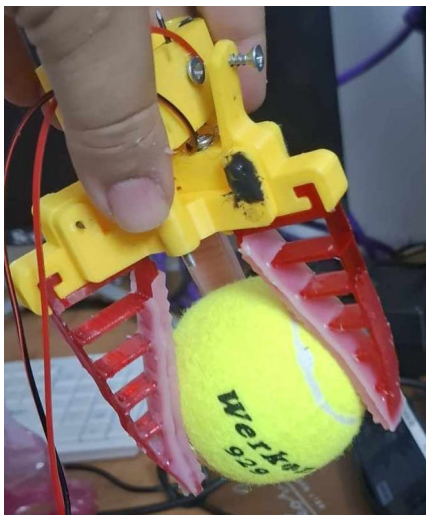

Figure 4.11: Tennis Ball Gripped by the Fingers

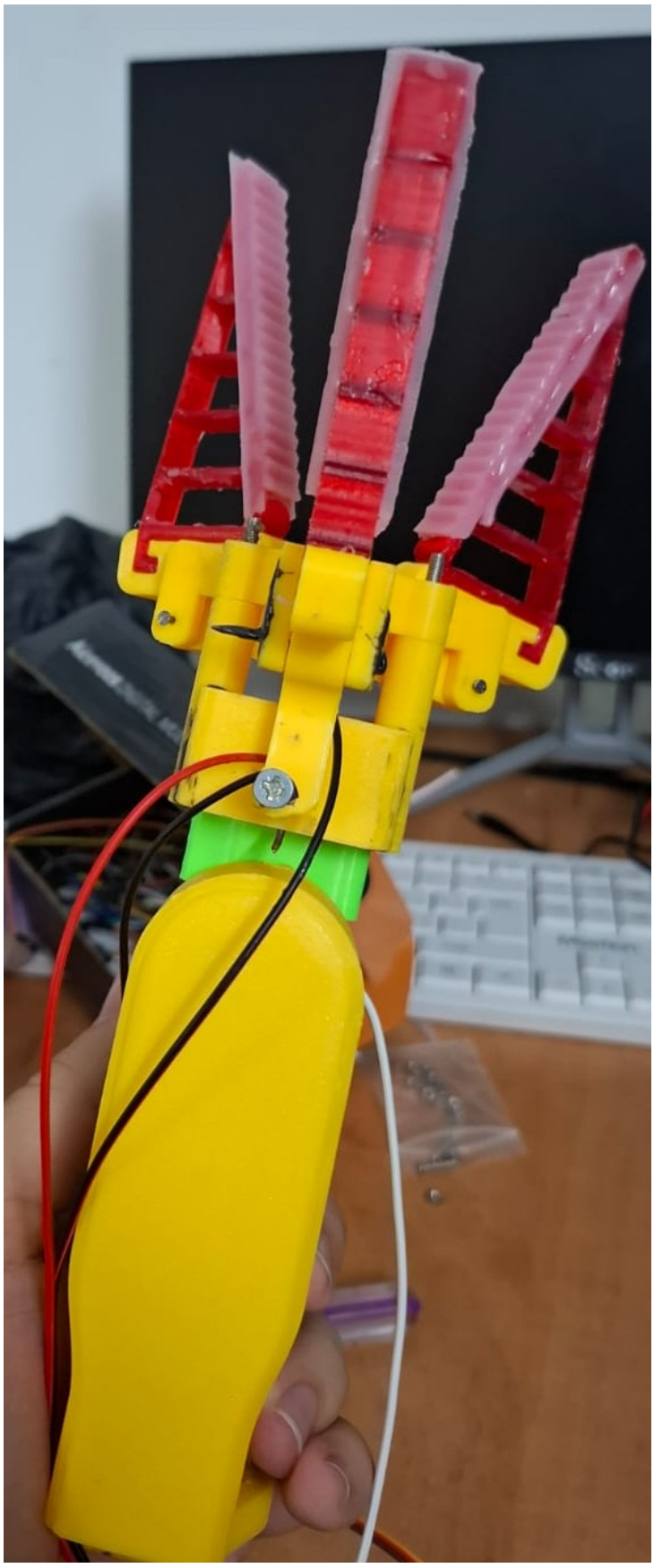

Figure 4.12: Full Adaptive Gripper

## 4.2 Computing Devices and Overall Components

### 4.2.1 Raspberry Pi 3 Model B

The Raspberry Pi 3 Model B carries out the key functions of the project including data collection, communication and control. It is the best candidate for the job of an autonomous robotics application because it is affordable, small and comes with built in wireless modules. The Raspberry Pi 3 has a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor and 1 GB of RAM, making it capable of handling system processes though it has some limitations on the side of computational capability.Image processing in real time for ball detection is done by the Oak-D camera which runs the ODM on its own processing block. This way the Raspberry Pi 3 is not engaged in heavy computational work and instead, it is in charge of sending commands to other parts of the system and effecting control measures. The raw data from the Oak-D camera is processed and formulates the appropriate actions to be taken for instance, to make the robotic arm open or close the claw and or, define the path that the system should follow.The Wi-Fi and Bluetooth modules in the Raspberry Pi 3 also allow for the remote control and monitoring of the system. A system status, number of balls picked and control buttons are also presented in a web based interface. This is very useful as it enables the user to change system parameters without having to physically interface with the system.The major limitation of the Raspberry Pi 3 is the limited processing power and the small amount of memory which can be a problem when developing features such as real time video streaming. To work around these deficiencies the project uses light weight algorithms and minimal data management to achieve good performance with the available resources.(Raspberry Pi Foundation, n.d.)

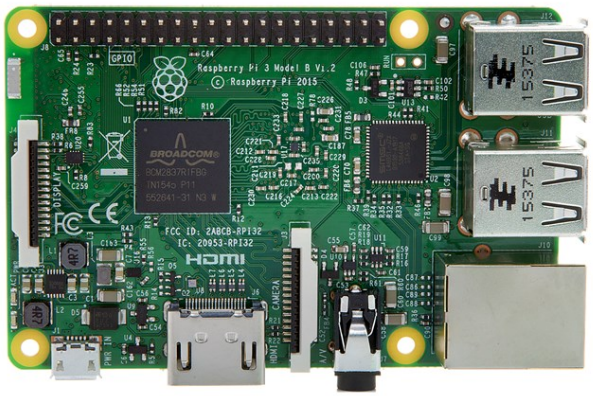


Figure 4.13: Raspberry Pi 3 Model B

### 4.2.2 Luxonis OAK-D camera

The Oak-D camera is the vision system of the project which does real time ball detection, localization and size estimation. The Oak-D camera that comes with an onboard Myriad X VPU (Vision Processing Unit) processes the visual data and identifies and tracks balls without burdening the Raspberry Pi 3. This hardware acceleration is very efficient and accurate and is therefore an important part of the project's autonomous operation.The Oak-D camera with its high resolution RGB imaging and depth sensing is able to give the project the information it needs to accurately determine the size of balls in various environments. The depth information improves the accuracy of the ball localization, so the system is able to determine the position and dimensions of the ball with respect to the robot. This is particularly important for distinguishing between different types of balls and for ensuring that they are properly picked up by the gripping mechanism. In the project, the Oak-D camera continuously acquires field images and uses the ODM to identify the balls from other features in the environment. The processed data of ball size and position is then sent to the Raspberry Pi 3, which controls the robot's movements on the basis of the characteristics of the detected ball.The Oak-D camera is easily integrated with the hardware of the project through a USB port and this ensures that there is high speed data transfer with minimal delay. This effective communication pipeline guarantees that ball detection and size estimation outcomes are delivered in a timely manner to enable the robot to make appropriate responses to changes in its environment.(Luxonis, n.d)

Figure 4.14: Luxonis OAK-D camera

### 4.2.3 Arduino Uno

The Arduino Uno, based on the ATmega328P, is a popular microcontroller for small projects due to its simplicity and versatility. It has 14 digital and 6 analog pins, supporting UART, I2C, and SPI communication for real-time control.(Arduino, n.d.-b)

In this project, the Arduino Uno manages the robotic arm's grippin and movement, ensuring smooth ball pickup, and sends sensor data to the Raspberry Pi 3 for overall system control.

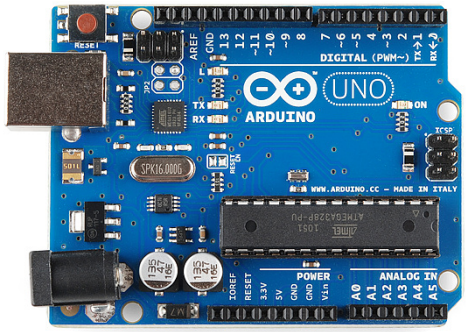


Figure 4.15: Arduino Uno

### 4.2.4 Arduino Mega

The Arduino Mega, powered by the ATmega2560, offers more processing power and I/O pins than the Arduino Uno, making it ideal for complex projects. With 54 digital and 16 analog inputs, it supports I2C and SPI communication, providing accurate control and real-time data processing.Arduino, n.d.-a

In this project, the Arduino Mega controls movement by processing ultrasonic sensor data for smooth navigation and precise turns. It communicates with the Raspberry Pi 3, sending real-time sensor data and receiving high-level commands to optimize path planning and efficiency.

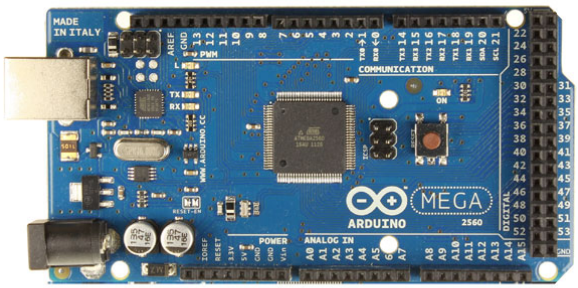


Figure 4.16: Arduino Mega

### 4.2.5 Micro Metal Gear Motor 100RPM 12V

The Micro Metal Gear Motor 100RPM 12V is a compact, high-torque motor with a gearbox that provides precise control and durability. It is used to operate the gripper fingers, to securely collect balls.

Figure 4.17: Micro Metal Gear Motor 100RPM 12V

### 4.2.6 Servo Motor MG946R

The MG946R Servo Motor is a high torque digital servo used in robotics for precise motion control. It has metal gears for durability and has a torque of up to 12 kg/cm, and therefore, it is well suited for use in applications that need strong and accurate movement. In the project, two MG946R servos are used, one on each joint of the robotic arm so that the arm can move smoothly and precisely in order to pick up the balls efficiently. The motors receive control signals from the Arduino Uno, which enable reliable and responsive arm movements.


Figure 4.18: Servo Motor MG946R

### 4.2.7 Nema 23 Stepper Motor

The NEMA 23 stepper motor is widely used in automation and robotics because it can move in step without feedback and is very precise. It has high holding torque, and is thus appropriate for applications which need good control and repeatability of position. It has a standard frame size of 56.4mm and a step angle of 1.8 degrees, which enables fine resolution and smooth movement.

In the project, 2 NEMA 23 stepper motors are used to assist the robot to move around the field in a systematic and efficient manner with high precision in positioning and low rate of movement. Since the motors have high torque density, they enable the robot to move on different surfaces and carry the required payloads steadily. The motors are controlled by stepper drivers that control the current and the step sequence to achieve the best performance and reduce vibrations while moving.

Figure 4.19: Nema 23 Stepper Motor

### 4.2.8 Microstep 3.5A Driver

The 3.5A microstepping driver provides precise motor control by dividing steps into smaller increments for smoother motion. It supports up to 3.5A of current, offers adjustable microstepping settings, and includes protections against overheating, overcurrent, and short circuits, ensuring reliable operation.

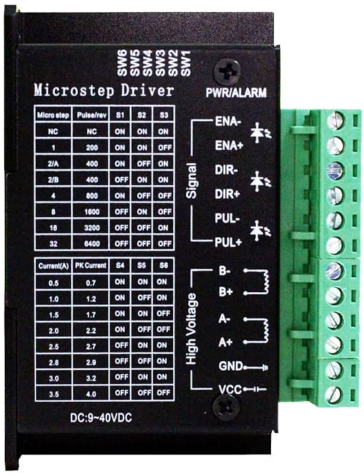


Figure 4.20: Microstep 3.5A Driver

### 4.2.9 L293D Motor Driver

The L293D motor driver is a dual H-bridge IC that controls the speed and direction of DC motors using PWM signals to control the bidirectional movement of motors. It has built in flyback diodes to protect the IC from back emf or voltage spikes. In the project, L293D is used to control the Micro Metal Gear Motor 100RPM 12V to control the opening and closing of the gripper mechanism for efficient ball handling. Its compact design and reliable performance make it suitable for smooth motor operation.

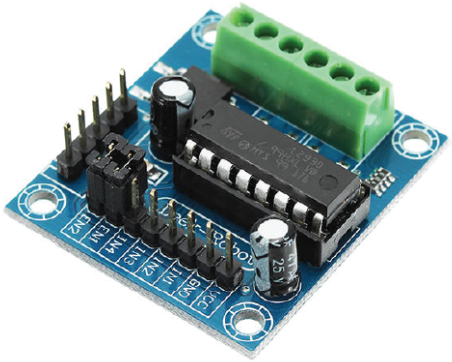

Figure 4.21: L293D Motor Driver

### 4.2.10 ACS712 5A Current Sensor Module

The ACS712 5A current sensor module is a compact and efficient device used for measuring current in both AC and DC circuits. It provides accurate, real-time current readings with a sensitivity of 185mV/A, making it ideal for monitoring and protection applications. The module features built-in isolation to ensure safe operation and minimize interference.

In the project, the ACS712 is used to monitor the current drawn by the Micro Metal Gear Motor 100RPM 12V, preventing it from stalling by detecting overload conditions. When excessive current is detected, the system can take corrective actions to protect the motor from damage and ensure smooth operation of the gripper mechanism.

Figure 4.22: ACS712 5A Current Sensor Module

### 4.2.11 HC-SR04 Ultrasonic Sensor

The HC-SR04 sends out ultrasonic waves and calculates the time taken for the echo to return from an object to work. The sensor has a measurement range of 2 cm to 400 cm with a precision of about 3 mm. As a reliable solution for non-contact distance sensing in robotics applications, the sensor consists of a transmitter and a receiver.

For the project, four HC-SR04 sensors are used; two are at the front and two are on the sides to help assist the robot in avoiding obstacles. These sensors give real time distance data, which allows the system to recognize and avoid potential obstacles in operation.


Figure 4.23: HC-SR04 Ultrasonic Sensor

### 4.2.12 IR Obstacle Avoidance Module

The IR obstacle avoidance module is an infrared sensor that works by emitting IR signals and detecting how much of them is reflected back by nearby objects. It uses reflected infrared light from objects within its range to detect them, and is thus well suited for close range object detection tasks. The module is compact, power efficient and offers a contactless method of detecting objects.

The IR obstacle avoidance module is applied to detect whether or not a ball has entered the chamber. When the ball is detected, the module sends a signal to the Arduino Uno, which indicates that the ball has been properly collected. This permits the system to continue with the next stages of the operation.

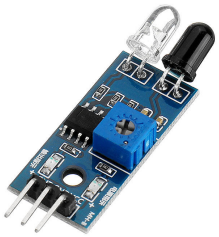

Figure 4.24: IR Obstacle Avoidance Module

### 4.2.13 12V 7Ah Lead Acid Battery

The 12V 7Ah lead acid battery is a good power source that is known for its durability and reliability in powering and supplying power. It has high energy density and good service life and is particularly well suited for applications in which a high output voltage and current with minimal variation is required. Lead acid batteries are also relatively inexpensive and can manage reasonable discharge properly.


Figure 4.25: 12V 7Ah Lead Acid Battery

### 4.2.14 Battery Indicator

The battery indicator module is used in the project to monitor and display the battery levels, providing real-time status updates to ensure optimal power management.

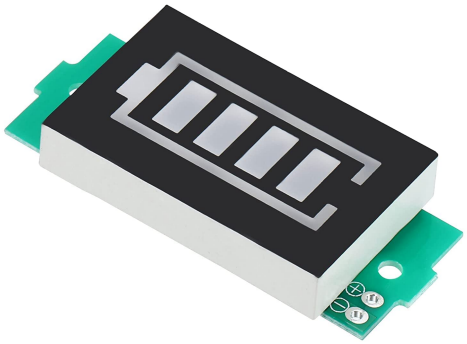

Figure 4.26: Battery Indicator

### 4.2.15 12 Volts Voltage Regulator Module

The step-down voltage regulator module based on LM2596 chip is used to transform 12V to 5V efficiently. The LM2596 is a DC-DC buck converter with high efficiency and stable output voltage, and it is appropriate for powering sensitive electronics. In the project, the module is connected to the servo motors and other electronic modules to supply 5V, thus guaranteeing their proper functioning and protecting them from voltage changes.

Figure 4.27: 12 Volts Voltage Regulator Module

## 4.3 Software Components

### 4.3.1 Google Cloud Pub/Sub

Google Cloud Pub/Sub is a messaging service that enables reliable asynchronous communication for distributed systems, like a scalable messaging service for real time message dispatch and data exchange between devices.(Google Cloud, n.d.)

In the project, the Google Cloud Pub/Sub is used to send commands to the robot so as to enable remote control of the robot from anywhere without having the systems being physically close. The Raspberry Pi subscribes to a specific topic named "comms" and in every given time, it is receiving new commands and executes them. At the same time, the system uses Pub/Sub to pull status information from the Raspberry Pi, which gives real time information about the state and performance of the robot. This cloud based communication model is very efficient, secure and responsive when interacting with the robot.

### 4.3.2 Google Cloud Run

Google Cloud Run is a fully managed serverless platform to securely and scalably run containerized applications. It provides high availability, automatic scaling and seamless integration with other Google Cloud services.

In the project, both the backend and frontend services are hosted by Google Cloud Run. The backend is involved in dispatching commands to the Pub/Sub messaging system and in doing so, ensures that the communication with the robot is efficient and reliable. The frontend, which is also hosted on Cloud Run, offers a user-friendly interface for the user to monitor and control the robot remotely. This setup allows users to communicate with the robot at will, all the while taking advantage of the scalability and security of Google Cloud infrastructure.

### 4.3.3 ODM

The ODM is based on YOLOv8-small, which is a high performance deep learning model that has been optimized for real time object detection. Because YOLOv8-small is a good balance of speed and accuracy, it is well suited for applications that require fast and reliable object recognition with a limited amount of computational power.(Ultralytics, 2023)

YOLOv8-small was trained on a dataset of over 30,000 images and was trained on 25 categories of balls, including tennis, football, ping pong, and all the different pool balls. To make the system more generic, there are five pseudo color classes (red, green, yellow, blue and user defined color) that will override the color of the detected ball based on color.

It involves training the model on large amount of data that comprises of rotation, scaling and lighting adjustment to enhance the accuracy of the system for various conditions. The model is able to detect and classify the balls with a high confidence level and gives the bounding box coordinates and the class predictions in real time.

The inference is done on the Oak-D camera, which processes images and runs the YOLOv8 model on board. This way the system is able to perform the detection and classification of the object and then pass it to the Raspberry Pi for further processing, thus reducing the computational load and ensuring that the autonomous system works properly. The Oak-D camera first identifies the type of ball and then applies the pseudo color class if necessary, which enhances the sorting and handling of the balls.

Thus, with the help of YOLOv8-small, the project is able to achieve the real time detection and classification of the balls and hence the robot is able to sort and manage different types of balls with minimal human intervention.

A breakdown of the model metrics and the training process is listed in the following pages.

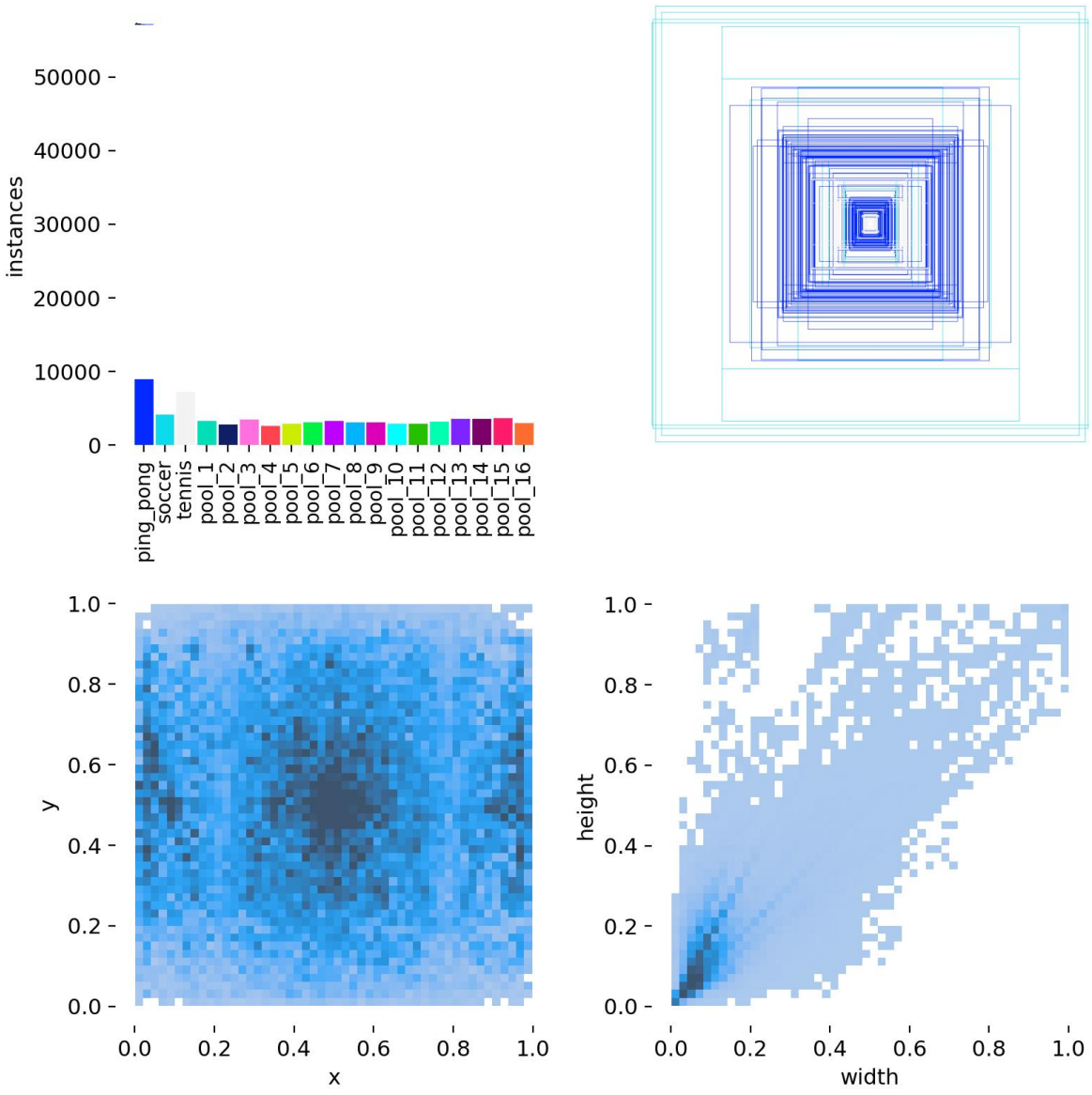


Figure 4.28: Labels Distribution

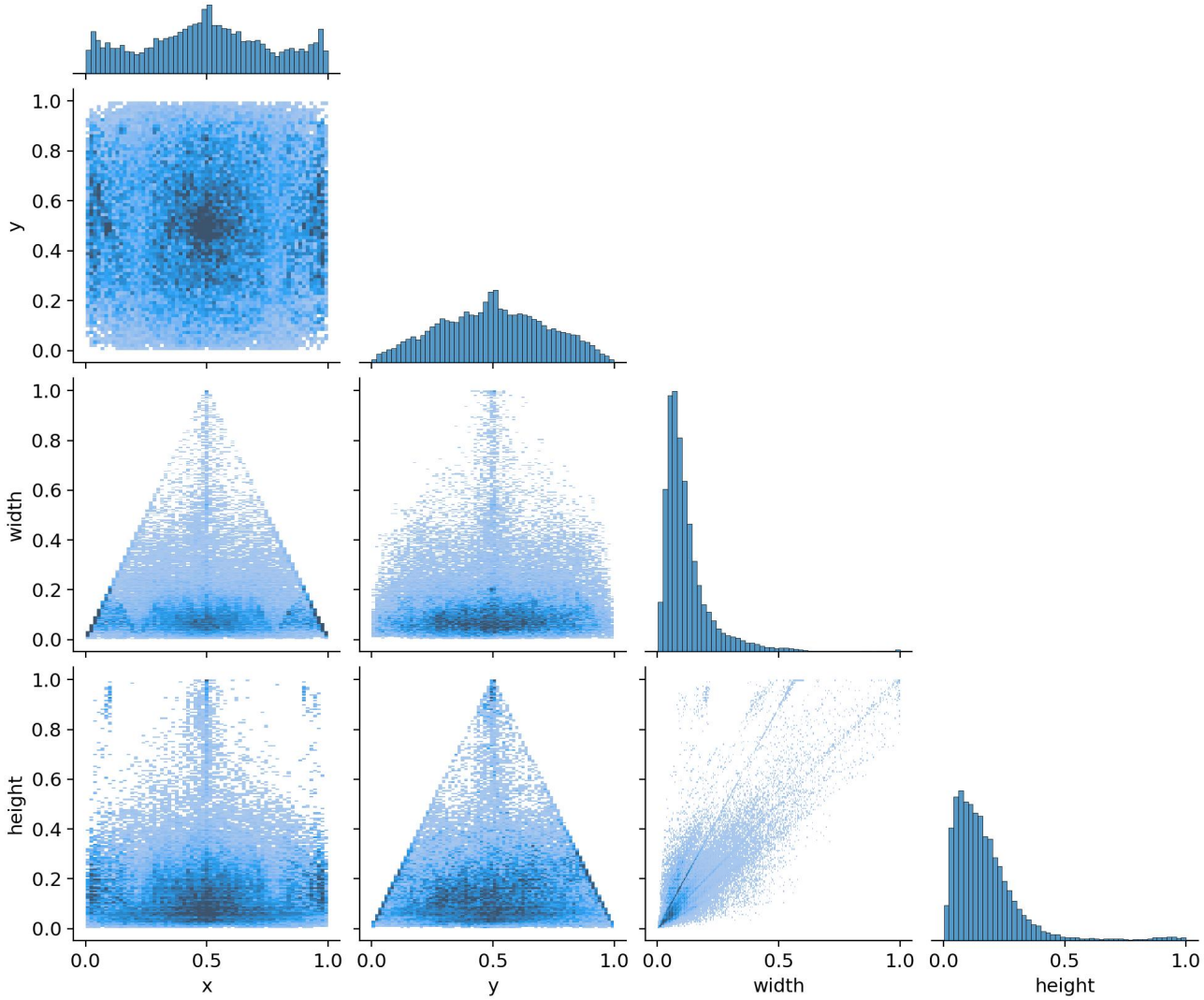


Figure 4.29: Labels Correlogram

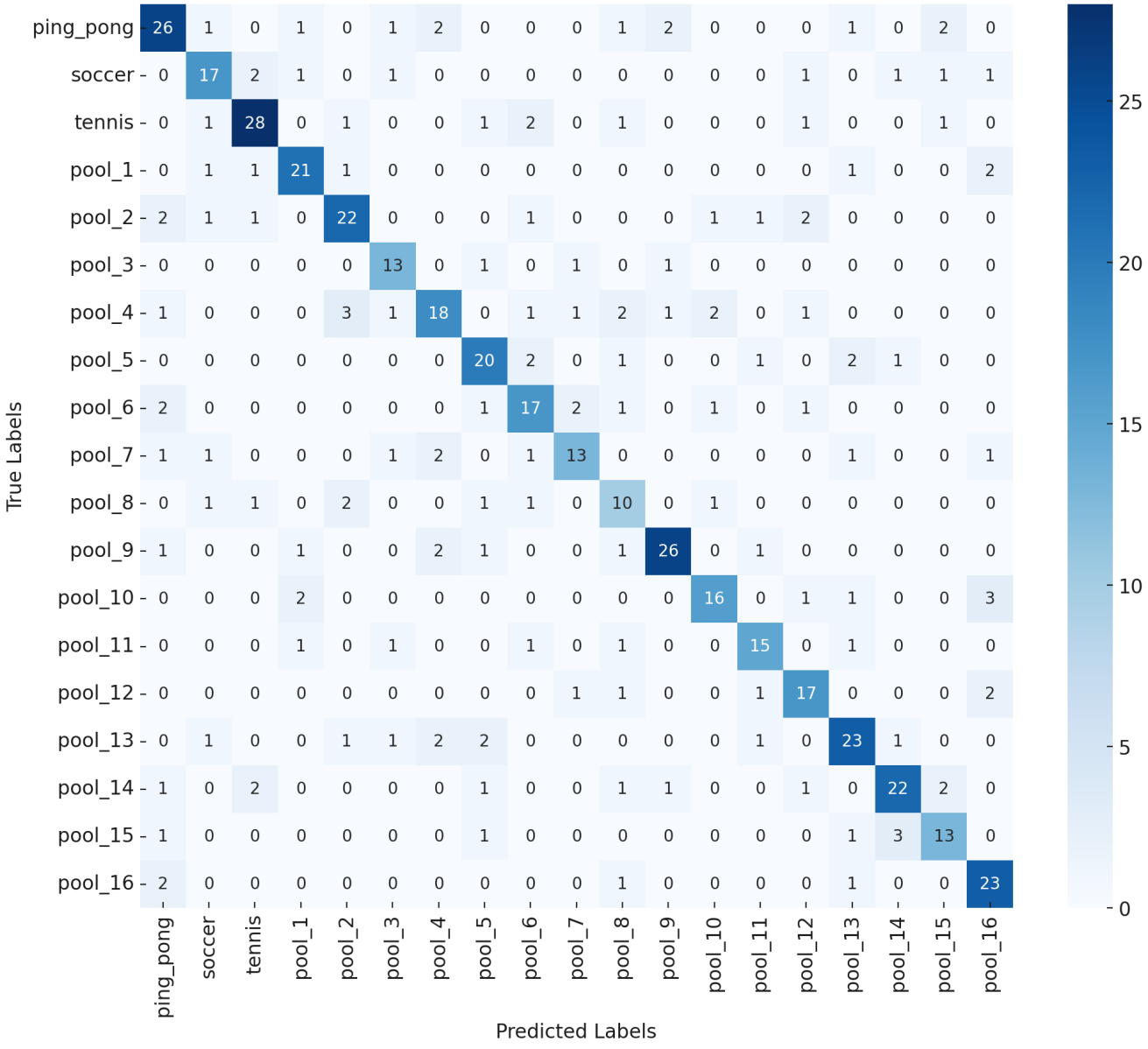


Figure 4.30: Confusion Matrix

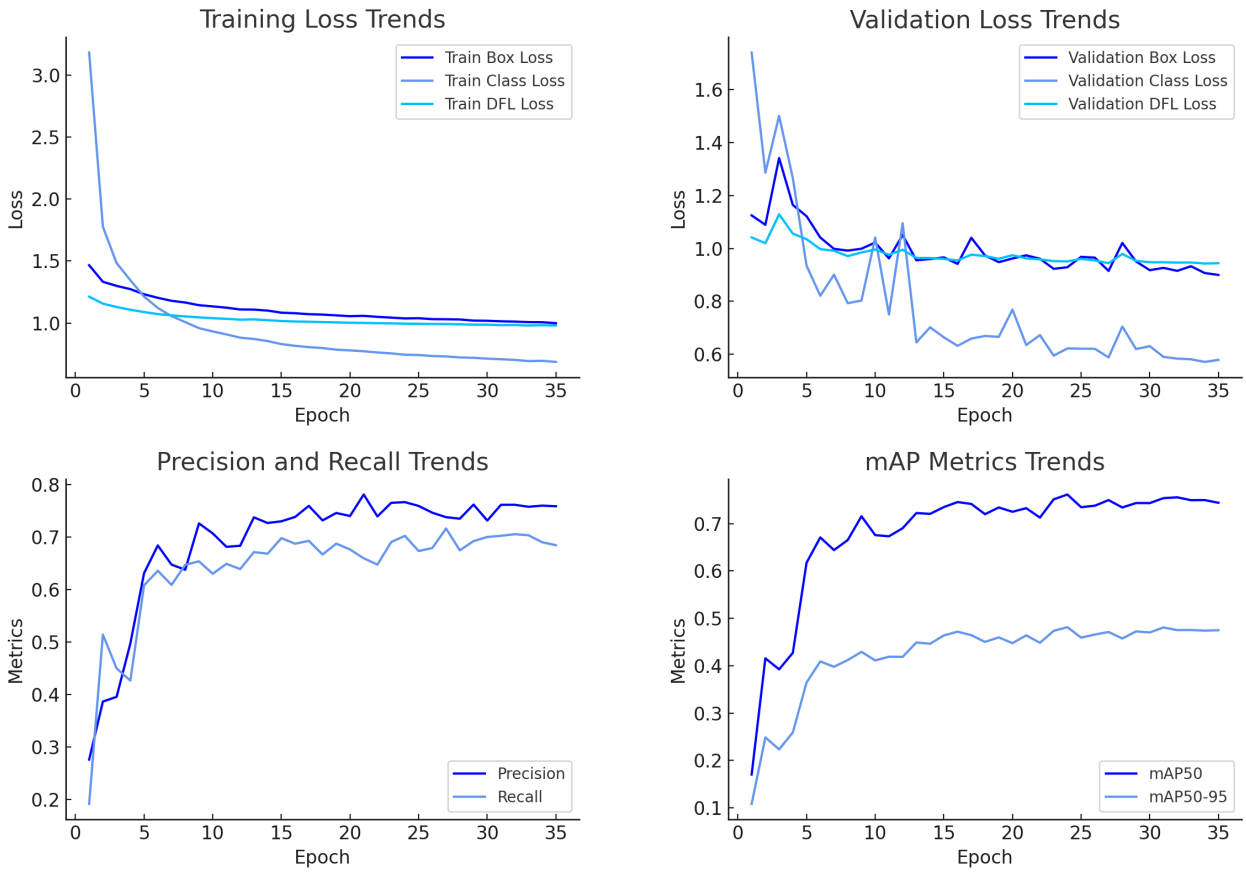


Figure 4.31: Results

## 4.4 How The System Works

### 4.4.1 Connections & Integrations

To ensure stability and efficiency of operation, the project uses several types of electrical connections with three separate ground references. The power bank ground is linked to the Raspberry Pi and Oak-D camera and has to be connected manually to their respective power supply because they are power hungry. The Raspberry Pi 3 Model B, with a quad-core processor and its peripherals connected to it, uses about 2.5 A at 5 V; the Oak-D camera, with an AI processor and depth sensor, needs up to 3 A at 5 V to work at its best. The Arduino Mega ground reference is for the LED strip, ultrasonic sensors and stepper motor drivers to maintain good signal integrity and electrical stability. The stepper motors and the geared DC motor are directly powered by the 12 V lead acid battery, which provides the required voltage and current for their functioning. On the other hand, the Arduino Uno ground reference is employed for the servos, step down voltage converter and other connected peripherals. The servos are controlled by a step down voltage converter that regulates the battery output of 12 V to 5 V to avoid damage and to operate the components efficiently. All components are pre-wired and the only manual work is to connect the Raspberry Pi and the Oak-D camera to the power bank, then activate the system through the main power switch which distributes power to all the modules smoothly.

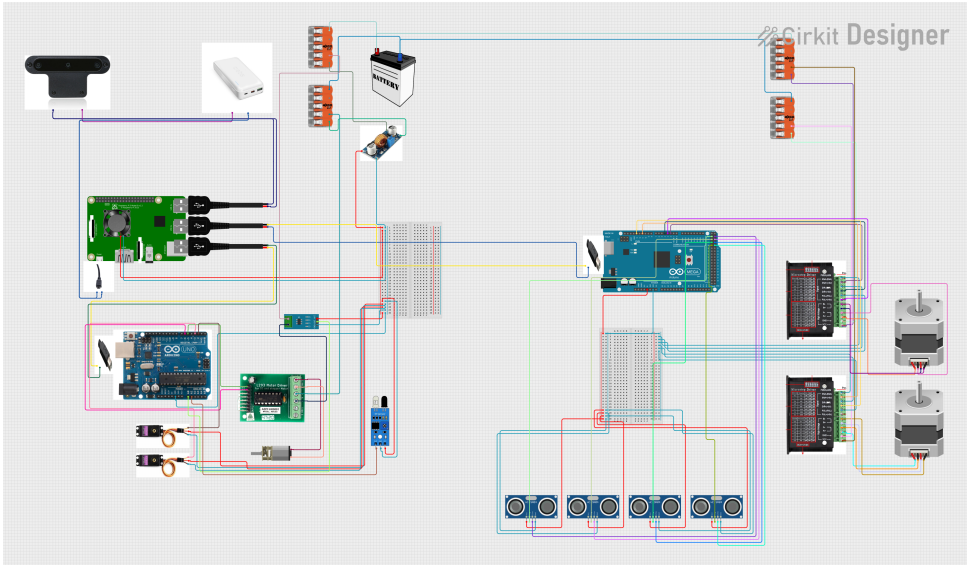


Figure 4.32: System Connections

### 4.4.2 Initializing The System

To initialize the system, first, manually connect the Raspberry Pi and Oak-D camera to their power source, as they require dedicated power. Once connected, pressing the main power button activates the entire system. The starting and configuration process is carried out through the web application, which can be accessed remotely, allowing the user to operate the system without being physically close to it. Through the web app, the user must select the collection mode—either specific ball collection or collect all balls mode. If the specific collection mode is chosen, the user can specify the desired ball types to be collected. Additionally, the movement pattern must be selected, allowing the system to navigate using pre-defined strategies. Once all settings are configured, pressing the start button in the web app initiates the autonomous ball collection process. A stop button is also available in the web app to immediately halt system operations, stopping both movement and ball detection when needed.

### 4.4.3 Ball Detection and Collection Process

When the robot is commanded to start, it first begins moving according to the selected movement mode—such as zig-zag or random—while actively scanning the environment for balls using its DepthAI camera. As soon as it detects a ball that meets the confidence threshold and matches the chosen classification (e.g., specific color or type), it transitions to navigation mode and turns or drives forward to close in on the target. Once the robot judges the ball to be within a safe gripping range, it switches to stopping mode, halts its motors, and precisely positions the gripper. Finally, it initiates a gripping routine: lowering the arm, grasping the ball, and lifting it securely and dropping it into the ramp then returning to the detection phase to look for more targets.

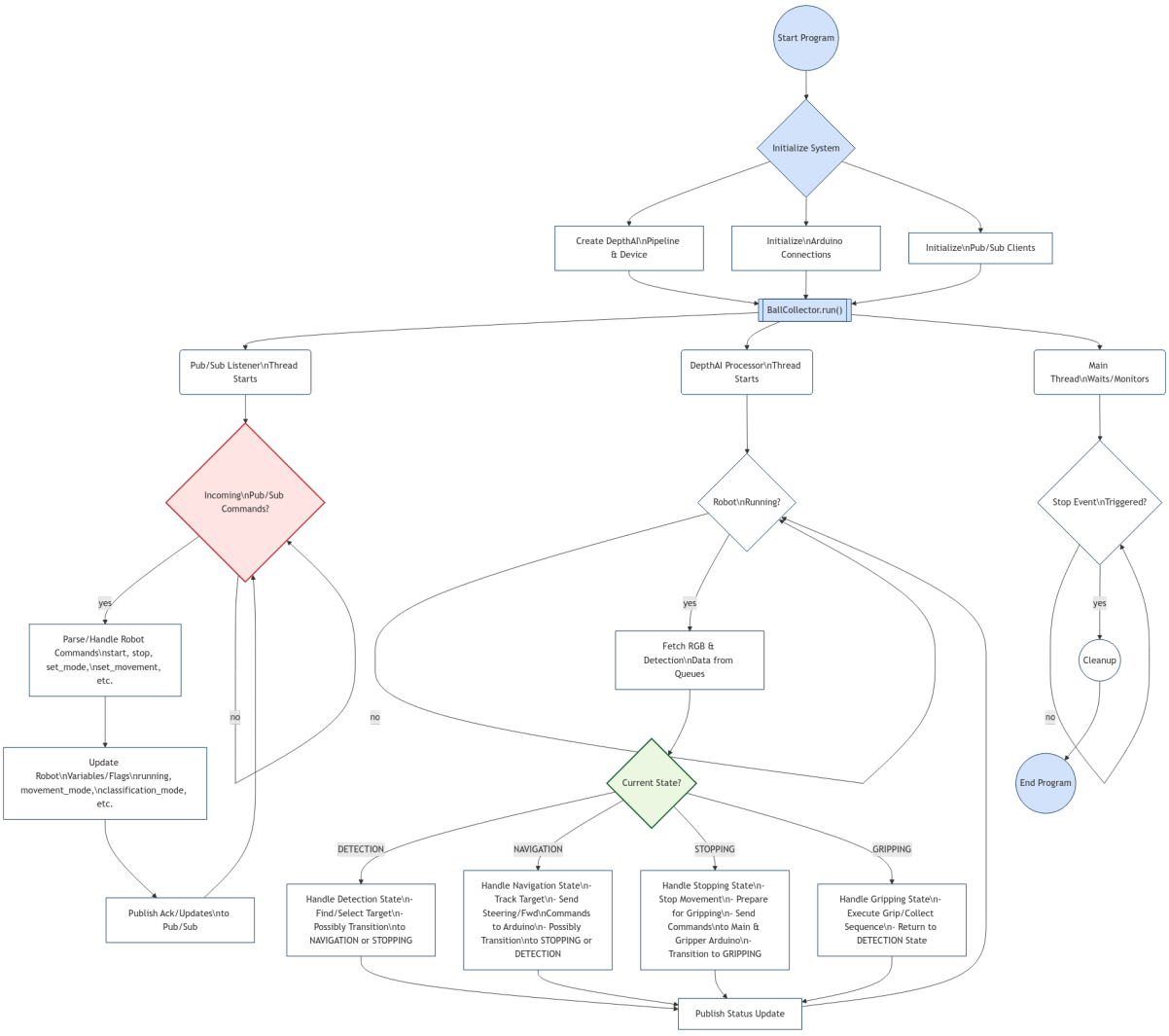


Figure 4.33: System Flowchart

# 5 Results & Discussion

## 5.1 Results

The project demonstrates the successful implementation of an autonomous ball collection system with real-time detection and classification. The system efficiently identified and collected various ball types, while the integration of the Oak-D camera and YOLOv8-small model ensured smooth and reliable operation. The robot effectively navigated the environment, detected obstacles, and provided real-time status updates, validating the project's objectives.

## 5.2 Discussion

The projects provides modern and efficient method of using autonomous ball collection systems as against manual methods. It gives a reliable solution which can be useful to increase the efficiency and convenience of sports training and recreational activities.

# 6 Conclusions & Future Work

## 6.1 Conclusions

We believe this project enhances efficiency and convenience with advanced robotics and AI. It effectively collects various balls, offering value in sports and recreation. Despite challenges like power efficiency, it provides a strong foundation for future improvements.

## 6.2 Future Work

- Improve environmental adaptability to enhance performance in diverse conditions.
- Optimize power efficiency to extend battery life and operational time.
- Enhance obstacle detection and avoidance capabilities for better navigation.
- Explore the integration of alternative energy sources to support longer usage.
- Develop a more compact and lightweight design for increased portability.
- Expand the system's application to other areas beyond sports and recreation.
- Conduct further testing in varied real-world environments to identify additional improvements.

# A References


Arduino. (n.d.-a). Arduino mega 2560 rev3 technical specifications [Retrieved January 27, 2025, from https://docs.arduino.cc/hardware/mega-2560].

Arduino. (n.d.-b). Arduino uno rev3 technical specifications [Retrieved January 27, 2025, from https://docs.arduino.cc/hardware/uno-rev3].

Cults3D. (2022). Improved adaptive gripper v2.0 - complete kit [Retrieved January 27, 2025, from https://cults3d.com/en/3d-model/gadget/improved-adaptive-gripper-v2-0-complete-kit].

Google Cloud. (n.d.). Pub/sub messaging service [Retrieved January 26, 2025, from https://cloud.google.com/pubsub].

Luxonis. (n.d). Oak-d camera documentation [Retrieved January 24, 2025, from https://docs.luxonis.com/hardware/products/OAK-D].

Raspberry Pi Foundation. (n.d.). Raspberry pi 3 model b specifications [Retrieved January 22, 2025, from https://www.raspberrypi.com/products/raspberry-pi-3-model-b/].

Ultralytics. (2023). Yolov8: Real-time object detection [Retrieved January 25, 2025, from https://docs.ultralytics.com/models/yolov8/].

# B Arduino Mega Code for Robot Movement and Obstacle Avoidance

```
    // ---------------------------------------------------------------------
// Pin Configuration
// ---------------------------------------------------------------------

// Motor 1 (Back Left)
#define ENA_PIN1 2   // Enable Pin for Motor 1
#define STEP_PIN1 3  // Step Pin for Motor 1
#define DIR_PIN1 4   // Direction Pin for Motor 1

// Motor 2 (Back Right)
#define ENA_PIN2 11  // Enable Pin for Motor 2
#define STEP_PIN2 12 // Step Pin for Motor 2
#define DIR_PIN2 13  // Direction Pin for Motor 2

// Ultrasonic Sensors
#define TRIG_LEFT 22
#define ECHO_LEFT 23
#define TRIG_RIGHT 24
#define ECHO_RIGHT 25
#define TRIG_FRONT_LEFT 26
#define ECHO_FRONT_LEFT 27
#define TRIG_FRONT_RIGHT 28
#define ECHO_FRONT_RIGHT 29

// Threshold Distances (in cm)
#define FRONT_THRESHOLD 50
#define SIDE_THRESHOLD 50

// Obstacle Avoidance Flags
volatile bool manualOverride = false;

// ---------------------------------------------------------------------
// Setup Function
// ---------------------------------------------------------------------
void setup() {
  // Initialize Serial Communication
  Serial.begin(9600);
  Serial.println("System Ready. Awaiting Commands...");

  // Initialize Motor Pins as Outputs
  pinMode(ENA_PIN1, OUTPUT);
  pinMode(STEP_PIN1, OUTPUT);
  pinMode(DIR_PIN1, OUTPUT);

  pinMode(ENA_PIN2, OUTPUT);
  pinMode(STEP_PIN2, OUTPUT);
  pinMode(DIR_PIN2, OUTPUT);

  // Enable Motors (Active LOW)
  digitalWrite(ENA_PIN1, LOW);
  digitalWrite(ENA_PIN2, LOW);

  // Initialize Ultrasonic Sensor Pins
```

```
  pinMode(TRIG_LEFT, OUTPUT);
  pinMode(ECHO_LEFT, INPUT);

  pinMode(TRIG_RIGHT, OUTPUT);
  pinMode(ECHO_RIGHT, INPUT);

  pinMode(TRIG_FRONT_LEFT, OUTPUT);
  pinMode(ECHO_FRONT_LEFT, INPUT);

  pinMode(TRIG_FRONT_RIGHT, OUTPUT);
  pinMode(ECHO_FRONT_RIGHT, INPUT);
}

// ---------------------------------------------------------------------------
// Loop Function
// ---------------------------------------------------------------------------
void loop() {
  // Check for Serial Commands
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();
    if (command.length() > 0) {
      manualOverride = true; // Activate Manual Override
      controlMotors(command);
    }
  }

}

// ---------------------------------------------------------------------------
// Obstacle Detection and Autonomous Navigation
// ---------------------------------------------------------------------------
void autonomousNavigation() {
  // Measure Distances from All Sensors
  long frontLeftDist = getDistance(TRIG_FRONT_LEFT, ECHO_FRONT_LEFT);
  long frontRightDist = getDistance(TRIG_FRONT_RIGHT, ECHO_FRONT_RIGHT);
  long leftDist = getDistance(TRIG_LEFT, ECHO_LEFT);
  long rightDist = getDistance(TRIG_RIGHT, ECHO_RIGHT);

  // Debugging: Print Sensor Readings
  Serial.print("Front Left: "); Serial.print(frontLeftDist);
  Serial.print(" cm | Front Right: "); Serial.print(frontRightDist);
  Serial.print(" cm | Left: "); Serial.print(leftDist);
  Serial.print(" cm | Right: "); Serial.println(rightDist);

  // Decision Making Based on Sensor Readings
  if (frontLeftDist > FRONT_THRESHOLD && frontRightDist > FRONT_THRESHOLD) {
    // Path is clear, move forward
    moveForward(600);
  }
  else {
    // Obstacle detected ahead
    if (leftDist > rightDist && leftDist > SIDE_THRESHOLD) {
      // Prefer turning left if more space on the left
      rotateLeft(300);
    }
    else if (rightDist > SIDE_THRESHOLD) {
      // Turn right if space on the right
```

```
      rotateRight(300);
    }
    else {
      // If no space on sides, rotate 180 degrees
      rotateLeft(600);
    }
  }
}

// ---------------------------------------------------------------------
// Ultrasonic Distance Measurement Function
// ---------------------------------------------------------------------
long getDistance(int trigPin, int echoPin) {
  // Clear Trigger Pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Send 10us Pulse to Trigger
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read Echo Pin
  long duration = pulseIn(echoPin, HIGH, 30000); // Timeout after ~5ms

  // Calculate Distance in cm
  long distance = (duration * 0.034) / 2;

  // Handle out-of-range measurements
  if (duration == 0) {
    distance = 400; // Assume max distance if no echo received
  }

  return distance;
}

// ---------------------------------------------------------------------
// Autonomous Movement Functions
// ---------------------------------------------------------------------
void moveForward(int steps) {
  Serial.println("Autonomous: Moving Forward");

  // Set Directions for Forward Movement
  digitalWrite(DIR_PIN1, LOW);  // Motor 1 Forward
  digitalWrite(DIR_PIN2, HIGH); // Motor 2 Forward

  // Move Specified Steps
  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(2500);
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(2500);

    // Check for Stop Command During Movement
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
```

```
      cmd.trim();
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}

void moveBackward(int steps) {
  Serial.println("Autonomous: Moving Backward");

  // Set Directions for Backward Movement
  digitalWrite(DIR_PIN1, HIGH); // Motor 1 Backward
  digitalWrite(DIR_PIN2, LOW);  // Motor 2 Backward

  // Move Specified Steps
  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(2500);
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(2500);

    // Check for Stop Command During Movement
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}

void rotateLeft(int steps) {
  Serial.println("Autonomous: Rotating Left");

  // Set Directions for Left Rotation (In-Place)
  digitalWrite(DIR_PIN1, LOW);  // Motor 1 Backward
  digitalWrite(DIR_PIN2, LOW);  // Motor 2 Forward

  // Rotate Specified Steps
  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(2500);
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(2500);

    // Check for Stop Command During Rotation
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
```

```
      cmd.trim();
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}

void rotateRight(int steps) {
  Serial.println("Autonomous: Rotating Right");

  // Set Directions for Right Rotation (In-Place)
  digitalWrite(DIR_PIN1, HIGH); // Motor 1 Forward
  digitalWrite(DIR_PIN2, HIGH); // Motor 2 Backward

  // Rotate Specified Steps
  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(2500);
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(2500);

    // Check for Stop Command During Rotation
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}

// ------------------------------------------------------------------------
// Manual Control Functions and Patterns
// ------------------------------------------------------------------------
volatile bool inPattern = false;

void controlMotors(String command) {
  Serial.print("Manual Command Received: ");
  Serial.println(command);

  // Reset pattern state
  inPattern = false;

  // Ensure Motors are Enabled
  digitalWrite(ENA_PIN1, LOW);
  digitalWrite(ENA_PIN2, LOW);

  // Argument-Based Commands (e.g., "fa 3200", "ba 2000")
  if (command.startsWith("fa ") || command.startsWith("ba ") ||
      command.startsWith("la ") || command.startsWith("ra ")) {
```

```
  int spaceIndex = command.indexOf(' ');
  if (spaceIndex > 0) {
    String prefix = command.substring(0, spaceIndex);
    String stepsStr = command.substring(spaceIndex + 1);
    stepsStr.trim();
    int stepsVal = stepsStr.toInt();

    if (prefix.equalsIgnoreCase("fa")) {
      moveForward(stepsVal);
    }
    else if (prefix.equalsIgnoreCase("ba")) {
      moveBackward(stepsVal);
    }
    else if (prefix.equalsIgnoreCase("la")) {
      rotateLeft(stepsVal);
    }
    else if (prefix.equalsIgnoreCase("ra")) {
      rotateRight(stepsVal);
    }
    else {
      Serial.println("Unknown prefix for argument-based command!");
    }
    return;
  }
}

// Single-Letter or Existing Commands
if (command.equalsIgnoreCase("f")) {
  moveForward(3200);
}
else if (command.equalsIgnoreCase("b")) {
  moveBackward(3200);
}
else if (command.equalsIgnoreCase("l")) {
  rotateLeft(300);
}
else if (command.equalsIgnoreCase("r")) {
  rotateRight(300);
}
else if (command.equalsIgnoreCase("sl")) {
  slowRotateLeft(300);
}
else if (command.equalsIgnoreCase("sr")) {
  slowRotateRight(300);
}
else if (command.equalsIgnoreCase("z")) {
  inPattern = true;
  zigzagPattern();
}
else if (command.equalsIgnoreCase("m")) {
  inPattern = true;
  lawnmowerPattern();
}
else if (command.equalsIgnoreCase("rnd")) {
  inPattern = true;
  randomPattern();
}
```

```
  else if (command.equalsIgnoreCase("c")) {
    moveForward(3200);
  }
  else if (command.equalsIgnoreCase("s")) {
    stopMotors();
    manualOverride = false;
  }
  else {
    Serial.println("Invalid command! Use f, b, l, r, sl, sr, z, m, rnd, c, s, or
    ↪  argument-based commands: fa <steps>, ba <steps>, la <steps>, ra <steps>");
  }
}

void slowRotateLeft(int steps) {
  Serial.println("Manual: Slow Rotating Left");

  // Set Directions for Slow Left Rotation
  digitalWrite(DIR_PIN1, LOW);  // Motor 1 Backward
  digitalWrite(DIR_PIN2, LOW);  // Motor 2 Forward

  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(5000); // Slower rotation
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(5000);

    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}

void slowRotateRight(int steps) {
  Serial.println("Manual: Slow Rotating Right");

  // Set Directions for Slow Right Rotation
  digitalWrite(DIR_PIN1, HIGH); // Motor 1 Forward
  digitalWrite(DIR_PIN2, HIGH); // Motor 2 Backward

  for (int i = 0; i < steps; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    digitalWrite(STEP_PIN2, HIGH);
    delayMicroseconds(5000); // Slower rotation
    digitalWrite(STEP_PIN1, LOW);
    digitalWrite(STEP_PIN2, LOW);
    delayMicroseconds(5000);

    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
```

```
      if (cmd.length() > 0) {
        manualOverride = true;
        controlMotors(cmd);
        break;
      }
    }
  }
}
// Add at the top with other global variables


void zigzagPattern() {
  Serial.println("Manual: Executing Zigzag Pattern with Enhanced Obstacle Avoidance
  ↪  and Dynamic Adjustments");

  while (inPattern) {
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        inPattern = false;
        controlMotors(cmd);
        return;
      }
    }

    if (!inPattern) return;  // Check if pattern should continue

    long frontLeftDist = getDistance(TRIG_FRONT_LEFT, ECHO_FRONT_LEFT);
    long frontRightDist = getDistance(TRIG_FRONT_RIGHT, ECHO_FRONT_RIGHT);
    long leftDist = getDistance(TRIG_LEFT, ECHO_LEFT);
    long rightDist = getDistance(TRIG_RIGHT, ECHO_RIGHT);

    if (!inPattern) return;  // Check if pattern should continue

    if (frontLeftDist > FRONT_THRESHOLD && frontRightDist > FRONT_THRESHOLD) {
      moveForward(1200);
    } else {
      moveBackward(frontLeftDist < 20 || frontRightDist < 20 ? 700 : 500);
      if (leftDist > rightDist && leftDist > SIDE_THRESHOLD) {
        rotateLeft(750);
      } else if (rightDist > SIDE_THRESHOLD) {
        rotateRight(750);
      } else {
        rotateLeft(random(750, 1000));
      }
      if (!inPattern) return;
      continue;
    }

    if (!inPattern) return;  // Check if pattern should continue
    rotateRight(random(500, 800));
    if (!inPattern) return;
    moveForward(800);
    if (!inPattern) return;
    rotateLeft(random(500, 800));
  }
}
```

```
void lawnmowerPattern() {
  Serial.println("Manual: Executing Lawnmower Pattern with Enhanced Obstacle
  ↪  Avoidance");

  while (inPattern) {
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        inPattern = false;
        controlMotors(cmd);
        return;
      }
    }

    if (!inPattern) return;  // Check if pattern should continue

    long frontLeftDist = getDistance(TRIG_FRONT_LEFT, ECHO_FRONT_LEFT);
    long frontRightDist = getDistance(TRIG_FRONT_RIGHT, ECHO_FRONT_RIGHT);
    long leftDist = getDistance(TRIG_LEFT, ECHO_LEFT);
    long rightDist = getDistance(TRIG_RIGHT, ECHO_RIGHT);

    if (!inPattern) return;  // Check if pattern should continue

    if (frontLeftDist > FRONT_THRESHOLD && frontRightDist > FRONT_THRESHOLD) {
      moveForward(1400);
    } else {
      moveBackward(600);
      if (leftDist > rightDist && leftDist > SIDE_THRESHOLD) {
        rotateLeft(800);
      } else if (rightDist > SIDE_THRESHOLD) {
        rotateRight(800);
      } else {
        rotateLeft(1000);
      }
      if (!inPattern) return;
      continue;
    }

    if (!inPattern) return;  // Check if pattern should continue
    rotateRight(800);
    if (!inPattern) return;
    moveForward(1100);
    if (!inPattern) return;
    rotateLeft(800);
  }
}

void randomPattern() {
  Serial.println("Manual: Executing Random Pattern with Advanced Obstacle Avoidance");

  while (inPattern) {
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      if (cmd.length() > 0) {
        inPattern = false;
```

```
        controlMotors(cmd);
        return;
      }
    }

    if (!inPattern) return;  // Check if pattern should continue

    long frontLeftDist = getDistance(TRIG_FRONT_LEFT, ECHO_FRONT_LEFT);
    long frontRightDist = getDistance(TRIG_FRONT_RIGHT, ECHO_FRONT_RIGHT);
    long leftDist = getDistance(TRIG_LEFT, ECHO_LEFT);
    long rightDist = getDistance(TRIG_RIGHT, ECHO_RIGHT);

    if (!inPattern) return;  // Check if pattern should continue

    if (frontLeftDist > FRONT_THRESHOLD && frontRightDist > FRONT_THRESHOLD) {
      moveForward(random(700, 1400));
    } else {
      moveBackward(500);
      if (leftDist > rightDist && leftDist > SIDE_THRESHOLD) {
        rotateLeft(random(600, 900));
      } else if (rightDist > SIDE_THRESHOLD) {
        rotateRight(random(600, 900));
      } else {
        rotateLeft(random(800, 1100));
      }
      if (!inPattern) return;
      continue;
    }

    if (!inPattern) return;  // Check if pattern should continue
    if (random(0, 2) == 0) {
      rotateLeft(random(500, 800));
    } else {
      rotateRight(random(500, 800));
    }
  }
}


// ---------------------------------------------------------------------------
// Stop Function
// ---------------------------------------------------------------------------
void stopMotors() {
  Serial.println("Stopping Motors");
  digitalWrite(ENA_PIN1, HIGH); // Disable Motor 1
  digitalWrite(ENA_PIN2, HIGH); // Disable Motor 2
}
```

# C Arduino Uno Code for Ball Collection Operations

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

/*****************************************************************************
 *                                Pin Definitions
 *****************************************************************************/
const int motorPin1 = 5; // Motor driver input 1
const int motorPin2 = 6; // Motor driver input 2

// ACS712 current sensor analog pin
const int currentSensorPin = A0;

// IR Sensor Pin
const int irSensorPin = A1;

/*****************************************************************************
 *                                Servo Definitions
 *****************************************************************************/
Servo servo1; // Servo object for the first servo
Servo servo2; // Servo object for the second servo

int servo1Position = 145; // Default position of Servo 1
int servo2Position = 10;  // Default position of Servo 2
const int stepDelay = 50; // Delay between steps for smooth movement (in ms)

// Predefined Positions
const int servo1CollectPosition = 50;   // Servo 1 position for ball collection
const int servo2CollectPosition = 140;  // Servo 2 position for ball collection

const int servo1ReleasePosition = 145;  // Servo 1 position for ball release
const int servo2ReleasePosition = 10;   // Servo 2 position for ball release

/*****************************************************************************
 *                        Current Sensor / Threshold Definitions
 *****************************************************************************/
const float ACS_OFFSET_VOLTAGE = 2.5;  // V, midpoint for ACS712
const float ACS_SENSITIVITY    = 0.185; // V/A for ACS712 5A module
const float CURRENT_THRESHOLD = 0.19;  // A (190 mA threshold)

/*****************************************************************************
 *                             IR Sensor / Ball Detection
 *****************************************************************************/
// Global variable to indicate whether a ball is detected
bool ballCollected = false;


float readDistanceFromIR() {
  int rawValue = analogRead(irSensorPin);
  float voltage = (5.0 * rawValue) / 1023.0;
  float distance = 13.0 / (voltage + 0.01);
  return distance;
}
```

```
// Check IR sensor and update ballCollected if object < 8.4 cm
void checkBallDetected() {
  float distance = readDistanceFromIR();
  if (distance > 0 && distance < 8.4) {
    ballCollected = true;
  }
}

/*****************************************************************************
 *                          Servo Control Functions
 *****************************************************************************/

void moveArmGradually(int targetServo1, int targetServo2) {
  Serial.println("Moving arm gradually...");

  int steps1 = abs(targetServo1 - servo1Position);
  int steps2 = abs(targetServo2 - servo2Position);
  int maxSteps = max(steps1, steps2);

  float increment1 = (float)(targetServo1 - servo1Position) / maxSteps;
  float increment2 = (float)(targetServo2 - servo2Position) / maxSteps;

  for (int step = 0; step <= maxSteps; step++) {
    int newServo1Pos = servo1Position + round(step * increment1);
    int newServo2Pos = servo2Position + round(step * increment2);

    servo1.write(newServo1Pos);
    servo2.write(newServo2Pos);
    delay(stepDelay);
  }

  servo1Position = targetServo1;
  servo2Position = targetServo2;

  Serial.println("Arm movement complete.");
}

void moveToBallCollectPosition() {
  Serial.println("Moving to ball collecting position...");

  // Reset ballCollected whenever a new collect command is received
  ballCollected = false;

  moveArmGradually(145, 40);
  delay(220);

  moveArmGradually(90, 90);
  delay(200);

  moveArmGradually(servo1CollectPosition, servo2CollectPosition);

  Serial.println("Arm is in ball collecting position.");
}

void moveToBallReleasePosition() {
  Serial.println("Moving to ball releasing position...");

  moveArmGradually(servo1ReleasePosition, 30);
```

```
  delay(200);

  moveArmGradually(servo1ReleasePosition, 0);

  Serial.println("Arm is in ball releasing position.");
}

/*****************************************************************************
 *                          Motor Control Functions
 *****************************************************************************/

void stopMotor() {
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  delay(100);
}

float readCurrentACS712() {
  int sensorValue = analogRead(currentSensorPin);
  float sensorVoltage = (sensorValue * 5.0) / 1023.0;
  float voltageOffset = sensorVoltage - ACS_OFFSET_VOLTAGE;
  float current = voltageOffset / ACS_SENSITIVITY;
  return current;
}

void openGripper(unsigned long duration) {
  Serial.print("Opening gripper for ");
  Serial.print(duration);
  Serial.println(" ms...");

  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);

  unsigned long startTime = millis();
  while (millis() - startTime < duration) {
    float current = readCurrentACS712();
    if (current > CURRENT_THRESHOLD) {
      Serial.println("Current threshold exceeded. Stopping motor.");
      break;
    }
    delay(10);
  }

  stopMotor();
  Serial.println("Gripper opening complete.");
}

void closeGripper(unsigned long duration) {
  Serial.print("Closing gripper for ");
  Serial.print(duration);
  Serial.println(" ms...");

  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);

  unsigned long startTime = millis();
  while (millis() - startTime < duration) {
    float current = readCurrentACS712();
```

```
    if (current > CURRENT_THRESHOLD) {
      Serial.println("Current threshold exceeded. Stopping motor.");
      break;
    }
    delay(10);
  }

  stopMotor();
  Serial.println("Gripper closing complete.");
}

/*****************************************************************************
 *                                Setup Function
 *****************************************************************************/
void setup() {
  Serial.begin(9600);

  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);

  pinMode(currentSensorPin, INPUT);
  pinMode(irSensorPin, INPUT); // If digital sensor, consider pinMode(irSensorPin,
  ↪  INPUT_PULLUP);

  servo1.attach(2);
  servo2.attach(3);

  servo1.write(servo1Position);
  servo2.write(servo2Position);

  Serial.println("Gripper, Servo control, and IR sensor initialized.");
  Serial.println("Available Commands via Serial:");
  Serial.println("  'collect' -> move arm to ball collecting position");
  Serial.println("  'release' -> move arm to ball releasing position");
  Serial.println("  'o <time>' -> open gripper for <time> milliseconds");
  Serial.println("  'c <time>' -> close gripper for <time> milliseconds");
  Serial.println("  'done'    -> set 'ballCollected = false'");
  Serial.println("  'status'  -> prints whether a ball is collected");
  Serial.println("----------------------------------------\n");
}

/*****************************************************************************
 *                                Loop Function
 *****************************************************************************/
void loop() {
  // Continuously check if a ball is detected
  checkBallDetected();

  // Check for user commands via Serial
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();

    if (command.equalsIgnoreCase("collect")) {
      moveToBallCollectPosition();
    }
    else if (command.equalsIgnoreCase("release")) {
      moveToBallReleasePosition();
```

```
    }
    else if (command.startsWith("o ") || command.startsWith("O ")) {
      unsigned long duration = command.substring(2).toInt();
      openGripper(duration);
    }
    else if (command.startsWith("c ") || command.startsWith("C ")) {
      unsigned long duration = command.substring(2).toInt();
      closeGripper(duration);
    }
    // 1) DONE command: set ballCollected = false
    else if (command.equalsIgnoreCase("done")) {
      ballCollected = false;
      Serial.println("Ball collection flag reset (ballCollected = false).");
    }
    // 2) STATUS command: report whether a ball is collected
    else if (command.equalsIgnoreCase("status")) {
      if (ballCollected) {
        Serial.println("STATUS: A ball IS currently collected.");
      } else {
        Serial.println("STATUS: No ball collected.");
      }
    }
    else {
      Serial.println("Invalid command. Available: collect, release, o <time>, c
      ↪  <time>, done, status");
    }
  }
}
```

# D Logic, Control & Training Codes

## D.1 OAK-D Setup & Main Robot Controller

```
#!/usr/bin/env python3



import time
import json
import cv2
import depthai as dai
import numpy as np
import serial
import threading
import logging
import sys
from collections import deque
from enum import Enum

# Google Pub/Sub
from google.cloud import pubsub_v1
from google.api_core.exceptions import NotFound, GoogleAPICallError, RetryError
from google.oauth2 import service_account

################################################################################
# 1. CONFIGURATION CONSTANTS
################################################################################

# Google Cloud Pub/Sub Configuration
PROJECT_ID = "ball-e-1"              # <-- GCP project
COMMANDS_SUB_ID = "comms-sub"        # <-- Subscription name
UPDATES_TOPIC_ID = "progress"        # <-- Topic name for updates
SERVICE_ACCOUNT_FILE = "service.json"  # <-- service account JSON

# Arduino Ports
ARDUINO_PORT_MAIN = "/dev/ttyACM1"
ARDUINO_PORT_GRIPPER = "/dev/ttyACM0"
BAUD_RATE = 9600

# DepthAI Configuration
BLOB_PATH = "/home/Ball-E/Desktop/final/best_openvino_2022.1_6shave.blob"  # <--
↪  Update path
FRAME_WIDTH = 416
FRAME_HEIGHT = 416
FPS = 15  # Reduced FPS for Raspberry Pi 3

# Detection Parameters
BALL_CLASSES = {
    0:   "ping_pong",
    10:  "red_ball",
    11:  "blue_ball",
    12:  "green_ball",
    13:  "yellow_ball",
    1:   "soccer",
```

```
    2:   "tennis",
    99:  "user_color"  # User-defined color
}

MIN_CONFIDENCE_THRESHOLD = 0.8
STOP_DISTANCE_MIN = 0.30
STOP_DISTANCE_MAX = 0.35

CENTER_DEADZONE = 75
SLOW_MODE_THRESHOLD = 150

WAIT_BEFORE_GRIP = 15
WAIT_AFTER_GRIPPING = 15

CAMERA_FOCAL_LENGTH_PIXELS = 700
DIRECTION_BUFFER_SIZE = 5

IGNORE_TIME_WINDOW = 20.0
DIAMETER_TOLERANCE = 1.5

# Logging Configuration
LOG_FILE = "robot_pubsub.log"
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s [%(levelname)s] %(message)s',
    handlers=[
        logging.FileHandler(LOG_FILE),
        logging.StreamHandler(sys.stdout)
    ]
)

################################################################################
# 2. ENUMS AND DATA CLASSES
################################################################################

class RobotState(Enum):
    DETECTION = 1
    NAVIGATION = 2
    STOPPING = 3
    GRIPPING = 4

################################################################################
# 3. HSV DETECTION + USER COLOR
################################################################################

def in_hsv_bounds(hsv_roi, bounds):
    """
    Checks if the ROI in HSV color space falls within specified bounds.
    """
    lower = np.array(bounds["lower"], dtype=np.uint8)
    upper = np.array(bounds["upper"], dtype=np.uint8)
    mask = cv2.inRange(hsv_roi, lower, upper)
    return (mask > 0).sum()

def guess_ball_color_hsv(roi_bgr, user_bounds=None):
    """
    Determines the ball's color by first checking user-defined bounds.
    If not matched, checks against standard color ranges.
```

```
    """
    hsv_roi = cv2.cvtColor(roi_bgr, cv2.COLOR_BGR2HSV)

    # 1) Check user-defined color first
    if user_bounds is not None:
        user_count = in_hsv_bounds(hsv_roi, user_bounds)
        if user_count > 50:
            return 99  # user_color

    # 2) Otherwise, guess among standard sets
    RED_BOUNDS = [
        {"lower": [0, 120, 70], "upper": [10, 255, 255]},
        {"lower": [170, 120, 70], "upper": [180, 255, 255]},
    ]
    BLUE_BOUNDS = [{"lower": [100, 120, 70], "upper": [130, 255, 255]}]
    GREEN_BOUNDS = [{"lower": [40, 120, 70], "upper": [70, 255, 255]}]
    YELLOW_BOUNDS = [{"lower": [20, 120, 70], "upper": [35, 255, 255]}]
    PINGPONG_BOUNDS = [
        {"lower": [0, 0, 200], "upper": [180, 50, 255]},
        {"lower": [10, 100, 100], "upper": [25, 255, 255]},
    ]

    def count_pixels_in_range(hsv_image, color_bounds):
        total = 0
        for bounds in color_bounds:
            mask = cv2.inRange(hsv_image, np.array(bounds["lower"], dtype=np.uint8),
                               np.array(bounds["upper"], dtype=np.uint8))
            total += (mask > 0).sum()
        return total

    red_count = count_pixels_in_range(hsv_roi, RED_BOUNDS)
    blue_count = count_pixels_in_range(hsv_roi, BLUE_BOUNDS)
    green_count = count_pixels_in_range(hsv_roi, GREEN_BOUNDS)
    yellow_count = count_pixels_in_range(hsv_roi, YELLOW_BOUNDS)
    ping_count = count_pixels_in_range(hsv_roi, PINGPONG_BOUNDS)

    color_counts = [
        (10, red_count),
        (11, blue_count),
        (12, green_count),
        (13, yellow_count),
        (0, ping_count)
    ]
    color_counts.sort(key=lambda x: x[1], reverse=True)
    best_label, best_val = color_counts[0]
    if best_val < 50:
        return 0  # fallback to ping_pong or unknown
    return best_label

################################################################################
# 4. UTILITY FUNCTIONS
################################################################################

def send_command_to_arduino(ser, command, last_command, cooldown):
    """
    Sends commands to Arduino with cooldown to prevent spamming.
    """
    current_time = time.time()
```

```
    if not ser or not ser.is_open:
        logging.warning("Arduino serial connection not open.")
        return last_command

    if command == 's':
        # Force stop always allowed
        try:
            ser.write((command + '\n').encode())
            logging.info("[FORCED STOP] Sent command: %s", command)
        except Exception as e:
            logging.error("Error sending 's': %s", e)
        return last_command

    if command != last_command and (current_time - cooldown['last_sent'] > 0.3):
        try:
            ser.write((command + '\n').encode())
            logging.info("Sent command: %s", command)
            cooldown['last_sent'] = current_time
        except Exception as e:
            logging.error("Error sending command '%s': %s", command, e)
        return command
    return last_command

def calculate_ball_size(bbox, depth_z):
    """
    Calculates the diameter of the ball in centimeters.
    """
    pixel_width = bbox[2] - bbox[0]
    diameter_cm = (pixel_width * (depth_z * 100)) / CAMERA_FOCAL_LENGTH_PIXELS
    return diameter_cm

def calculate_gripping_time(diameter_cm):
    """
    Calculates the gripping time based on the ball's diameter.
    """
    if diameter_cm > 10:
        logging.warning("Diameter > 10 cm: Might not grip fully.")
        return 0
    return max(0, int((10 - diameter_cm) * 2000))

def bbox_center(bbox):
    """
    Returns the center coordinates of a bounding box.
    """
    (xmin, ymin, xmax, ymax) = bbox
    return ((xmin + xmax) // 2, (ymin + ymax) // 2)

def center_distance(c1, c2):
    """
    Calculates the Euclidean distance between two centers.
    """
    return np.sqrt((c1[0] - c2[0]) ** 2 + (c1[1] - c2[1]) ** 2)

def find_matching_detection_center(detections, frame_shape, old_label, old_bbox,
↪   threshold=60):
    """
    Finds a detection that matches the previous one based on label and proximity.
    """
```

```
    (old_cx, old_cy) = bbox_center(old_bbox)
    best_match = None
    best_dist = threshold

    for det in detections:
        if det.confidence < MIN_CONFIDENCE_THRESHOLD:
            continue
        if det.label != old_label:
            continue

        xmin = int(det.xmin * frame_shape[1])
        ymin = int(det.ymin * frame_shape[0])
        xmax = int(det.xmax * frame_shape[1])
        ymax = int(det.ymax * frame_shape[0])
        new_bbox = (xmin, ymin, xmax, ymax)
        (cx, cy) = bbox_center(new_bbox)
        dist = center_distance((old_cx, old_cy), (cx, cy))

        if dist < best_dist:
            best_dist = dist
            best_match = (new_bbox, det.spatialCoordinates.z / 1000.0)  # Convert mm
            ↪  to meters

    return best_match

################################################################################
# 5. DEPTHAI PIPELINE CREATION
################################################################################

def create_pipeline(blob_path):
    """
    Sets up the DepthAI pipeline.
    """
    pipeline = dai.Pipeline()

    camRgb = pipeline.createColorCamera()
    spatialDetectionNetwork = pipeline.createYoloSpatialDetectionNetwork()
    monoLeft = pipeline.createMonoCamera()
    monoRight = pipeline.createMonoCamera()
    stereo = pipeline.createStereoDepth()

    xoutRgb = pipeline.createXLinkOut()
    xoutNN = pipeline.createXLinkOut()

    xoutRgb.setStreamName("rgb")
    xoutNN.setStreamName("detections")

    # RGB Camera configuration
    camRgb.setPreviewSize(FRAME_WIDTH, FRAME_HEIGHT)
    camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_720_P)
    camRgb.setInterleaved(False)
    camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
    camRgb.setFps(FPS)

    # Mono Cameras configuration
    monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
    monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
    monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
```

```
    monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

    # Stereo Depth configuration
    stereo.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_DENSITY)
    stereo.setExtendedDisparity(False)
    stereo.setDepthAlign(dai.CameraBoardSocket.RGB)
    stereo.setSubpixel(True)
    stereo.initialConfig.setMedianFilter(dai.MedianFilter.KERNEL_3x3)
    stereo.setLeftRightCheck(True)

    # Link Mono Cameras to Stereo Depth
    monoLeft.out.link(stereo.left)
    monoRight.out.link(stereo.right)

    # Spatial Detection Network configuration
    spatialDetectionNetwork.setBlobPath(blob_path)
    spatialDetectionNetwork.setConfidenceThreshold(MIN_CONFIDENCE_THRESHOLD)
    spatialDetectionNetwork.setNumClasses(len(BALL_CLASSES))
    spatialDetectionNetwork.setCoordinateSize(4)
    spatialDetectionNetwork.setIouThreshold(0.5)
    spatialDetectionNetwork.setBoundingBoxScaleFactor(0.2)
    spatialDetectionNetwork.setDepthLowerThreshold(200)
    spatialDetectionNetwork.setDepthUpperThreshold(4000)

    # Link RGB Camera to Spatial Detection Network
    camRgb.preview.link(spatialDetectionNetwork.input)
    stereo.depth.link(spatialDetectionNetwork.inputDepth)

    # Link Spatial Detection Network to XLinkOut
    spatialDetectionNetwork.out.link(xoutNN.input)

    # Link RGB preview to XLinkOut
    camRgb.preview.link(xoutRgb.input)

    return pipeline

################################################################################
# 6. MAIN FSM CLASS WITH ENHANCED NAVIGATION
################################################################################

class BallCollector:
    def __init__(self, blob_path):
        # Initialize DepthAI Pipeline
        try:
            self.pipeline = create_pipeline(blob_path)
            logging.info("DepthAI pipeline created successfully.")
        except Exception as e:
            logging.error("Failed to create DepthAI pipeline: %s", e)
            sys.exit(1)

        self.current_state = RobotState.DETECTION
        self.last_command = None
        self.cooldown = {'last_sent': 0}

        self.ball_last_seen = time.time()
        self.distance_buffer = deque(maxlen=DIRECTION_BUFFER_SIZE)

        self.last_grip_time = 0
```

```
        self.last_grip_class = None
        self.last_grip_diameter = 0
        self.current_target = None
        self.lost_counter = 0

        # Variables updated via Pub/Sub
        self.running = False
        self.classification_mode = 1
        self.specified_ball_classes = [10, 2]
        self.movement_mode = "idle"
        self.user_color_bounds = None

        # New Variable: Pending Movement Mode
        self.pending_movement_mode = None

        # Setup Pub/Sub Clients
        try:
            self.credentials = service_account.Credentials.from_service_account_file(
            ↪  SERVICE_ACCOUNT_FILE)
            self.subscriber = pubsub_v1.SubscriberClient(credentials=self.credentials)
            self.publisher = pubsub_v1.PublisherClient(credentials=self.credentials)
            self.commands_sub_path = self.subscriber.subscription_path(PROJECT_ID,
            ↪  COMMANDS_SUB_ID)
            self.updates_topic_path = self.publisher.topic_path(PROJECT_ID,
            ↪  UPDATES_TOPIC_ID)
            logging.info("Successfully connected to Pub/Sub.")
        except Exception as e:
            logging.error("Failed to connect to Pub/Sub: %s", e)
            sys.exit(1)

        # Initialize Serial Connections
        self.ser_main = None
        self.ser_gripper = None
        self.initialize_serial_connections()

        # Initialize DepthAI Device
        try:
            self.device = dai.Device(self.pipeline)
            self.qRgb = self.device.getOutputQueue(name="rgb", maxSize=4,
            ↪  blocking=False)
            self.qDet = self.device.getOutputQueue(name="detections", maxSize=4,
            ↪  blocking=False)
            logging.info("DepthAI device initialized successfully.")
        except Exception as e:
            logging.error("Failed to initialize DepthAI device: %s", e)
            self.cleanup()
            sys.exit(1)

        # Thread Control
        self.stop_event = threading.Event()

    def initialize_serial_connections(self):
        """
        Initializes serial connections to Arduino devices.
        """
        try:
            self.ser_main = serial.Serial(ARDUINO_PORT_MAIN, BAUD_RATE, timeout=1)
            logging.info("Connected to Arduino Main at %s", ARDUINO_PORT_MAIN)
```

```
        except Exception as e:
            logging.error("Failed to connect to Arduino Main at %s: %s",
            ↪  ARDUINO_PORT_MAIN, e)
            self.ser_main = None

        try:
            self.ser_gripper = serial.Serial(ARDUINO_PORT_GRIPPER, BAUD_RATE,
            ↪  timeout=1)
            logging.info("Connected to Arduino Gripper at %s", ARDUINO_PORT_GRIPPER)
        except Exception as e:
            logging.error("Failed to connect to Arduino Gripper at %s: %s",
            ↪  ARDUINO_PORT_GRIPPER, e)
            self.ser_gripper = None

    def run(self):
        """
        Starts the main operational threads.
        """
        try:
            # Start Pub/Sub Listener Thread
            pubsub_thread = threading.Thread(target=self.pubsub_listener, daemon=True)
            pubsub_thread.start()
            logging.info("Pub/Sub listener thread started.")

            # Start DepthAI Processing Thread
            depthai_thread = threading.Thread(target=self.depthai_processor,
            ↪  daemon=True)
            depthai_thread.start()
            logging.info("DepthAI processor thread started.")

            # Keep the main thread alive
            while not self.stop_event.is_set():
                time.sleep(1)
        except KeyboardInterrupt:
            logging.info("KeyboardInterrupt received. Shutting down.")
            self.stop_event.set()
        except Exception as e:
            logging.error("Unexpected error: %s", e)
            self.stop_event.set()
        finally:
            self.cleanup()

    def pubsub_listener(self):
        """
        Listens for incoming Pub/Sub commands and processes them using Streaming Pull
        ↪  with Callbacks.
        """
        def callback(message):
            cmd_str = message.data.decode("utf-8")
            logging.info("Received Pub/Sub command: %s", cmd_str)
            try:
                cmd_data = json.loads(cmd_str)
                self.handle_robot_command(cmd_data)
            except json.JSONDecodeError:
                logging.warning("Invalid JSON command: %s", cmd_str)
            message.ack()

        while not self.stop_event.is_set():
```

```
        try:
            streaming_pull_future = self.subscriber.subscribe(
                self.commands_sub_path,
                callback=callback
            )
            logging.info(f"Listening for messages on
            ↪  {self.commands_sub_path}..\n")

            # Blocks indefinitely, will exit on exception
            streaming_pull_future.result(timeout=None)
        except (GoogleAPICallError, RetryError) as e:
            logging.error("Pub/Sub listener encountered an error: %s", e)
            time.sleep(5)  # Wait before retrying
        except NotFound:
            logging.error("Pub/Sub subscription not found: %s",
            ↪  self.commands_sub_path)
            time.sleep(5)
        except Exception as e:
            logging.error("Unexpected error in Pub/Sub listener: %s", e)
            time.sleep(5)

    def depthai_processor(self):
        """
        Processes DepthAI frames and handles FSM transitions.
        """
        while not self.stop_event.is_set():
            if not self.running:
                time.sleep(0.1)
                continue
            try:
                inRgb = self.qRgb.tryGet()
                inDet = self.qDet.tryGet()
                if inRgb and inDet:
                    frame = inRgb.getCvFrame()
                    detections = inDet.detections

                    if self.current_state == RobotState.DETECTION:
                        self.handle_detection_state(frame, detections)
                    elif self.current_state == RobotState.NAVIGATION:
                        self.handle_navigation_state(frame, detections)
                    elif self.current_state == RobotState.STOPPING:
                        self.handle_stopping_state()
                    elif self.current_state == RobotState.GRIPPING:
                        self.handle_gripping_state(frame)

                    # Publish status updates periodically or based on state changes
                    self.publish_update()
            except Exception as e:
                logging.error("Error in DepthAI processor: %s", e)
            time.sleep(0.01)  # Small delay to prevent CPU overuse

    def handle_robot_command(self, cmd_data):
        """
        Handles incoming robot commands from Pub/Sub.
        """
        action = cmd_data.get("action", "")
        payload = cmd_data.get("payload", None)
```

```
if action == "start":
    if not self.running:
        self.running = True
        logging.info("Action: start => running=True")

        # **Publish the current movement mode upon receiving 'start'**
        self.publish_movement_mode(self.movement_mode)

        # **Send the current movement mode to Arduino upon 'start'**
        self.send_movement_mode_to_arduino(self.movement_mode)

        # Execute any pending movement mode
        if self.pending_movement_mode:
            logging.info("Executing pending movement mode: %s",
            ↪  self.pending_movement_mode)
            self.send_movement_command(self.pending_movement_mode)
            self.pending_movement_mode = None
    else:
        logging.info("Action: start received but robot is already running.")

elif action == "stop":
    if self.running:
        self.running = False
        logging.info("Action: stop => running=False and forced stop.")
        self.send_arduino_command('s')
        self.publish_movement_mode("stopped")

        # Reset FSM
        self.current_state = RobotState.DETECTION
        self.current_target = None  # Reset current_target
        logging.debug("Resetting current_target to None after stop.")
    else:
        logging.info("Action: stop received but robot is already stopped.")

elif action == "force_stop":
    if self.running:
        logging.info("Action: force_stop")
        self.send_arduino_command('s')
        self.publish_movement_mode("force_stopped")

        # Reset FSM
        self.current_state = RobotState.DETECTION
        self.current_target = None  # Reset current_target
        logging.debug("Resetting current_target to None after force_stop.")
    else:
        logging.info("Action: force_stop received but robot is already
        ↪  stopped.")

elif action == "set_mode":
    if isinstance(payload, int):
        self.classification_mode = payload
        logging.info("Action: set_mode => classification_mode=%d", payload)
        self.publish_movement_mode(f"mode_set_{payload}")
    else:
        logging.warning("Action: set_mode received with invalid payload: %s",
        ↪  payload)

elif action == "set_classes":
```

```
            if isinstance(payload, list):
                self.specified_ball_classes = payload
                logging.info("Action: set_classes => specified_ball_classes=%s",
                ↪  payload)
                self.publish_movement_mode(f"classes_set_{payload}")
            else:
                logging.warning("Action: set_classes received with invalid payload:
                ↪  %s", payload)

        elif action == "set_movement":
            self.movement_mode = payload if payload else "idle"
            logging.info("Action: set_movement => movement_mode=%s",
            ↪  self.movement_mode)
            self.publish_movement_mode(f"movement_mode_set_{self.movement_mode}")

            if self.running:
                # Execute movement command immediately
                logging.info("Robot is running. Executing movement mode: %s",
                ↪  self.movement_mode)
                self.send_movement_command(self.movement_mode)
            else:
                # Store movement mode for later execution
                self.pending_movement_mode = self.movement_mode
                logging.info("Robot is not running. Stored movement mode: %s",
                ↪  self.movement_mode)

        elif action == "set_user_color":
            if payload and "hsv_bounds" in payload:
                self.user_color_bounds = payload["hsv_bounds"]
                logging.info("Action: set_user_color => user_color_bounds=%s",
                ↪  self.user_color_bounds)
                self.publish_movement_mode("user_color_set")
            else:
                logging.warning("Action: set_user_color received with invalid payload:
                ↪  %s", payload)

        elif action == "get_updates":
            logging.info("Action: get_updates => Publishing immediate status update.")
            self.publish_update()

        elif action == "get_diagnostics":
            logging.info("Action: get_diagnostics => Publishing diagnostics.")
            self.publish_diagnostics()

        else:
            logging.warning("Unrecognized action: %s", action)

    def send_movement_mode_to_arduino(self, movement_mode):
        """
        Sends the current movement mode to Arduino.
        """
        movement_commands = {
            "zig_zag": 'z',
            "random": 'rnd',
            "lawnmower": 'm',
            "idle": 's'  # Assuming 's' stops the movement
        }
        command = movement_commands.get(movement_mode, 's')
```

```
        self.send_arduino_command(command)
        logging.info(f"Sent movement mode '{movement_mode}' to Arduino as command
        ↪  '{command}'.")

    def send_movement_command(self, movement_mode):
        """
        Maps movement mode to Arduino command and sends it.
        """
        movement_commands = {
            "zig_zag": 'z',
            "random": 'rnd',
            "lawnmower": 'm',
            "idle": 's'  # Assuming 's' stops the movement
        }
        command = movement_commands.get(movement_mode, 's')
        self.send_arduino_command(command)
        self.publish_movement_mode(movement_mode)

    def send_arduino_command(self, command):
        """
        Sends a command to the main Arduino.
        """
        if self.ser_main and self.ser_main.is_open:
            self.last_command = send_command_to_arduino(self.ser_main, command,
            ↪  self.last_command, self.cooldown)
            logging.debug("Arduino command sent: %s", command)
        else:
            logging.warning("Main Arduino serial connection not available.")

    def handle_detection_state(self, frame, detections):
        """
        Handles the DETECTION state of the FSM.
        """
        logging.debug("Entering DETECTION state.")
        if self.current_target:
            logging.debug("Current target exists. Evaluating state transition.")
            z_m = self.current_target['z']
            if STOP_DISTANCE_MIN <= z_m <= STOP_DISTANCE_MAX:
                self.current_state = RobotState.STOPPING
                logging.info("Transition to STOPPING state based on distance.")
            else:
                self.current_state = RobotState.NAVIGATION
                logging.info("Transition to NAVIGATION state based on distance.")
            return

        if not detections:
            if time.time() - self.ball_last_seen > 2 and self.last_command != 'z':
                logging.info("No detections. Sending 'z' command to search.")
                self.send_arduino_command('z')
            return

        self.ball_last_seen = time.time()
        processed = []
        now = time.time()

        for det in detections:
            if det.confidence < MIN_CONFIDENCE_THRESHOLD:
                continue
```

```
            # If labeled '0' (ping_pong), override with HSV color guess
            if det.label == 0:
                self.override_label_with_hsv(frame, det)

            # classification_mode=2 => skip if not in specified classes
            if self.classification_mode == 2 and det.label not in
            ↪  self.specified_ball_classes:
                continue

            # Ignore window check
            within_window = (now - self.last_grip_time) < IGNORE_TIME_WINDOW
            if within_window:
                z_m = det.spatialCoordinates.z / 1000.0  # Convert mm to meters
                bbox, diam = self.compute_bbox_and_diameter(frame, det, z_m)
                same_class = (det.label == self.last_grip_class)
                close_diam = (abs(diam - self.last_grip_diameter) <=
                ↪  DIAMETER_TOLERANCE)
                if same_class or close_diam:
                    logging.info("Skipping detection: same class or similar diameter
                    ↪  within ignore window.")
                    continue

            processed.append(det)

        if not processed:
            return

        # Select the detection with the highest confidence
        best = max(processed, key=lambda d: d.confidence)
        z_m = best.spatialCoordinates.z / 1000.0  # Convert mm to meters
        bbox, _ = self.compute_bbox_and_diameter(frame, best, z_m)
        self.current_target = {
            'label': best.label,
            'bbox': bbox,
            'z': z_m
        }
        self.lost_counter = 0

        if STOP_DISTANCE_MIN <= z_m <= STOP_DISTANCE_MAX:
            self.current_state = RobotState.STOPPING
            logging.info("Transition to STOPPING state based on target distance.")
        else:
            self.current_state = RobotState.NAVIGATION
            logging.info("Transition to NAVIGATION state based on target distance.")

    def handle_navigation_state(self, frame, detections):
        """
        Handles the NAVIGATION state of the FSM.
        """
        logging.debug("Handling NAVIGATION state.")
        if not self.current_target:
            self.current_state = RobotState.DETECTION
            logging.info("No current target. Transitioning to DETECTION state.")
            return

        label = self.current_target['label']
        old_bbox = self.current_target['bbox']
```

```
        match = find_matching_detection_center(detections, frame.shape, label,
        ↪  old_bbox, threshold=60)

        if match:
            new_bbox, new_z = match
            self.current_target['bbox'] = new_bbox
            self.current_target['z'] = new_z
            self.lost_counter = 0
        else:
            self.lost_counter += 1
            if self.lost_counter >= 5:
                logging.info("Lost target. Transitioning to DETECTION state.")
                self.current_target = None
                self.current_state = RobotState.DETECTION
                return
            else:
                logging.info("Lost target: count=%d", self.lost_counter)

        z_m = self.current_target['z']
        if STOP_DISTANCE_MIN <= z_m <= STOP_DISTANCE_MAX:
            self.current_state = RobotState.STOPPING
            logging.info("Transition to STOPPING state based on distance.")
            return

        xmin, ymin, xmax, ymax = self.current_target['bbox']
        center_x = (xmin + xmax) / 2
        frame_center = frame.shape[1] / 2
        deviation = abs(center_x - frame_center)

        if deviation > CENTER_DEADZONE:
            if deviation > SLOW_MODE_THRESHOLD:
                direction = 'la' if center_x < frame_center else 'ra'
            else:
                direction = 'la' if center_x < frame_center else 'ra'  # Could
                ↪  implement 'sl' and 'sr' if Arduino supports
        else:
            direction = 'fa'

        logging.debug(f"Navigation decision based on deviation: {deviation} ->
        ↪  Direction: {direction}")
        self.send_arduino_command(direction)

    def handle_stopping_state(self):
        """
        Handles the STOPPING state of the FSM.
        """
        logging.info("Entering STOPPING state. Sending stop command.")
        self.send_arduino_command('s')
        time.sleep(0.5)

        logging.info("Waiting before gripping: %d seconds.", WAIT_BEFORE_GRIP)
        time.sleep(WAIT_BEFORE_GRIP)

        # Perform gripping actions
        if self.ser_main and self.ser_main.is_open:
            try:
                self.ser_main.write(b"fa 500\n")
                logging.info("Sent command: fa 500")
```

```
            time.sleep(2.0)

            self.ser_main.write(b"la 180\n")
            logging.info("Sent command: la 180")
            time.sleep(1.5)
        except Exception as e:
            logging.error("Error sending movement commands to Arduino: %s", e)

    if self.ser_gripper and self.ser_gripper.is_open:
        try:
            self.ser_gripper.write(b"servo2_10\n")
            logging.info("Sent command: servo2_10 (lower arm)")
            time.sleep(1.0)

            self.ser_gripper.write(b"collect\n")
            logging.info("Sent command: collect (partial collect)")
            time.sleep(1.0)

            if self.ser_main and self.ser_main.is_open:
                self.ser_main.write(b"ba 50\n")
                logging.info("Sent command: ba 50")
                time.sleep(1.5)

            grip_ms = 2000
            if self.current_target:
                z_m = self.current_target['z']
                bbox = self.current_target['bbox']
                diam = calculate_ball_size(bbox, z_m)
                grip_ms = calculate_gripping_time(diam)
                logging.info("Calculated gripping time: %d ms based on diameter:
                ↪  %.2f cm", grip_ms, diam)

            cmd_str = f"c {grip_ms}\n"
            self.ser_gripper.write(cmd_str.encode())
            logging.info("Sent gripping command: %s", cmd_str.strip())
            time.sleep(grip_ms / 1000.0 if grip_ms > 0 else 2.0)

            self.ser_gripper.write(b"release\n")
            logging.info("Sent command: release")
            time.sleep(1.0)

            self.ser_gripper.write(b"o 2000\n")
            logging.info("Sent command: o 2000 (open)")
            time.sleep(2.0)

            self.ser_gripper.write(b"servo2_0\n")
            logging.info("Sent command: servo2_0 (raise arm)")
            time.sleep(1.0)
        except Exception as e:
            logging.error("Error during gripping operations: %s", e)

    logging.info("Waiting after gripping: %d seconds.", WAIT_AFTER_GRIPPING)
    time.sleep(WAIT_AFTER_GRIPPING)

    # Update grip-related variables
    if self.current_target:
        z_m = self.current_target['z']
        bbox = self.current_target['bbox']
```

```
            diameter_cm = calculate_ball_size(bbox, z_m)
            self.last_grip_class = self.current_target['label']
            self.last_grip_diameter = diameter_cm
            self.last_grip_time = time.time()
            logging.info("Grip completed: class=%d, diameter=%.2f cm",
                         self.last_grip_class, self.last_grip_diameter)

        self.current_target = None
        self.current_state = RobotState.GRIPPING
        logging.info("Transitioning to GRIPPING state.")

    def handle_gripping_state(self, frame):
        """
        Handles the GRIPPING state of the FSM.
        """
        logging.info("GRIPPING state completed. Transitioning to DETECTION state.")
        self.current_state = RobotState.DETECTION

    def compute_bbox_and_diameter(self, frame, det, z_m):
        """
        Computes the bounding box coordinates and diameter.
        """
        xmin = int(det.xmin * frame.shape[1])
        ymin = int(det.ymin * frame.shape[0])
        xmax = int(det.xmax * frame.shape[1])
        ymax = int(det.ymax * frame.shape[0])
        xmin = max(0, xmin)
        ymin = max(0, ymin)
        xmax = min(xmax, frame.shape[1])
        ymax = min(ymax, frame.shape[0])
        bbox = (xmin, ymin, xmax, ymax)
        diam = calculate_ball_size(bbox, z_m)
        return bbox, diam

    def override_label_with_hsv(self, frame, det):
        """
        Overrides the detection label based on HSV color analysis.
        """
        xmin = int(det.xmin * frame.shape[1])
        ymin = int(det.ymin * frame.shape[0])
        xmax = int(det.xmax * frame.shape[1])
        ymax = int(det.ymax * frame.shape[0])
        xmin = max(0, xmin)
        ymin = max(0, ymin)
        xmax = min(xmax, frame.shape[1])
        ymax = min(ymax, frame.shape[0])

        roi_bgr = frame[ymin:ymax, xmin:xmax]
        new_label = guess_ball_color_hsv(roi_bgr, user_bounds=self.user_color_bounds)
        det.label = new_label
        name = BALL_CLASSES.get(new_label, 'unknown')
        logging.info("HSV Override: label=%d, className=%s", new_label, name)

    def publish_update(self):
        """
        Publishes status updates to Pub/Sub.
        """
        status_data = {
```

```
            "type": "status",
            "state": self.current_state.name,
            "running": self.running,
            "classification_mode": self.classification_mode,
            "specified_ball_classes": self.specified_ball_classes,
            "movement_mode": self.movement_mode,
            "user_color_bounds": self.user_color_bounds,
            "timestamp": time.time()
        }
        try:
            data_str = json.dumps(status_data).encode("utf-8")
            self.publisher.publish(self.updates_topic_path, data_str)
            logging.info("Published status update.")
        except Exception as e:
            logging.error("Failed to publish status update: %s", e)

    def publish_movement_mode(self, mode):
        """
        Publishes the current movement mode to the 'progress' topic.
        """
        movement_data = {
            "type": "movement_mode",
            "mode": mode,
            "timestamp": time.time()
        }
        try:
            data_str = json.dumps(movement_data).encode("utf-8")
            self.publisher.publish(self.updates_topic_path, data_str)
            logging.info("Published movement mode: %s", mode)
        except Exception as e:
            logging.error("Failed to publish movement mode '%s': %s", mode, e)

    def publish_diagnostics(self):
        """
        Publishes diagnostic information to Pub/Sub.
        """
        diag_data = {
            "type": "diagnostics",
            "depthai_connected": self.device is not None,
            "arduino_main_connected": self.ser_main is not None and
            ↪  self.ser_main.is_open,
            "arduino_gripper_connected": self.ser_gripper is not None and
            ↪  self.ser_gripper.is_open,
            "robot_state": self.current_state.name,
            "timestamp": time.time()
        }
        try:
            data_str = json.dumps(diag_data).encode("utf-8")
            self.publisher.publish(self.updates_topic_path, data_str)
            logging.info("Published diagnostics.")
        except Exception as e:
            logging.error("Failed to publish diagnostics: %s", e)

    def cleanup(self):
        """
        Cleans up resources before shutting down.
        """
        logging.info("Cleaning up resources.")
```

```
        try:
            if self.device:
                self.device.close()
                logging.info("DepthAI device closed.")
        except Exception as e:
            logging.error("Error closing DepthAI device: %s", e)

        try:
            if self.ser_main and self.ser_main.is_open:
                self.ser_main.close()
                logging.info("Arduino Main serial connection closed.")
        except Exception as e:
            logging.error("Error closing Arduino Main serial connection: %s", e)

        try:
            if self.ser_gripper and self.ser_gripper.is_open:
                self.ser_gripper.close()
                logging.info("Arduino Gripper serial connection closed.")
        except Exception as e:
            logging.error("Error closing Arduino Gripper serial connection: %s", e)

        try:
            self.subscriber.close()
            self.publisher.close()
            logging.info("Pub/Sub clients closed.")
        except Exception as e:
            logging.error("Error closing Pub/Sub clients: %s", e)

################################################################################
# 7. MAIN EXECUTION
################################################################################

def main():
    collector = BallCollector(BLOB_PATH)
    collector.run()

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

## D.2 Flask Pub/Sub & Commands Dispatcher API

```python
#!/usr/bin/env python3
"""
Flask API with CORS enabled and various endpoints including:
- Basic command endpoints
- User color configuration
- Scheduled commands
- Updates retrieval
- Health diagnostics
- Graceful shutdown
"""

import os
import time
import json
import colorsys
from datetime import datetime
import atexit

from flask import Flask, request, jsonify
from flask_cors import CORS

from google.oauth2 import service_account
from google.cloud import pubsub_v1

from apscheduler.schedulers.background import BackgroundScheduler
import pytz

# Service Account Configuration
SERVICE_ACCOUNT_FILE = "service.json"
try:
    credentials = service_account.Credentials.from_service_account_file(
        SERVICE_ACCOUNT_FILE
    )
    print("SUCCESS: Loaded service account credentials from:", SERVICE_ACCOUNT_FILE)
except Exception as e:
    print("ERROR: Unable to load service account credentials.")
    raise e

# GCP Configuration
PROJECT_ID = "ball-e-1"             # Keeping original value
COMMANDS_TOPIC_ID = "comms"         # Keeping original value
UPDATES_TOPIC_ID = "progress"       # Keeping original value
UPDATES_SUB_ID = "progress-sub"     # Keeping original value

# Timezone Configuration
HEBRON_TZ = pytz.timezone("Asia/Hebron")  # Keeping original timezone

# Flask App and Pub/Sub Clients Setup
app = Flask(__name__)
CORS(app)  # Enable CORS for all routes

publisher = pubsub_v1.PublisherClient(credentials=credentials)
subscriber = pubsub_v1.SubscriberClient(credentials=credentials)

commands_topic_path = publisher.topic_path(PROJECT_ID, COMMANDS_TOPIC_ID)
```

```
updates_topic_path = publisher.topic_path(PROJECT_ID, UPDATES_TOPIC_ID)
updates_sub_path = subscriber.subscription_path(PROJECT_ID, UPDATES_SUB_ID)

# Background Scheduler Setup
scheduler = BackgroundScheduler(timezone=HEBRON_TZ)
scheduler.start()

def publish_scheduled_command(action, payload=None):
    """Publish a scheduled command to the commands topic."""
    message_data = {
        "action": action,
        "timestamp": time.time()
    }
    if payload is not None:
        message_data["payload"] = payload

    data_str = json.dumps(message_data).encode("utf-8")
    publisher.publish(commands_topic_path, data=data_str)
    print(f"[SCHEDULER] Published command '{action}' (payload={payload}) at scheduled
    ↪  time.")

# Basic Command Endpoints
@app.route("/start", methods=["POST"])
def start_robot():
    msg = {"action": "start", "timestamp": time.time()}
    publisher.publish(commands_topic_path, data=json.dumps(msg).encode("utf-8"))
    return jsonify({"status": "Robot start command published"}), 200

@app.route("/stop", methods=["POST"])
def stop_robot():
    msg = {"action": "stop", "timestamp": time.time()}
    publisher.publish(commands_topic_path, data=json.dumps(msg).encode("utf-8"))
    return jsonify({"status": "Robot stop command published"}), 200

@app.route("/set_classification_mode", methods=["POST"])
def set_classification_mode():
    data = request.get_json() or {}
    mode = data.get("mode", 1)
    message_data = {"action": "set_mode", "payload": mode, "timestamp": time.time()}
    publisher.publish(commands_topic_path,
    ↪  data=json.dumps(message_data).encode("utf-8"))
    return jsonify({"status": f"Classification mode set to {mode}"}), 200

@app.route("/set_classes", methods=["POST"])
def set_classes():
    data = request.get_json() or {}
    classes = data.get("classes", [])
    message_data = {"action": "set_classes", "payload": classes, "timestamp":
    ↪  time.time()}
    publisher.publish(commands_topic_path,
    ↪  data=json.dumps(message_data).encode("utf-8"))
    return jsonify({"status": f"Classification classes set to {classes}"}), 200

@app.route("/set_movement_mode", methods=["POST"])
def set_movement_mode():
    data = request.get_json() or {}
    movement_mode = data.get("movement", "idle")
```

```
    message_data = {"action": "set_movement", "payload": movement_mode, "timestamp":
    ↪  time.time()}
    publisher.publish(commands_topic_path,
    ↪  data=json.dumps(message_data).encode("utf-8"))
    return jsonify({"status": f"Movement mode set to '{movement_mode}'"}), 200

@app.route("/set_user_color", methods=["POST"])
def set_user_color():
    data = request.get_json() or {}
    hex_color = data.get("hex_color", "#FFFFFF").lstrip('#')
    r, g, b = [int(hex_color[i:i+2], 16) for i in (0, 2, 4)]
    r_f, g_f, b_f = r/255.0, g/255.0, b/255.0

    h, s, v = colorsys.rgb_to_hsv(r_f, g_f, b_f)
    h_deg = int(h * 180)
    s_255 = int(s * 255)
    v_255 = int(v * 255)

    hue_tol = 10
    sat_tol = 40
    val_tol = 40

    h_lower = max(0, h_deg - hue_tol)
    h_upper = min(180, h_deg + hue_tol)
    s_lower = max(0, s_255 - sat_tol)
    s_upper = min(255, s_255 + sat_tol)
    v_lower = max(0, v_255 - val_tol)
    v_upper = min(255, v_255 + val_tol)

    hsv_bounds = {"lower": [h_lower, s_lower, v_lower],
                  "upper": [h_upper, s_upper, v_upper]}

    msg = {
        "action": "set_user_color",
        "payload": {
            "class": 99,
            "hsv_bounds": hsv_bounds
        },
        "timestamp": time.time()
    }
    publisher.publish(commands_topic_path, data=json.dumps(msg).encode("utf-8"))
    return jsonify({
        "status": "User color command published",
        "hex_color": f"#{hex_color.upper()}",
        "hsv_bounds": hsv_bounds
    }), 200

@app.route("/schedule_commands", methods=["POST"])
def schedule_commands():
    data = request.get_json() or {}
    schedule_entries = data.get("schedule", [])
    if not schedule_entries:
        return jsonify({"error": "No schedule entries provided"}), 400

    scheduled_jobs = []
    for entry in schedule_entries:
        action = entry.get("action")
        run_time_str = entry.get("run_time")
```

```
            payload = entry.get("payload")

            if not action or not run_time_str:
                continue

            # Convert the string to a naive datetime, then localize to Hebron's timezone
            naive_dt = datetime.fromisoformat(run_time_str)
            dt_hebron = HEBRON_TZ.localize(naive_dt)

            scheduler.add_job(
                func=publish_scheduled_command,
                trigger="date",
                run_date=dt_hebron,
                args=[action, payload]
            )
            scheduled_jobs.append({
                "action": action,
                "payload": payload,
                "scheduled_for": dt_hebron.isoformat()
            })

        return jsonify({
            "status": "Commands scheduled",
            "jobs": scheduled_jobs
        }), 200

    @app.route("/get_all_updates", methods=["GET"])
    def get_all_updates():
        """
        Pulls messages from movement, updates, and diagnostics subscriptions
        and returns them in a single JSON response.
        """
        # Define subscription paths
        movement_sub_path = subscriber.subscription_path(PROJECT_ID, "movement-sub")
        updates_sub_path = subscriber.subscription_path(PROJECT_ID, "updates-sub")
        diagnostics_sub_path = subscriber.subscription_path(PROJECT_ID, "diagnostics-sub")

        # Function to pull messages from a subscription
        def pull_messages(subscription_path, max_messages=5):
            response = subscriber.pull(
                request={"subscription": subscription_path, "max_messages": max_messages},
                retry=None,
                timeout=5
            )
            messages = []
            ack_ids = []
            for received_message in response.received_messages:
                try:
                    msg_str = received_message.message.data.decode("utf-8")
                    messages.append(json.loads(msg_str))  # Parse JSON
                    ack_ids.append(received_message.ack_id)
                except json.JSONDecodeError:
                    continue

            if ack_ids:
                subscriber.acknowledge(
                    request={"subscription": subscription_path, "ack_ids": ack_ids}
                )
```

```
            return messages

        # Pull messages from all subscriptions
        movement_messages = pull_messages(movement_sub_path)
        updates_messages = pull_messages(updates_sub_path)
        diagnostics_messages = pull_messages(diagnostics_sub_path)

        # Combine all messages into a single response
        combined_response = {
            "movement_updates": movement_messages,
            "status_updates": updates_messages,
            "diagnostics": diagnostics_messages,
            "timestamp": time.time()
        }

        return jsonify(combined_response), 200

    @app.route("/shutdown", methods=["POST"])
    def shutdown():
        """
        Attempts a graceful shutdown of the Flask app.
        Also stops APScheduler.
        """
        # 1) Publish a 'stop' command
        msg = {"action": "stop", "timestamp": time.time()}
        publisher.publish(commands_topic_path, data=json.dumps(msg).encode("utf-8"))
        print("[SHUTDOWN] Published 'stop' command to robot-commands.")

        # 2) Stop the scheduler
        scheduler.shutdown(wait=False)
        print("[SHUTDOWN] APScheduler stopped.")

        # 3) Stop Flask
        shutdown_server()
        return jsonify({"status": "Shutting down Flask server."}), 200

    def shutdown_server():
        from flask import request
        func = request.environ.get('werkzeug.server.shutdown')
        if func is None:
            raise RuntimeError("Not running with the Werkzeug Server")
        func()

    if __name__ == "__main__":
        print("Starting Flask API with service account:", SERVICE_ACCOUNT_FILE)
        atexit.register(lambda: scheduler.shutdown(wait=False))
        app.run(debug=True, host='0.0.0.0', port=int(os.environ.get('PORT', 8080)))
```

## D.3 Yolo Training Notebook

```
!pip install ultralytics

from ultralytics import YOLO

from google.colab import drive
drive.mount('/content/drive')

# Unzip the dataset into /content/dataset
import zipfile

with zipfile.ZipFile('/content/drive/MyDrive/Ball-E/FINAL.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/dataset')


# Write the dataset configuration file
with open('/content/dataset.yaml', 'w') as f:
    f.write("""
train: /content/dataset/FINAL/train/images
val: /content/dataset/FINAL/valid/images
test: /content/dataset/FINAL/test/images

nc: 19  # Replace with the actual number of classes (e.g., 18 for 3 sports types and
↪  15 pool balls)
names: ['ping_pong', 'soccer', 'tennis', 'pool_1', 'pool_2', 'pool_3', 'pool_4',
↪  'pool_5',
        'pool_6', 'pool_7', 'pool_8', 'pool_9', 'pool_10', 'pool_11', 'pool_12',
        ↪  'pool_13',
        'pool_14', 'pool_15', 'pool_16']
""")


# Initialize and train the YOLO model
model = YOLO('yolov8n.pt')

model.train(data='/content/dataset.yaml', epochs=150, imgsz=416, batch=256)
```

## D.4 Testing Notebook

```
 from ultralytics import YOLO

# 1. Load trained weights (best.pt)
model_path = 'runs/detect/ex1/weights/best.pt'
model = YOLO(model_path)

# 2. Evaluate (test) on the test split from dataset.yaml
#    This will output metrics such as mAP, precision, and recall.
test_metrics = model.val(data='/content/dataset.yaml', split='test')
print(test_metrics)

model.predict(source='/content/dataset/FINAL/test/images', save=True)
```

# E Web App Components

## E.1 Button

```
import {button} from "../styles/button.module.css"
export const Button = ({children , ...rest}) => {
    return (
        <>
            <button  {...rest}>{children}</button>
        </>
    )
}
```

## E.2 Header

```
import '../styles/header.css';
import { useRef, useState, useEffect } from 'react';
import {sendCommand} from "../sendCommand.js";


export default function Header() {
    const [section, setSection] = useState({
        clicked: false,
        selectedSection: 'select balls and movement style',
    });
    const [start, setStart] = useState(false);
    const sectionList = ['select balls and movement style', 'manual movement', 'manual
    ↪  gripping'];
    const dropdownRef = useRef(null);

    const toggleListHandler = () => {
        setSection(prevState => ({
            ...prevState,
            clicked: !prevState.clicked,
        }));
    };
    const handleStartClick = async () => {
        if (start) {
            await sendCommand('stop', {}, 'POST');
        } else {
            await sendCommand('start', {}, 'POST');
        }
        setStart(prevState => !prevState);
    };

    const onSelectSection = (event) => {
        const selectedText = event.target.textContent;
        setSection({
            clicked: false, // Close the list after selection
            selectedSection: selectedText
        });
    };

    const handleClickOutside = (event) => {
```

```
        if (dropdownRef.current && !dropdownRef.current.contains(event.target)) {
            setSection(prevState => ({
                ...prevState,
                clicked: false,
            }));
        }
    };

    useEffect(() => {
        document.addEventListener('click', handleClickOutside);

        return () => {
            document.removeEventListener('click', handleClickOutside);
        };
    }, []);

    return (
        <>
            <button onClick={handleStartClick} className="dialogButtons
            ↪  floatingButton" >
                {start ? 'Stop' :'Start'}
            </button>

        </>
    );
}
```

## E.3 Manual Controller

```
import "../styles/manualController.css"
export const ManualController = () => {
    return (
        <section className="manual">
            <div className="streaming">
                <h2>Manual Controller</h2>
                <div><h1>Vedio Streaming</h1></div>
            </div>
            <h2 id="up1"></h2>
            <h2 id="down1"></h2>
            <h2 id="left1"></h2>
            <h2 id="right1"></h2>
            <h2 id="up2"></h2>
            <h2 id="down2"></h2>
            <h2 id="left2"></h2>
            <h2 id="right2"></h2>
            <h2 id="up3"></h2>
            <h2 id="down3"></h2>
            <h2 id="left3"></h2>
            <h2 id="right3"></h2>
            <h2 id="outline1"></h2>
        </section>
    )
}
```

## E.4 Movement Options

```
import {SelectOption} from "./SelectOption.jsx";
import {useEffect, useRef} from "react";
import {useMovementRefs} from "./ContextData/MovementProvider.jsx";

export const MovementOptions = () => {
    const { randomMovement, zigZagMovement, lawnMower } = useMovementRefs();

    return (
        <>
          <h3>Select Movement type</h3>
          <div className="inputs">
              <SelectOption type="radio" label="circular" name="movementType"
              ↪  ref={randomMovement} />
              <SelectOption type="radio" label="ZigZag" name="movementType"
              ↪  ref={zigZagMovement} />
              <SelectOption type="radio" label="Lawn Mower" name="movementType"
              ↪  ref={lawnMower} />

          </div>
        </>
    )
}
```

## E.5 Select Balls And Movement Style

```
import {Button} from "./Button.jsx";
import {SelectAllBallsModal} from "./Modals/SelectAllBallsModal.jsx";
import {MovementProvider} from "./ContextData/MovementProvider.jsx";
import {SelectCustomBallsModal} from "./Modals/SelectCustomBallsModal.jsx";
import {useEffect, useRef, useState} from "react";
import "../styles/SelectBallsAndMovementStyle.css"
import {button} from "../styles/button.module.css"
import {sendCommand} from "../sendCommand.js";
export const SelectBallsAndMovementStyle = () => {

    const allBalls = useRef();
    const customBalls = useRef();
    const [data, setData] = useState({
        balls: [],
        movementType: "",
        mode: ''
    })

    const balls = useRef([]);
    const movementType = useRef('');
    const mode = useRef('');
    useEffect(() => {
        // console.log(JSON.stringify(balls.current) + " vs "+
        ↪  JSON.stringify(data.balls) )
        if (JSON.stringify(balls.current) !== JSON.stringify(data.balls)) {
            console.log("Data is Changed");
            balls.current = data.balls;
            sendCommand('set_classes', {"classes": data.balls});
        }
```

```
    if (movementType.current !== data.movementType) {
        console.log("Movement is Changed");
        movementType.current = data.movementType;
        sendCommand('set_movement_mode', {"movement": data.movementType});

    }

    if (mode.current !== data.mode) {
        console.log("Mode is Changed");
        mode.current = data.mode;
        sendCommand('set_classification_mode', {"mode": Number(data.mode)},
        ↪  "POST");
    }
}, [data]);
function handleReturnClick(){
    setData({
        movementType: 'random',
        mode: '2',
        balls: [1, 9999]
    });

}
return (
    <>
        <section className="selectBallsSec">
            <button onClick={handleReturnClick} className="dialogButtons
            ↪  floatingButton2" >
                Return Home
            </button>
            <MovementProvider>
                  <SelectAllBallsModal onSubmit={setData} ref={allBalls} />
                </MovementProvider>
                <MovementProvider>
                  <SelectCustomBallsModal onSubmit={setData} ref={customBalls} />
                </MovementProvider>
            <form className="form" method={'dialog'}>
                <div className="title-area">
                    <p className="text-area">Choose the type </p>
                    <svg xmlns="http://www.w3.org/2000/svg" width="150"
                    ↪  height="170" className={"arrow"} viewBox="0 0 24 24"
                    ↪  fill="#103126" preserveAspectRatio="none">

                        <rect x="10" y="5" width="4" height="12" />

                        <polygon points="4,16 12,22 20,16" />
                    </svg>


                </div>
                <div className="buttons-area">
                    <Button onClick={()=>
                            allBalls.current.showModal()}
                            className={button}
                    >Collect All Balls</Button>
                    <Button onClick={()=>
                        customBalls.current.showModal()}
                            className={button}
```

```
                        >Collect Specific Balls</Button>
                    </div>
                </form>
            </section>
        </>
    )
}
```

## E.6 Select Option

```
import {forwardRef} from "react";
import "../styles/checkBox.css"
import {FaTableTennis} from "react-icons/fa";
import {FaTableTennisPaddleBall} from "react-icons/fa6";
import {GiEightBall, GiPingPongBat, GiTennisBall, GiTennisCourt, GiTennisRacket} from
↪  "react-icons/gi";
import {PiPingPong, PiPingPongBold} from "react-icons/pi";
import {TbPingPong} from "react-icons/tb";
export const SelectOption = forwardRef (({type, styles, label, icon, ...rest}, ref)
↪  => {
    return (
        <>
            <label style={{color: 'black', ...styles}} className={(type ==='checkbox'
            ↪  || type === 'radio')? "container label": 'label' }>
                <input type={type} ref={ref} {...rest} />
                {label}
                {icon}
                <div className={type === 'checkbox'? "checkmark": type ==='radio'?
                ↪  'checkmarkRadio': ''}></div>
            </label>



        </>

    )
})
```