TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Report

On

Enhancing Humanoid Robot Functionality Through Vision-Based Navigation with Fall Recovery and Object Manipulation

Submitted By:

Aayush Pathak (35852)

Bal Krishna Shah (35858)

Sabin Acharya (35883)

Safal Karki (35884)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March, 2025

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Report

On

Enhancing Humanoid Robot Functionality Through Vision-Based Navigation with Fall Recovery and Object Manipulation

Submitted By:

Aayush Pathak (35852)

Bal Krishna Shah (35858)

Sabin Acharya (35883)

Safal Karki (35884)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor's Degree in Electronics Communication and Information Engineering.

Under the Supervision of:

Er. Saroj Shakya

March, 2025

DECLARATION

We hereby declare that the report of the project entitled **"Enhancing Humanoid Robot Functionality Through Vision-Based Navigation with Fall Recovery and Object Manipulation"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Electronics, Communication and Information Engineering**, is a bonafide report of the work carried out by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Aayush Pathak (Class Roll No: THA077BEI002) _____

Bal Krishna Shah (Class Roll No: THA077BEI010) _____

Sabin Acharya (Class Roll No: THA077BEI035) _____

Safal Karki (Class Roll No: THA077BEI036) _____

Date: March, 2025

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a major project work entitled "**Enhancing Humanoid Robot Functionality Through Vision-Based Navigation with Fall Recovery and Object Manipulation**" submitted **by Aayush Pathak, Bal Krishna Shah, Sabin Acharya** and **Safal Karki** in partial fulfillment for the award of Bachelor's Degree in Electronics, Communication and Information Engineering. The Project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfillment of Bachelor's degree of Electronics, Communication and Information Engineering.

_____

Project Supervisor
Er. Saroj Shakya
Department of Electronics and Computer Engineering, Thapathali Campus

_____

External Examiner
Dr. Kishor Adhikari
Research and Development Chief, National College of Engineering

_____

Project Co-ordinator
Er. Sudip Rana
Department of Electronics and Computer Engineering, Thapathali Campus

_____

Mr. Umesh Kanta Ghimire
Head of Department,
Department of Electronics and Computer Engineering, Thapathali Campus

March, 2025

COPYRIGHT

The author has agreed that the library, Department of Electronics and Computer Engineering, Thapathali Campus, may make this report freely available for inspection. Moreover, the author has agreed that the permission for extensive copying of this project work for scholarly purpose may be granted by the professor/lecturer, who supervised the project work recorded herein or, in their absence, by the head of the department. It is understood that the recognition will be given to the author of this report and to the Department of Electronics and Computer Engineering, IOE, Thapathali Campus in any use of the material of this report. Copying of publication or other use of this report for financial gain without approval of the Department of Electronics and Computer Engineering, IOE, Thapathali Campus and author's written permission is prohibited. Request for permission to copy or to make any use of the material in this project in whole or part should be addressed to department of Electronics and Computer Engineering, IOE, Thapathali Campus.

ACKNOWLEDGEMENT

We are grateful towards the Department of Electronics and Computer Engineering of Thapathali Engineering Campus for scheduling this project in an efficient manner. Our sincere thanks also go to all the faculty members of the department for providing us useful information required for this project.

We are highly indebted to our supervisor **Er. Saroj Shakya** for his constant guidance and support throughout this project.

Last but not the least, we would like to thank our friends, families, seniors and juniors for their kind support and encouragement.

Aayush Pathak (Class Roll No: THA077BEI002)

Bal Krishna Shah (Class Roll No: THA077BEI010)

Sabin Acharya (Class Roll No: THA077BEI035)

Safal Karki (Class Roll No: THA077BEI036)

ABSTRACT

The robotics sector struggles to integrate vision-based navigation on a bipedal humanoid robot capable of performing human-like tasks. Although the use of ultrasonic sensors and infrared sensors is the traditional method for object detection, it has deep drawbacks such as low range, high cost and sensitive to environment. **"Enhancing Humanoid Robot Functionality Through Vision-Based Navigation with Fall Recovery and Object Manipulation"** proposes to give vision to the robot making it capable to transport objects from one location to another. The two ESP32-CAMs setup is used as stereo camera for image capturing, employing the use of YOLOv11 for object detection, and the principle of stereo camera for depth calculation. With the use of one of the most robust and accurate object detection algorithms available today, the project aims to enhance object transportation within the visual range of the robot. The final robot is able to navigate intelligently as well as grab object using image processing. The developed humanoid robot encompasses the feature of automatic fall recovery in simulation and natural human movement patterns through the use of kinematical calculations showcasing potential application in hazardous environment, industrial automation and interplanetary exploration.

Keywords: Autonomous Navigation, Degree of Freedom, Distance Estimation, Humanoid Robot, YOLOv11

Table of Content

DECLARATION ..... **I**

CERTIFICATE OF APPROVAL ..... **II**

COPYRIGHT ..... **III**

ACKNOWLEDGEMENT ..... **IV**

ABSTRACT ..... **V**

LIST OF FIGURES ..... **XII**

LIST OF TABLES ..... **XV**

LIST OF ABBREVIATIONS ..... **XVI**

1. INTRODUCTION **1**

1.1 Background 1

1.2 Motivation 2

1.3 Problem Definition 2

1.4 Objective 3

1.5 Application 3

1.5.1 Autonomous Logistics 3

1.5.2 Hazardous Environments 4

1.5.3 Robotic Assistants 4

1.5.4 Space Explorations 4

1.5.5 Entertainment 4

1.6 Scope 5

1.6.1 Capabilities of the Project 5

1.6.2 Limitations of the Project 5

2. LITERATURE REVIEW **6**

3. REQUIREMENT ANALYSIS **15**

3.1 Project Requirement 15

3.1.1 Hardware Requirements 15

3.1.2 Software Requirements 15
3.2 Feasibility Study 15
3.2.1 Hardware Feasibility 15
3.2.2 Software Feasibility 17
3.2.3 Economic Feasibility 17
3.2.4 Schedule Feasibility 17
4. SYSTEM ARCHITECTURE AND METHODOLOGY 18
4.1 System Block Diagram 18
4.2 System Components 19
4.2.1 Humanoid Robot 19
4.2.2 Node Server 19
4.2.3 Image processing Unit 19
4.2.4 Power Supply Unit 19
4.3 Working Principle 20
4.3.1 Humanoid Robot Design 20
4.3.2 Circuit Diagram 24
4.3.3 MPU6050 25
4.3.4 LM7805 Voltage Regulator 28
4.3.5 9A Buck Converter 29
4.3.6 Servo Motor (SG90) 32
4.3.7 LU9685-20CU 16-Channel Servo Controller 35
4.3.8 MG996R Servo Motor 36
4.3.9 Battery 37
4.3.10 SMPS 38
4.3.11 Data Transmission Mechanism 39
4.3.12 Object Detection Using YOLO 39
4.3.13 YOLOv11 51

4.3.14 Estimating Distance via Stereo Camera 53
4.3.15 Bot Movement Planning 57
4.3.16 Control System 58
4.3.17 Kinematics 64
4.3.18 Balancing the Robot 68
4.3.19 Incorporating into Simulation 70
4.4 System Flowchart 73
4.5 Performance Evaluation 74
4.5.1 Object Detection 74
4.5.2 Navigation 75
4.5.3 Fall Recovery 75
4.5.4 Pick and Drop Objects 75
4.5.5 Distance Estimation 76
5. IMPLEMENTATION DETAILS 77
5.1 Network Setup 77
5.2 Dataset Generation 77
5.3 Data Augmentation 78
5.3.1 Data Augmentation Steps 78
5.4 Model Training 78
5.5 Server Implementation 79
5.5.1 Receiving Image Stream 79
5.5.2 Image Processing 80
5.5.3 Serving Clients 81
5.6 Hardware Control 82
5.6.1 Smooth Inter-Frame Interpolation 83
5.7 App Implementation 85
5.7.1 Motion Planning Interface 85

5.7.2 Use Case of Application 86
5.8 Robot Model 86
5.8.1 Setting Up Simulation Environment 87
5.8.2 Importing Robot Model in the Scene 87
5.9 Robot Movement 91
5.9.1 Forward Movement 91
5.9.2 Sideways Movement 92
5.9.3 Rotation Movement 92
5.9.4 Pick and Drop Movement 92
5.10 Physical Implementation of Forward Walking Motion 93
5.10.1 Motor Initialization 93
5.10.2 Dynamic Environment Implementation of Data from Walking Simulation 93
5.11 Physical Implementation of Pick-Drop Movement 94
5.12 Physical Implementation of Rotation Movement 94
5.13 Hardware Challenges and Solutions 95
6. RESULT AND ANALYSIS 97
6.1 Model Training 97
6.2 Cube Dataset 97
6.3 Parameters Calibration 104
6.4 Lagging and Heating in VGA resolution 104
6.5 Servo Motor angle multiplier effect 105
6.6 Issue with LM7805 connected in parallel 105
6.7 SMPS 106
6.8 Distance Estimation 107
6.9 Robot Movement in Simulation 110
6.9.1 Forward Movement 110

6.9.2 Sideways Movement 111
6.9.3 Pick and Drop Movement 113
6.9.4 Rotate Motion 114
6.9.5 Fall Recovery 115
6.10 Robot Assembly 116
6.11 Motions in Physical Robot 117
6.11.1 Motor Initialization 117
6.12 Walking Sequence 118
6.13 Left/Right Turn 119
6.14 Side Movement 120
6.15 Pick/Drop Movement 120
6.16 Navigation Towards the Object 121
6.17 Pick/Drop Evaluation 123
6.18 Simulation vs Real World Comparison 124
7. FUTURE ENHANCEMENTS 125
8. CONCLUSION 126
9. APPENDICES 127
Appendix A: 3D Render of Humanoid Robot 127
Appendix B: Robot Model using MG996R Servo Motors 128
Appendix C: Robot Assembly using SG90 Motors 129
Appendix D: Robot Assembly Built from MG996R Motors 130
Appendix E: Externally Powered Robot Final Design 131
Appendix F: Circuitry 132
Appendix G: Torque Measurement 135
Appendix H: Resolutions of ESP32-CAM 137
Appendix I: Physical Specifications of Humanoid Robot 138
Appendix J: Mobile Application User Interface 139

Appendix K: Source Code for Smooth Inter-Frame Interpolation 140

Appendix L: Motion Planning Interface 143

Appendix M: Project Schedule 144

Appendix N: Project Budget 145

References 146

List of Figures

Figure 1-1: Major Joints in a Human Body 1
Figure 4-1: Block Diagram of the Proposed System 18
Figure 4-2: Reference Robot Design with 18-DOFs 20
Figure 4-3: Robot Model's DOFs 21
Figure 4-4: Design of Left Arm with 3-DOFs 22
Figure 4-5: Design of Left Leg with 6-DOFs 23
Figure 4-6: Design of Torso for the Robot 24
Figure 4-7: Structure of MPU6050 26
Figure 4-8: Structure of Wafer 26
Figure 4-9: Micro Electro Mechanical System 27
Figure 4-10: Block Diagram of LM7805 Voltage Regulator 28
Figure 4-11: Circuit Diagram of Buck Converter 32
Figure 4-12: Block Diagram of SG90 32
Figure 4-13: Schematic Circuit Diagram of H-Bridge 34
Figure 4-14: Architecture of YOLOv8 40
Figure 4-15: Convolution Block 41
Figure 4-16: SiLU Activation Function Plot 43
Figure 4-17: C2f Block 44
Figure 4-18: Bottleneck Block 46
Figure 4-19: SPPF Block 47
Figure 4-20: Architecture of YOLOv11 52
Figure 4-21: C3K2 Block 53
Figure 4-22: Setup to Calculate Distance using Single Camera 54
Figure 4-23: Setup to Calculate Distance using Stereo Camera 55
Figure 4-24: Control System with Feedback 59
Figure 4-25: Three Different Scenarios of Object and View Range of Robot 60
Figure 4-26: Detection by Right Camera Only 61
Figure 4-27: Detection by Left Camera Only 62
Figure 4-28: Object on Left Side 63
Figure 4-29: Object on Right Side 64
Figure 4-30: Basic Kinematic Structure 65
Figure 4-31: Inverse Kinematics Structure and Link Length of Right Leg 66

Figure 4-32: States of Robot 71
Figure 4-33: Proposed System Flowchart 74
Figure 5-1: Use Case Diagram of Robot Mobile Application 86
Figure 5-2: URDF of Robot Model 87
Figure 5-3: Robot Model in Simulation 88
Figure 5-4: Convex Decomposed and Visibility Layers of Model 89
Figure 5-5: Convex Shape Extraction of Model 89
Figure 5-6: Torque Comparison of Servo Motors 96
Figure 6-1: Prediction of Cube Object 97
Figure 6-2: Prediction of Cubical Object 98
Figure 6-3: F1 Curve for Cube Dataset 98
Figure 6-4: PR Curve for Cube Dataset 99
Figure 6-5: Box Loss over 110 Epochs 100
Figure 6-6: DFL Loss Over 110 Epochs 101
Figure 6-7: Class Loss over 110 Epochs 101
Figure 6-8: Precision over 110 Epochs 102
Figure 6-9: Recall over 110 Epochs 103
Figure 6-10: mAP50-95 for 110 Epochs 103
Figure 6-11: Internal Circuitry of LM7805 for Additional Current Supply 105
Figure 6-12: Distance Estimation of Object 107
Figure 6-13: Distance Estimation Accuracy Plot 108
Figure 6-14: Estimation Error for Distances 108
Figure 6-15: Forward Movement in Simulation 110
Figure 6-16: Trajectory of Left and Right Feet in Forward Movement 110
Figure 6-17: Sideways Movement in Simulation 111
Figure 6-18: Trajectory of Left and Right Feet in Left Sideways Movement 112
Figure 6-19: Pick Low Object Motion 113
Figure 6-20: Pick Object at Elevated Position 113
Figure 6-21: Anti-Clockwise Rotation in Simulation 114
Figure 6-22: Fall Recovery Facing Up 115
Figure 6-23: Fall Recovery Facing Down 115
Figure 6-24: Motor Initialization in Robot 117
Figure 6-25: Walking Sequence of Robot 118
Figure 6-26: Rotating Motion of Robot 119

Figure 6-27: Sideways Movement of Robot 120
Figure 6-28: Pick Up Motion 120
Figure 6-29: Distance Covered on Each Walk Cycle 122
Figure 6-30: Alignment towards Object 123
Figure 9-1: 3D Render of Humanoid Robot (Front View) 127
Figure 9-2: 3D Rendered Isometric View of Humanoid Robot 127
Figure 9-3: Front View of Robot Model 128
Figure 9-4: Side View of Robot Model 128
Figure 9-5: Assembled Robot Body 129
Figure 9-6: Robot Assembled using MG996R Motors 130
Figure 9-7: Externally Powered Design 131
Figure 9-8: Overall Circuit Diagram 132
Figure 9-9: SMPS and Buck Converter Power Supply Unity 132
Figure 9-10: 3S LiPo Battery with Balanced Chargers 133
Figure 9-11: Battery with B3 Charger 133
Figure 9-12: LM7805 Voltage Regulation Circuit 134
Figure 9-13: Circuitry of ESP32-CAMs 134
Figure 9-14: Weight of Test Object 135
Figure 9-15: Torque by Motor on Test Object 135
Figure 9-16: Torque Measurement 136
Figure 9-17: User Interface of Mobile Application 139
Figure 9-18: App Interface for Motion Planning 143

List of Tables

Table 2-1: Summary Table of Literature Review 11
Table 4-1: Specifications of SG90 and MG996R Motors 37
Table 4-2: Denavit-Hartenberg Parameters 65
Table 4-3: DH-Parameters Configuration for Legs 67
Table 9-1: Resolutions of ESP32-CAM 137
Table 9-2: Mass Specification 138
Table 9-3: Height and Width Specifications 138
Table 9-4: Gantt Chart 144
Table 9-5: Project Budget 145

List of Abbreviations

CNN	Convolutional Neural Network
COG	Center of Gravity
CSPnet	Cross Stage Partial Network
DOF	Degree of Freedom
FK	Forward Kinematics
GG-CNN	Generative Grasping CNN
IDE	Integrated Development Environment
I2C	Inter Integrated Circuit
IK	Inverse Kinematics
IMU	Inertial Measurement Unit
IoU	Intersection over Union
ML	Machine Learning
mAP	Mean Average Precision
OBB	Oriented Bounding Boxes
PWM	Pulse Width Modulation
ReLU	Rectified Linear Unit
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SMPS	Switch Mode Power Supply
SVM	Support Vector Machine
URDF	Unified Robotics Description Format
USB	Universal Serial Bus
YOLO	You Only Look Once

1. INTRODUCTION

1.1 Background

The concept of Humanoid Robot has been around for quite a long time. With the development of automation systems, human labor workers have been imagined to be replaced with more sophisticated human-like machines capable of performing assigned tasks with far greater speed and accuracy than humans.

Furthermore, Sci-Fi movies have played a crucial role in popularizing humanoid robots among the general populace. The hype generated by science fiction has definitely influenced people's imagination about robots and their capabilities.

A typical humanoid robot is designed to mimic the human body with two legs, two arms, torso and head as seen in Figure 1-1. Based on the real human structure and joints present, the robot that is designed has all major joints that are required to perform complex maneuvers that are possible by humans with some limitations. The major joints include: Neck, Shoulder, Elbow, Hand, Pelvis, Knee and Ankle with their own limited range of motions and motor requirements.

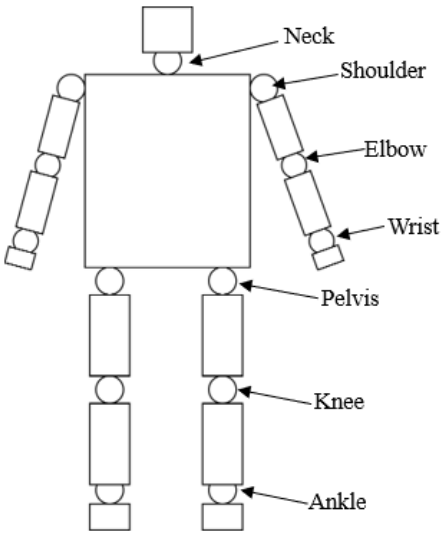


Figure 1-1: Major Joints in a Human Body

The robot incorporates multiple problem-solving approaches for executing defined tasks of object detection, recognition and grasping for picking and placing a defined object form one point to another through stereo-vision and kinematics models. It utilizes

YOLOv11 for object recognition and detection along with geometric mathematical solutions for distance estimation to visually navigate to the target location for object manipulation. With the help of a gyroscope sensor and the complete set of motors for complex degrees of freedom, the robot is able to recover from falls and continue its task along with the integration of a feedback mechanism.

1.2 Motivation

With the rise in the field of Artificial Intelligence, companies are rushing to create a humanoid robot with similar capabilities to humans. The humanoid robot, as the name suggests, has a similar degree of freedom and movement pattern as humans. Sophia, a social humanoid robot developed by a Hongkong based company Hanson Robotics is a renowned example [1].

Robo-One a popular humanoid robot competition originated in Japan showcases the capabilities of humanoid bipedal robots through dynamic and acrobatic movements twice a year serving as a platform for innovation, creativity and friendly competition in the field of humanoid robotics.

Similarly, Boston Dynamics is also continuously conducting research and experimenting to create a humanoid robot. Their robots are even capable of complex human activities like jumping, rolling etc. utilizing complex mechanics along with electronics.

Humanoid robots can act as very effective and friendly assistants in human tasks. Similar motion and structure like humans allow them to easily take over the human tasks without need of designing some other specific instruments. The prospect of humanoid robot seems more pronounced with the ML architectures like reinforcement learning and imitation learning. They can imitate humans by learning on abundance of human data.

1.3 Problem Definition

The integration of vision-based navigation in humanoid robots and bipedal robots in general is a significant challenge, specifically in environments requiring autonomous

assets transportation. Unlike wheel-based robots, bipedal robots must maintain balanced gait while navigating a dynamically changing environments, making real-time perception and decision making crucial. So, the vision-based navigation enables these robots to detect and recognize the object, identify target location and adapt to unknown environmental conditions. However, implementing such systems requires high precision sensors and computationally intensive algorithms, which are often expensive and difficult to integrate efficiently and effectively within in the constraints of humanoid platform.

Additionally, manual assets transportation is prone to human errors, including misplacement, navigation errors in unknown locations, and incorrect deliveries. In hazardous environments, such as environments containing bio-hazards or extreme conditions direct human involvement is either unsafe or impossible, creating a necessity of autonomous solutions. A humanoid robot equipped with vision-based navigation offers a robust alternative by automating transportation and deliveries, eliminating human errors and ensuring safe operations in restricted and dangerous zones. Furthermore, automating repetitive tasks using such robots reduces reliance on human labour, increasing efficiency and productivity. Developing a cost-effective and reliable vision-based navigation system for humanoid robots is a crucial step towards enhancing automation in industrial, healthcare and hazardous environments.

1.4 Objective

- To simulate the humanoid robot functionality for object identification, pick and drop with fall-recovery.
- To fabricate a humanoid robot capable of object identification, pick and drop.

1.5 Application

A humanoid robot can have a very wide range of applications based on its ability to perform complex movements and technical powers. Some of its applications are:

1.5.1 Autonomous Logistics

Humanoid robots along with similar robots equipped with vision-based navigation and object manipulation can be significantly useful for intelligent assets transportation and

delivery. Such robots can be developed to navigate complex environments autonomously, identifying and avoiding obstacles using vision systems. They can optimally and continuously handle a tight workflow and work environment of logistics reducing human's physical efforts and errors thereby enhancing the overall robustness and effectiveness of logistics operations.

1.5.2 Hazardous Environments

Humanoid Robots can be used to navigate hazardous environments which can potentially be extremely dangerous for humans such as nuclear plants, disaster zones, chemical factories and biohazard areas and perform all the activities that a human can perform. Sophisticated humanoid robots can handle hazardous materials safely and accurately increasing safety of humans reducing human exposure to danger.

1.5.3 Robotic Assistants

Humanoid robots can transform both retail shops, restaurants and home environments. Maid robots can autonomously navigate through shops, assist customers, restock shelves and also handle transactions, thereby enhancing customer service and operational efficiency. In homes as well, they can be used to perform a range of tasks such as fetching certain items, providing companionships and improving overall quality of life.

1.5.4 Space Explorations

Humanoid robots can also be utilized to their full potential in space travels and explorations. These robots can either assist humans in space travels or fully replace humans in journeys where human involvement is not possible. They can navigate alien planets, lunar and martian surfaces autonomously or assist astronauts in tasks. Robots can traverse through space freely without space suits unlike humans do which reduces complexities in human involved space travels. By deploying humanoid robots in space explorations, missions can be conducted more efficiently, safely and autonomously.

1.5.5 Entertainment

Humanoid Robots can also be an immense source of entertainment showcasing advanced humanoid robots and their capabilities in engaging and thrilling battles.

People can showcase their robot's agility, strategy and technical powers in battles against other robots.

1.6 Scope

1.6.1 Capabilities of the Project

The humanoid robot is capable to:

- Recognize and pick defined objects and drop them in defined location
- Visual navigation to object location
- Self-balancing and Fall-recovery

1.6.2 Limitations of the Project

- Robot can possess degree of freedom for major human body joints only
- Robot cannot walk on uneven surface
- Pick and drop object location must be within the robot's visual range

2. LITERATURE REVIEW

Authors of "A YOLO-GGCNN based grasping framework for mobile robots in unknown environments" [2] proposed an integrated framework for mobile robots operating in unfamiliar environments which combined an accurate map construction using a multirobot cooperative simultaneous localization and mapping (SLAM)-based algorithm with a two-step grasping module. The grasping module used YOLOv4 for object detection and YOLO-GGCNN for refining grasping predictions. The generated map is transmitted to the robotic arm through a Robot Operating System (ROS) The system achieved a capture accuracy of 86.0% and operated efficiently with a detection time of 0.11 seconds per frame. They validated the framework through applications in various environments and comparison with existing approaches.

An Object Detection and Recognition algorithm for a pick and place robot was developed by researchers at University of South Pacific [3]. They extracted feature from a cluttered scene, cropped the object and filed the object location. Then they converted it to grayscale. After proper resize procedure, they sent it to a classifier for testing. The final output produced by the algorithm was the object classification and coordinates which were to be fed to robot's system for picking the object. This algorithm showed great accuracy in image processing. Though the feasibility in integrating this algorithm to an actual robot was yet to be measured.

Moreover, E. DANDIL and K. K. ÇEVİK [4] measured the distance between a stereo camera setup composed of two Logitech C310 USB WebCams and human face by calculating the shifts in pixels between the two faces through the use of disparity maps. But before measuring the distance, the setup needed to be calibrated for the application to succeed. The accuracy was pretty decent for small distances up to 77-80 cm. But the accuracy dropped greatly with the increase in distance. A similar approach is used for calculating the distance between the humanoid robot and the object in this project.

"Robot Pathfinding Using Vision Based Obstacle Detection" [5] developed a vision-based obstacle detection and pathfinding algorithm for NAO humanoid robot using Support Vector Machine (SVM) and A* algorithms. A camera inserted in a NAO robot was used of the real time observation. To reduce the computation complexity, 640x480 pixels were converted into 20x20 pixels. The Support Vector Machine algorithm was

used for the classification of the captured image. Then, A* algorithm was used to find the optimal possible path between the start and the end point. Although their system was able find the optimal path, it was limited to vision-based data only. They didn't use any of fall recovery technique for their NAO robot, so it may not work properly if any disbalance occurred during navigation.

Related to kinematics of bipedal robot, in 2015, Rahul R. Thavai and Shishir Kumar N. Kadam in their paper "Inverse Kinematics Solution for Biped Robot," investigated the creation of an inverse kinematic joint solution for a bipedal robot [6]. Due to their complexity, biped robots, which imitated human locomotion through multiple joint mechanisms, presented significant modeling challenges in mathematics. The objective of this study was to develop and examine a biped robot with an 11-degree-of-freedom (DOF) structure. It is made up of five links that are joined by revolute joints, which include ankle, knee, and hip joints. By symbolically lowering the complexity of solving univariate polynomials. Their methodology made the inverse kinematics problem easier to understand and produced an analytical solution for a given end-effector position. The mechanical design featured modular parts made of lightweight acrylic for easy replacement and wide range of motion, based on human body ratios and ranges of motion. In order to ascertain the location and orientation of the robot's foot in relation to its torso, a thorough transformation matrix for every joint was established by formulating the kinematic model using Denavit-Hartenberg parameters. In order to provide a methodical approach to solving the inverse kinematics problem, which is essential for the accurate control and movement of biped robots, the paper detailed the calculation of joint variables using geometric methods. The suggested approach improved joint solution accuracy and effectiveness by tackling common challenges such as computational complexity and singularities linked to iterative and Jacobian matrix-centered methods.

In the paper "Inverse Kinematics Model for a 18 Degrees of Freedom Robot" (2023), Miguel Angel Ortega-Palacios, Amparo Dora Palomino-Merino, and Fernando Reyes-Cortes presented a thorough investigation addressing the inverse kinematics model of a bipedal robot with 18 degrees of freedom (DOF) [7]. They focused on the Bioloid Premium Robot, with the goal of creating a comprehensive inverse kinematic model to support future dynamic modeling efforts. The researchers drew attention to the

shortcomings of the previous research, especially the absence of detailed kinematic models for the upper and lower bodies of biped robots. They presented a thorough methodology to model the robot's forward and inverse kinematics using the Denavit-Hartenberg method. The robot was split up into four open kinematic chains for the study, which began with the support feet and went all the way to the hands and pelvis. Specific Denavit-Hartenberg parameters were used to characterize each chain in order to accurately represent the joint positions and orientations of the robot. Additionally, the forward kinematics were computed by using transformation matrices for every joint. Their work provided an organized method for achieving precise control of biped robots, addressing the computational challenges presented by high DOF systems.

The paper “Design of a Humanoid Bipedal Robot Based on Kinematics and Dynamics Analysis of Human Lower Limbs” proposed a humanoid bipedal robot based on kinematics and dynamics of human’s lower limb [8]. They designed the robot with primarily lower limbs with high-torque hip joint combined with 2-DOF parallel mechanisms and a high-torque knee joint with variable damping. They presented the overall structure and mechanical construction of the bipedal robot. They used a distributed control system approach dividing them into a main control module, a motion execution module and a sensor module with the integration of IMUs and pressure sensors. They validated their structural design and feasibility of their distributed control system through some experimental results.

Mounika Sakhineti and Sudharsan Jayabalan in their paper “Design and Fabrication of SHRALA: Social Humanoid Robot Based on Autonomous Learning Algorithm” presented asocial humanoid robot torso (upper body): SHRALA designed for human interactions and assistance, with their focus on healthcare and rehabilitation [9]. They fabricated the robot using Solidworks and 3D - printers and constructed the robot with a mobile base and robotic arms providing the motor 5 degrees of freedom each, mimicking human-like movement. They used an Arduino Mega 2560 microcontroller to control the robot’s servo and Unity3D as a graphical interface for real-time control using a joystick. Their initial results demonstrated effective control and task execution.

In the paper “Vison-Based Robot Following Using PID Control” by Chandra sekhar pati et al. presents a mobile robot following another tele operated mobile robot based

on PID controller [10]. A master robot is manually controlled through a bluetooth device that recives user command from an application. The follower robot receives the images from the kincct sensor mounted on it and recognizes the master robot. The follower robot identifies the x, y positions by employing the camera and the depth by using the Kinect depth sensor. By identifying the x, y, and z locations of the master robot, the follower robot finds the angle and distance between the master and follower robot, which is given as the error term of a PID controller. Using this the follower robot follows the master robot.

Authors of "Standalone brushless motor module optimized for legged robots" focused on the design of advanced actuators that are well optimized for small and dynamic legged robots [11]. The actuator designed consists of brushless motors with inbuilt controller and also a gear section consisting of planetary gear system with the gear ratio of 4.5:1. The controller circuit consisted of STM32 microcontroller with MOSFET driver and transistors for motor power control, AS5147 14-bit magnetic encoder to determine the angular position and velocity of the motor's rotor. The 33x33mm mini-PCB controller for each motor is able to drive the phase currents up to 40 A. The motor is able to provide a peak torque of about 3Nm with roughly 0.103Nm/A of torque for currents below 22A and a bit saturated torque on current above that. This motor with 3Nm/A is best suited for a humanoid robot too but considering the weight of each motor being 210 g, the scale of robot should be of a scale larger than what is aimed to be built in this project.

The paper "Trajectory Planning and Walking Pattern Generation of Humanoid Robot Motion" [12] focuses on the planning of walking cycle of a humanoid robot with a total of 5 joints in each leg in 9 simple steps. It describes two methods of holding balance in humanoid robots. One method is to calculate the COG of the robot and for the robot to be balanced the COG must lie within the support polygon of the robot. Another method is to find the ZMP (Zero Moment Point) which is the point where the dynamic reaction force at the contact of the foot with the ground doesn't produce any moment. This paper however uses the ZMP approach using the FSR (Force Sensing Resistor) sensor values implanted under the sole of each leg. For motion generation, the human motion was investigated and the walking cycle of 9 phases were generated and upon analyzing the ZMP trajectory, the trajectory was found to be slightly sinusoidal. The concept of ZMP

is important as it allows the motion frames of the walking cycle to be stable in zero torque from motors too but due to unavailability of FSR sensors, the formulation described in this paper cannot be implemented for this project. However qualitative measurement of ZMP i.e. inspecting if the motion frame is stable in the position for no torque from the motor is done for finding the ZMP point. This causes the motion to be non-dynamic according to the surface but the static motion induced is within the scope of this project.

The paper on "RMA: Rapid Motor Adaptation for Legged Robots" focuses on the development of an algorithm for adapting the motor control for legged robots like (humanoid robots, especially quadruped robots) to successfully walk on different terrains like sand, mud, tall grass, oily surface and many more [13]. Reinforcement Learning with bioenergetics-based reward system is used to train the quadruped robot in a simulated environment. The training in simulation used a base policy for robot's normal motion but it trained the environmental factor encoder which considered the center of mass, friction, terrain height, motor strength and also an adaptation module to compensate the motor's control algorithm for adapting to these factors. The physical testing of the algorithm was done on the A1 quadruped robot from Unitree. The results were quite satisfactory with robot successfully walking on different terrains as described earlier. Moreover, the robot was also able to dynamically adjust its motor control based on the dynamic payload added or removed from the robot's back. This shows the pronounced prospect of using AI algorithms like Reinforcement learning for motion planning of the robot rather than the manual calculative approach. Considering the scale of the project and hardware used, reinforcement learning requires more industry level expensive hardware which is what his project is to replace for. So, this can be future enhancement for the project but not a direct implementation methodology for this project.

"Design and implementation of flyback converter design and control of driving power supply for unmanned robot" [14] focused to solve one of the major problems in the design of unmanned robots like humanoid robots which is the issue with power supply. As the motors used in these types of robots draw a lot of currents from the supply instantly, lot of power supplies suffer the transient voltage drop during the process of current spike and take a certain time to stabilize to the regulated voltage. This happens

in case if the input voltage varies suddenly. This can cause the devices like microcontrollers to cause reboot and also motors to become unstable during the transient period. This paper aimed to design a driving power supply (flyback converter) with flyback transformer to solve the issue. The circuit uses transistor controlled by PWM signal to store power in primary winding (when ON) and to transfer the energy to secondary side powering the load (when OFF). PI(Proportional-Integral) controller design is used for stabilizing the output voltage. The model was tested in simulation and in physical implementation where the system was able to maintain stable output volage for changing the input voltage in range of 24V to 35V. This method uses transformer and power resistors with heat-sinks which are heavy components and causes the mass of the robot to be increased drastically as a result, it becomes hard for the robot to maintain its COG and ZMP points. Although this solved the issue of transient voltage drops, it cannot be directly implemented in this project.

Table 2-1: Summary Table of Literature Review

Title	Methodology	Inferences
A YOLO-GGCNN based grasping framework for mobile robots in unknown environments (2023) **Author: Z. Li et al.**	Uses an accurate Map Construction using a multirobot co-operative simultaneous localization and mapping SLAM based algorithm with two step gasping.	• Uses Yolov4 for the object detection. • Uses YOLO-GGCNN for refining grasping predictions. • Uses Robot Operating System as a bridge to transmit the generated map to the robot's arm control system.
Object Detection and Recognition for a Pick and Place (2014) **Author: R. Kumar et al.**	Classify and extracts the objects location coordinates from messy scene and feeds it to robot's system for picking it.	• Uses image processing algorithm to determine precise object coordinates.

Computer Vision Based Distance Measurement System using Stereo Camera View (2019) **Author: E. Danil et al.**	Calculates the shifts in pixels between the two faces through the use of disparity maps.	• Uses two Logitech C310 USB WebCams setup. • Accuracy was pretty decent for small distance up to 70-80 cm.
Robot Pathfinding Using Vision Based Obstacle (2017) **Author: R.H. Abiyev et al.**	Uses Support Vector Machine for image classification and A* algorithm to find the optimal path between start and end point.	• Reducing the picture from a real time camera into 20x20 pixel size SVM classifies the image. • A* algorithm efficiently finds the optimal path.
Inverse Kinematics Solution for Biped Robot (2015) **Author: R.R. Thavai et al.**	Formulation of Denavits-Hartenberg matrix to model bipedal robot and deduce geometrical Solution of inverse kinematics.	• Uses a biped robot with 11 DOFs. • Uses 5 DOFs for each leg and formulates kinematic modeling using Denavits-Hartenberg parameters.
Inverse Kinematics Model for a 18 Degrees of Freedom Robot (2023) **Author: M.A. Ortega-Palacios et al.**	Division of Robot into four open kinematic chains and usage of transformation matrices to describe spatial relationship among joints	• Use of DH Parameters and transformation matrices ensured high level of accuracy • Division of Robot into kinematic chains provided a modular approach • Inverse Kinematics model served as a basis for dynamic modeling
Design of a Humanoid Bipedal Robot Based on Kinematics (2020)	Design of a humanoid bipedal robot with a distributed control system encompassing main control, motion	• Division of modules into main control, motion execution and sensor

Author: D. Huang et al.	execution and sensor modules	modules improved coordination among joints
Design and Fabrication of SHRALA: Social Humanoid Robot (2020) **Author: M. Sakhineti et al.**	Design of 5 DOF arms using SolidWorks and Arduino Mega 2560 to manage servos and facilitate real-time control with USB joystick	• Streamlined approach to design humanoid robot by rapidly prototyping and testing while cycling through iterations between virtual design and fabrication using 3D printer
Vison-Based Robot Following Using PID Control (2017) **Authors: Chandra Sekhar et al.**	Implementation of robot following system. Master robot is controlled manually via Bluetooth controller. Kinect sensor captures image and depth information to locate the master robot's position. PID ensures follower maintains a constant distance.	• Combining image data with depth information provides robust position tracking. • PID ensures smooth following behavior.
Standalone brushless motor module optimized for legged robots (2022) **Authors:Piotr WASILEWSKI et. al.**	Designed a motor brushless motor with a inbuilt controller circuit to determine the angular position and velocity of the motor's rotor.	• Motor delivered a peak torque of 3Nm and 0.103Nm/A of torque for current less than 22A • The motor can weigh arround 210g
Trajectory Planning and Walking Pattern Generation of Humanoid Robot Motion (2015) **Authors: Saeed Abdolshah et. Al.**	Calculate the CG of the robot and make sure it lies within the support polygon. Find the ZMP then analyze the ZMP trajectory to generate the walking cycle.	• Force Sensing Resistor can be used to analyze ZMP • Walking cycle of humanoid robot can be planned using 5 joints for each leg in 9 steps

RMA: Rapid Motor Adaptation for Legged Robots (2021) **Authors: Deepak Pathak et. Al.**	Develop an algorithm for adapting the motor control for legged robot using reinforcement learning with bioenergetic-based reward	• Quadruped robot can walk in difficult terrain and can dynamically adjust motor control using the algorithm.
Design and implementation of flyback converter design and control of driving power supply for unmanned robot (2022) **Authors: Lingfeng Meng et. Al.**	Design a power supply controlled by a transistor to store power in a primary winding of a transformer to transfer the energy to secondary windings when power off. PI controller was used to stabilize the output voltage.	• Transformer and power resistor along with the heat sinks are able to maintain a stable output voltage on changing input voltage ia a range of 24-35V.

3. REQUIREMENT ANALYSIS

3.1 Project Requirement

3.1.1 Hardware Requirements

- **ESP32 microcontroller** is used to execute the programs to control the circuit components.
- **ESP32-CAMs** are mounted on the robot to capture the visuals and transmit it to image processing model for object detection and distance calculation
- **MPU6050 Gyroscope Sensor** is used to detect the balance state of the robot
- **MG996R Servo Motors** are used to move the joints as per the kinematical instructions
- **LU9685 16-Channel PWM Servo Driver** is used for driving the servo motors
- **Switch Mode Power Supply (SMPS)** is used for powering all the motors.

3.1.2 Software Requirements

- **Arduino IDE** is used to compile and upload codes into ESP32 microcontroller
- **CoppeliaSim** is used to simulate the physical environment as well as run the simulation
- **Node.js** is used to create a server for transmitting, receiving as well as processing the data from the robot.
- **VScode** is used for editing and debugging programs
- **Android Studio** for developing and emulating mobile application

3.2 Feasibility Study

3.2.1 Hardware Feasibility

This project incorporates ESP32, ESP32-Cam, Servo Motors, and Voltage Regulators as the nitty-gritty components. Their feasibility is explained as follows:

ESP32 is an efficient microcontroller in terms of processing power and memory for this project. The availability of Wi-Fi module in ESP32 is crucial for wireless implementation of the humanoid robot as transmission of data is necessary between the

robot and the server. It doesn't draw much power from the power supply either. For these reasons, ESP32 is feasible to be deployed in this project.

ESP32-CAM has a resolution of 1200x1600, which is more than enough for the requirements in this project. So, it is feasible to be used.

SG90 Servo Motors are very light with only 14.7 grams and a torque of 2.5kg-cm and also operates on 5V as described by the datasheet shown in Table 4-1. This motor is applicable as neck motor, due to low torque requirement in rotating the head.

MG996R Servo Motors are metal gear servo motor with 9.4 kg cm at 5V torque. These motors don't wear out easily and are ideal for lifting the mass of the robot to produce a stable motion.

16-Channel LU9685 Servo Driver allows to control the 16 servo motor simultaneously using the I2C communication protocol. Since 4 other motors can be controlled using the ESP32 PWM pins too, the 16 channels servo driver is enough for this project. It also provides an easy 48-pin interface to connect the 16 servo pins providing ground, Vcc and PWM pins for each motor, reducing the hassle to connect the motors. Moreover, it also contains the extension pins for the I2C communication protocol which can be used to interface the MPU6050 sensor.

LM7805 is a 5V voltage regulator which converts the 7-20V input supply to a regulated 5V supply with a very low error in output. Since all the components work on 5V supply, LM7805 is feasible to act as a power supply.

LiPo battery with current supply of 5A at voltage 7-12V is enough to power the microcontroller boards.

SMPS has a current rating of 40 A with Voltage rating of 5V. The current requirement for proper torque output of all motors is satisfied with this power supply.

3.2.2 Software Feasibility

CoppeliaSim's controller code can be scripted in the python programming language and Lua scripting language can be deployed in this software. Moreover, it has the options to import 3D models for the robot as well as its own physics engine, making it feasible to simulate the Humanoid Robot in this software. It also has demo models of popular Humanoid Robots. In addition, it has an intuitive user interface making it simple to simulate our robot.

Arduino IDE provides features to compile and upload source code edited in a computer to ESP32 microcontroller. It is currently the most used and documented software for this purpose. So, it is feasible to be deployed in this project.

Python environment is used to train object detection models, so, it is obviously the most feasible programming language for this project.

Node.js a JavaScript runtime environment which is used for executing JavaScript code outside of the browser. It provides functionalities for server-side scripting. Its fast server processing with non-blocking event loop makes it suitable for real time transmission of data from ESP32. It is compatible with socket programming, so that a seamless and continuous transmission of data is ensured.

3.2.3 Economic Feasibility

The budget is affordable as seen from the budget table in Table 9-5. The hardware components are readily available in the market at a fair price. The software components are open source and freely available to download from the internet.

3.2.4 Schedule Feasibility

Employing the methodology of parallel work on different sectors of the project and combine them with rapid testing and deployment, the project is feasibly expected to be completed within the schedule.

4. SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 System Block Diagram

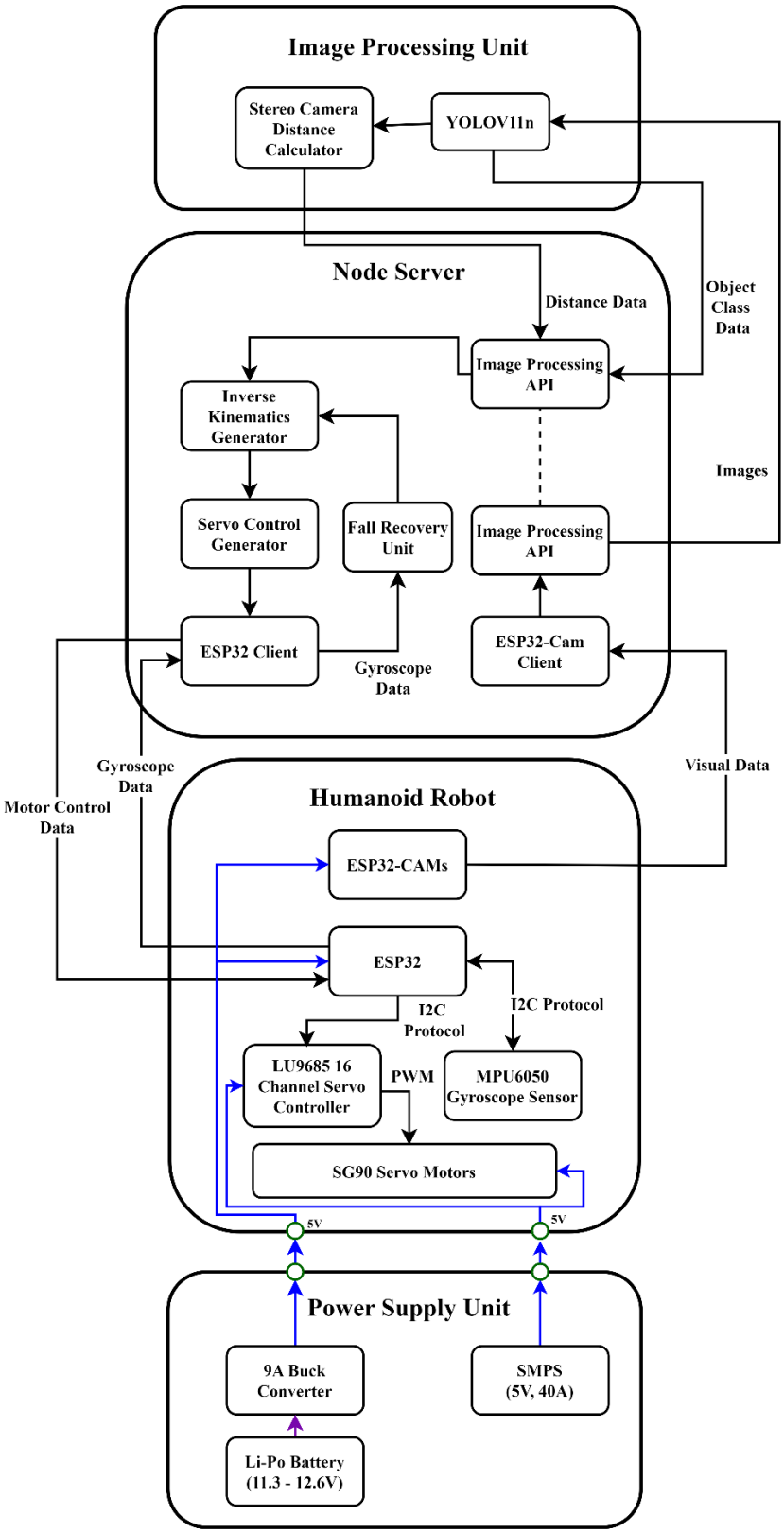


Figure 4-1: Block Diagram of the Proposed System

4.2 System Components

4.2.1 Humanoid Robot

The Humanoid Robot is a robotic system consisting of ESP32-CAMs for capturing visuals, MG996R servo motors for moving joints of the robot and LU9685 16 Channel Servo Controller for driving the servo motors. Moreover, an ESP32 microcontroller controls the robot's arms and legs based on the generated commands received from the server in the computer. A MPU6050 gyroscope sensor is also present for detecting the balance state. The body of the Humanoid Robot is 3D printed.

4.2.2 Node Server

The Node server makes use of WebSockets to receive and transmit data from other units. It feeds the data received from the Image Processing Unit to the servo control generator which contains necessary data for the ESP32 to drive the servo motors.

Moreover, the Node Server also transmits the data from ESP32-CAM visuals received from the robot to the Image Processing Unit.

4.2.3 Image processing Unit

The image processing unit consists of a distance calculator which calculates the distance from the object to the robot. This unit employs the use of the YOLOv11 for object detection on the stream received from the stereo camera setup and calculating the distance.

4.2.4 Power Supply Unit

The power supply unit drives the power into the whole humanoid robot. It consists of a 3S Li-Po battery, 9A Buck Convertor and a 5V 40A Switch Mode Power Supply.

4.3 Working Principle

4.3.1 Humanoid Robot Design

An open-source design of a humanoid robot with 18-DOFs designed in Fusion 360, as shown in Figure 4-2 is used as a reference design and modified to achieve needed functionality.

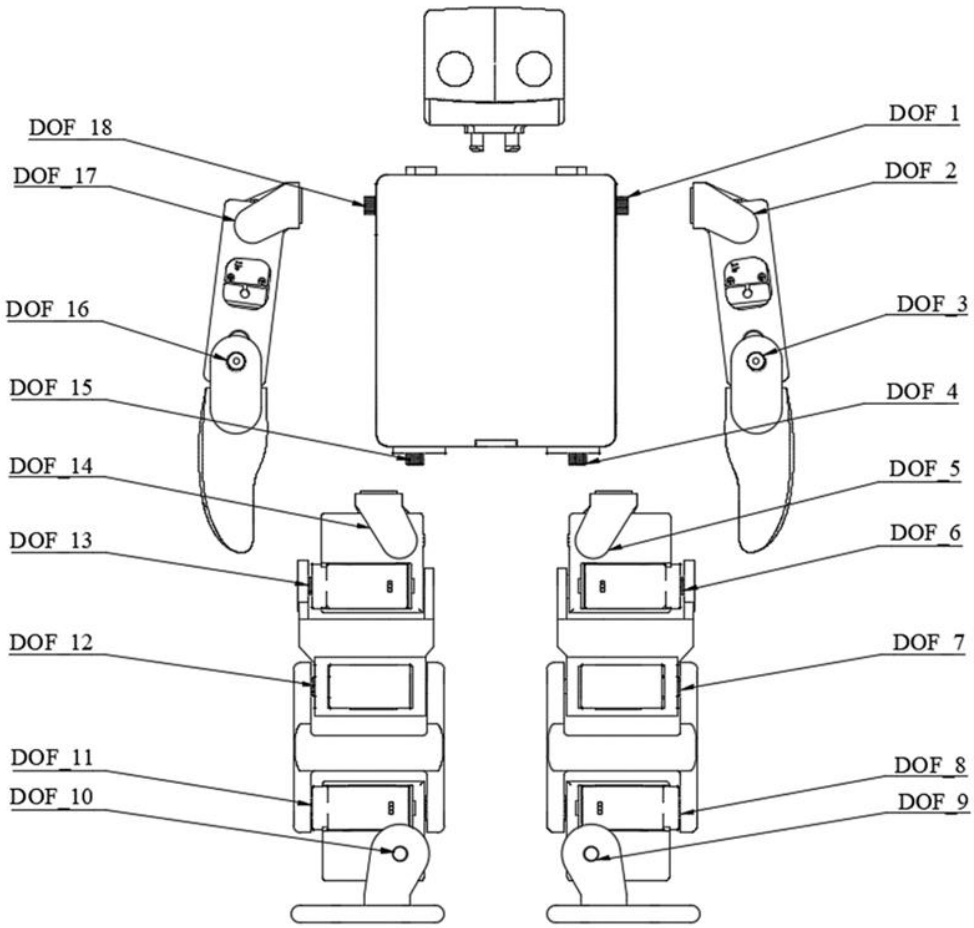


Figure 4-2: Reference Robot Design with 18-DOFs

Based on this reference, the robot is designed as seen in Figure 4-3 to achieve movements and construct a prototype of the final robot. The design consists of all needed joints for legs and hands are designed with all needed degrees of freedom (6 in each leg, 3 in each arm, 1 in neck); 19 DOFs in total. The designed model of the actual robot is much different than the reference design to accommodate different circuits and sensors. The designed robot is designed to be larger to use much powerful motors, microcontroller and 2 cameras with an additional DOF compared to the reference model

and changed link length to support different motions appropriate for object manipulation and balancing.

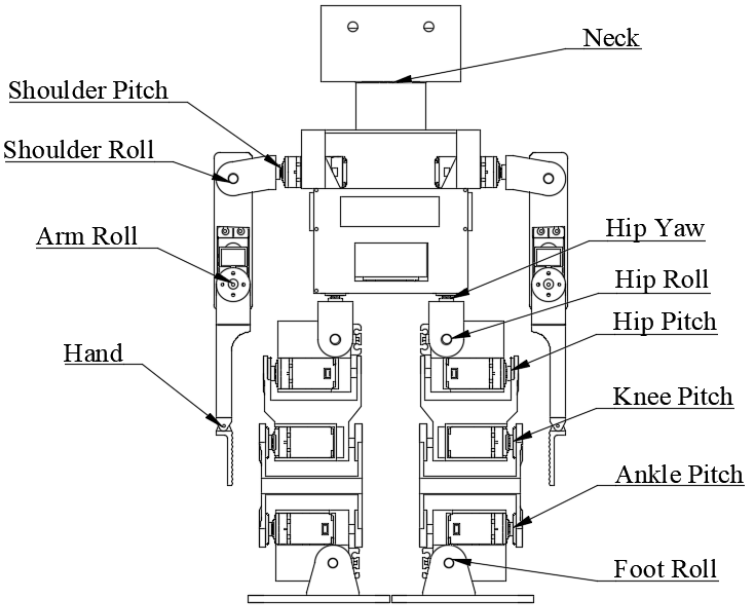


Figure 4-3: Robot Model's DOFs

Degree of Freedoms

The robot has a total of 19 degrees of freedom to achieve complex maneuvers to do specific tasks obtained from motors for each DOFs. The robot uses each motor for one degree of freedom in particular axes. Most of the DOFs are in X and Z axes with some in Y axis for Pitch, Roll and Yaw.

The overall robots can be divided into 4 main parts sections as: Head, Torso, Arms and Legs with their individual set of motors for a total of 19 DOFs.

- **Head:** 1 DOFs for Neck joint
- **Torso:** 0 DOFs (Circuits and Controllers)
- **Arms:** 3 DOFs in each arm for a total of 6 DOFs with 2-Shoulder, 1-Elbow
- **Legs:** 6 DOFs in each leg for a total of 12 DOFs with 3-Pelvis, 1-Knee, 2-Ankle

Arm Design

The robots' arm is designed to have 3 MG996R servo motors per arm for each major joint for its articulation. The major joints that are implemented in the arms are shoulder and elbow as shown in Figure 4-4.

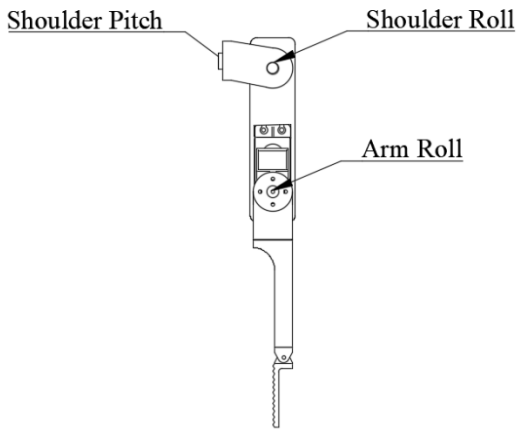


Figure 4-4: Design of Left Arm with 3-DOFs

The motor in the shoulder is used for X-axis rotation of the whole arm, Shoulder Roll is responsible for both Y-axis and Z-axis rotation depending on the arm state. Similarly, an Arm joint is used as an Elbow Joint, also an additional non dynamic joint is introduced at the wrist to adjust hand position for object manipulation. Also, the hand is created as a plain surface to maximize surface area while picking the object. Same configuration is repeated for the right arm as well for a total of 6-DOFs in Arms and a total of 6 Motors.

Leg Design

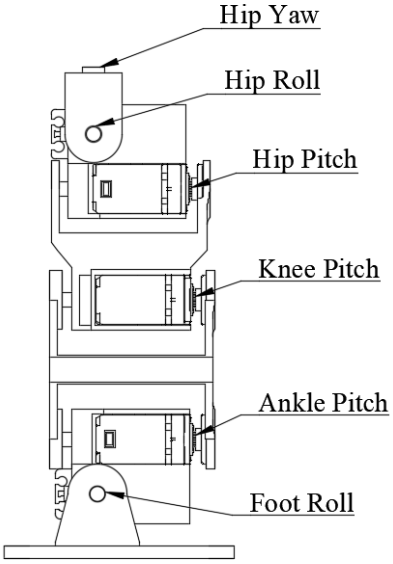


Figure 4-5: Design of Left Leg with 6-DOFs

The leg of the robot is designed to contain 6-DOFs per leg for each major joint in a leg for its movement as shown in Figure 4-5. The major joints involved: Hip, Knee and Ankle.

There are 3 joints (motors) in the pelvis (Hip Yaw, Hip Roll, Hip pitch) for movement of the whole leg in 3 different axes: X, Y and Z-axis. Then the Ankle and Feet allows the ankle which is divided into two sub-parts as Ankle Pitch and Foot Roll for their individual pitch control. Similarly, the Knee Pitch in the knee is responsible for knee pitch control. Same configuration is repeated for the right leg as well for a total of 12-DOFs in leg for robot's lower-limb functionality using 12 motors in total. The lower-limb of the robot is used in locomotion and is used for fall recovery.

Head and Torso Design

The head and torso of the robot is designed to house two ESP-32 Cams for stereo-vision, circuits, sensor, driver and battery along with some additional degree of freedoms for neck joint and waist joint which is not present in the reference design.

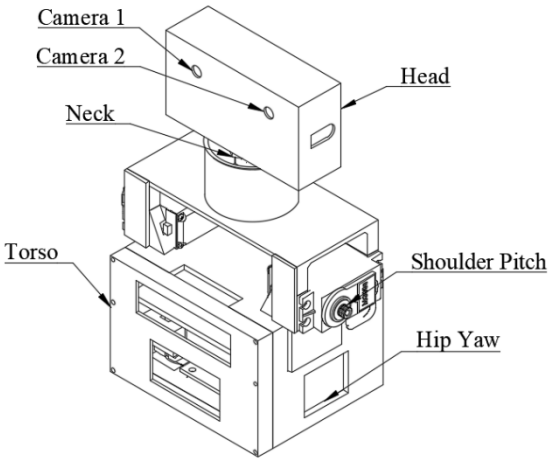


Figure 4-6: Design of Torso for the Robot

The torso of the robot houses motors for Hip Yaw and Shoulder Pitch for both legs and arms and it contains all the circuits, sensors, microcontrollers, motor driver and battery. The torso is divided into 3 layers. The bottom layer is designed to contain motors and has space for any circuit required. The Middle layer is designed to hold the battery and motor driver or any other necessary component and the top layer has enough space to accommodate two motors for shoulder pitch and microcontroller as shown in Figure 4-6.

The head as shown in Figure 4-6, is designed in such a way that it securely holds two cameras used for stereo vision with adequate pass throughs for wires without affecting the range of motion of the head. The head connects to the rest of the body using a micro servo and a semi cylindrical support around the neck keeps the head from extreme wobbles. Whole robot is designed and optimized for lower center of gravity and mass reduction.

All other parts; legs and arms are attached to the torso for a complete robot to mimic humanoid structure considering all major joints for achieving complex movements and dynamics.

4.3.2 Circuit Diagram

The circuit diagram can be seen on Figure 9-8. An ESP32-Wroom32 module microcontroller is used to provide overall control purposes. A Switch Mode Power

Supply that can supply 5V output voltage with maximum current up to 40A is used to power the servo motors and LU9685 servo controller. A Buck Converter that can provide 5V regulated output voltage with maximum current up to 9A is used to power two ESP32 CAM module and an ESP32 microcontroller as shown in Figure 9-8. Nineteen servo motors (MG996R) are used for moving different joints. Thirteen Servo motors are directly controlled through the ESP32 microcontroller. A LU9865 16-PWM servo controller is used to control remaining six servo motors. An accelerometer and gyroscope sensor (MPU6050) is used to track the orientation of the robot. Two ESP32-CAMs are used for tasks related to image processing.

Output terminal of 9A Buck converter is provided to the Vin pin of ESP32, and Vin pin of two ESP32-CAM. MPU6050, V_{cc} pin of LU9865 and nineteen servo motors are connected to the output terminal of the SMPS module. The control pin of thirteen servo motors is interfaced with the thirteen ADC pins of ESP32. The remaining six servo motors are interfaced with PWM, Vcc and GND pin of LU9865. The two I2C devices, namely LU9865 and MPU6050 are interfaced with ESP32 in daisy chaining fashion i.e. D21 and D22 pins of ESP32 are interfaced with the SDA and SCL pins of the LU9865 motor controller and the remaining SCL and SDA pins of LU9865 are connected with SCL and SDA pins of MPU6050.

4.3.3 MPU6050

MPU6050 is a motion-sensing device that uses a 3-axis accelerometer and a 3-axis gyroscope for measuring motion and orientation. Accelerometers are based on Micro Electro-Mechanical-System (MEMS) fabrication technology. It is a micro-machined structure built in a silicon wafer, as shown in Figure 4-7

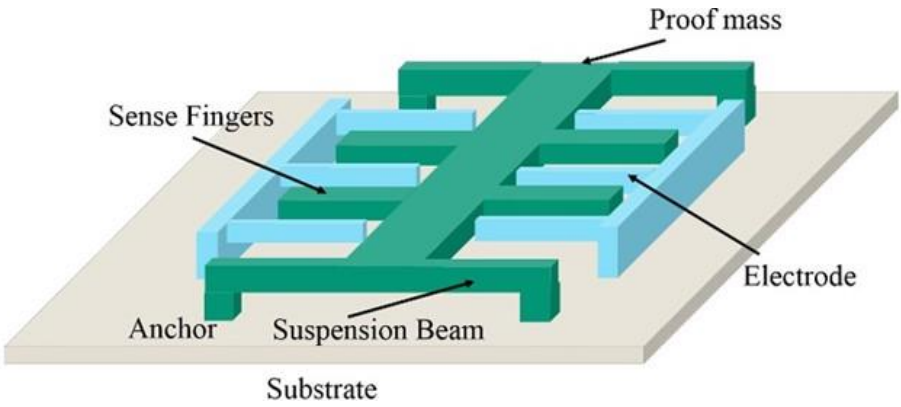


Figure 4-7: Structure of MPU6050

The Silicon wafer as seen in Figure 4-8 contains a polysilicon spring/suspension bridge, which allows the proof mass to deflect when accelerated along the X, Y, and/or Z axes.

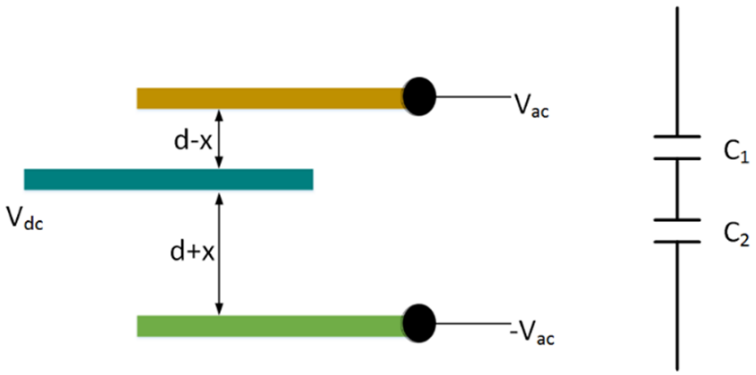


Figure 4-8: Structure of Wafer

As a result of the deflection of the proof mass, the capacitance between fixed electrodes and plates attached to the suspended structure changes as shown in the figure. This change in capacitance is proportional to the acceleration along that axis. The sensor processes this change in capacitance and converts it into an analog output voltage. In this project to find roll angle (angle of rotation along x-axis) and pitch angle (angle of rotation along y-axis) we decided to use the accelerometer sensor. The formula that gives the relation between acceleration and angle is shown in (4-1) and (4-2).

$$\boldsymbol{\theta = tan^{-1}\left(\frac{A_y}{\sqrt{{A_y}^2 + {A_z}^2}}\right)} \qquad \textbf{(4-1)}$$

$$\boldsymbol{\phi} = \boldsymbol{tan^{-1}}\left(\frac{\boldsymbol{A_x}}{\sqrt{\boldsymbol{A_x}^2 + \boldsymbol{A_z}^2}}\right) \tag{4-2}$$

Where, $\boldsymbol{\theta}$ = pitch angle

ϕ = roll angle

A_y = acceleration along Y-axis

A_x = acceleration along X-axis

A_z = acceleration along Z-axis

Gyroscope sensors are used to measure angular velocity. Gyroscope sensors are also based on Micro-Electro-Mechanical-System (MEMS). It consists of a proof mass consisting of four parts M1, M2, M3, and M4 as shown in the figure.

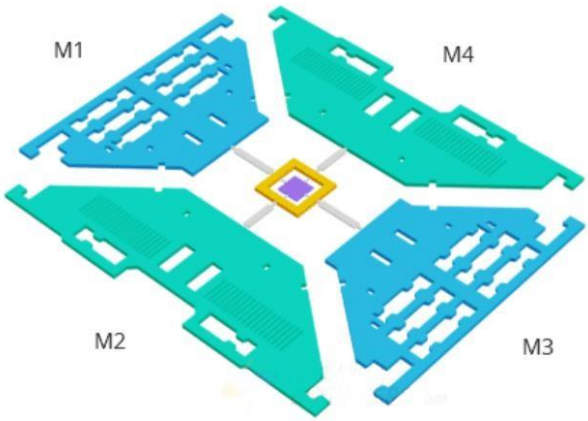


Figure 4-9: Micro Electro Mechanical System

Those proof masses are maintained in a continuous oscillating movement so that they can respond to the Coriolis effect. They simultaneously move inward and outward in the horizontal plane. When the structure starts to rotate, the Coriolis force starts to act on the proof masses, which causes the vibration to change from horizontal direction to vertical direction. Based on the axis along which the angular rotation is applied, the proof mass rotates in different directions. When the angular rotation is applied along the X-axis, masses M1 and M3 move up and down out of the horizontal plane due to the coriolis effect. When the angular rotation is applied along the Y-axis, masses M2 and M4 move up and down out of the horizontal plane due to the Coriolis

effect. When the angular rotation is applied along the Z-axis, masses M2 and M4 move horizontally in opposite directions due to the Coriolis effect. Due to the change in motion, Capacitance is changed. This change in capacitance is proportional to the angular velocity applied. Sensor processes this change in capacitance and converts it into analog output voltage. As Yaw refers to the angle of rotation along the Z-axis, when the sensor is rotated along the Z-axis the magnitude of the force acting along the Z-axis doesn't change, so the accelerometer cannot measure the yaw angle. To measure the yaw angle, a gyroscope sensor is used.

The MPU 6050 gyroscope sensor uses the I2C communication protocol which is a serial communication protocol that communicates using 2 lines, namely SCL and SDA. It works on a voltage range of 2.375V - 3.46V which is compatible with the operating voltage of ESP 32. SCL (Serial Clock Line) is controlled by the master and determines the timing of the data bits being communicated. The SDA (Serial Data Line) carries the actual data between the devices, the information on this line is only allowed to change when the signal on SCL is low.

4.3.4 LM7805 Voltage Regulator

LM7805 is a voltage regulator that belongs to the 78xx family, where “xx” in their name represents their output value. LM7805 is made up of different blocks as shown in Figure 4-10.

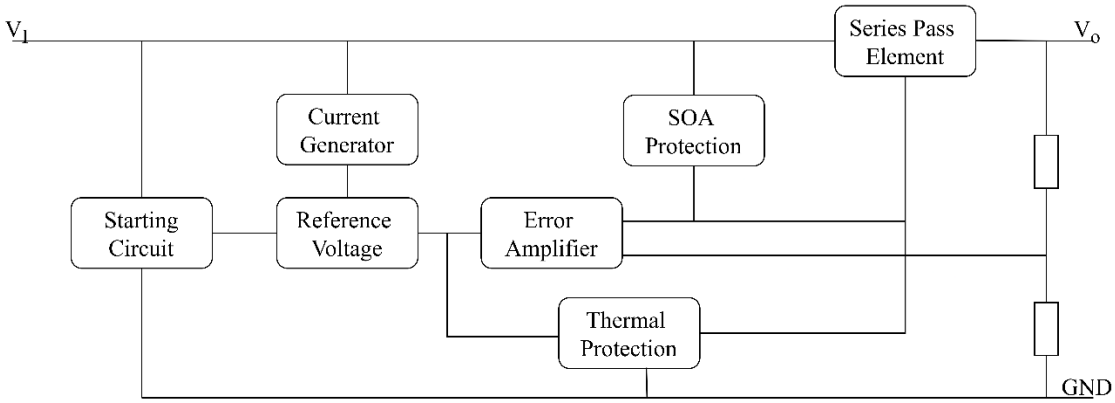


Figure 4-10: Block Diagram of LM7805 Voltage Regulator

The LM7805 consists of a series of interconnected components working in a proper sequence to generate a constant 5V output. When input voltage (7-12) Volt is provided

to the input terminal, the starting circuit initiates the operation of the internal components, ensuring proper start of the regulator. The current generator block then provides the constant bias currents required for the reference voltage block to generate a stable reference voltage. The error amplifier continuously compares the output voltage with the generated reference voltage and if any deviation between output voltage and reference voltage is detected, then produces an error signal. This error signal helps to adjust the series pass element, so that the desired 5V output is obtained.

The Safe Operating Area (SOA) Protection block helps the series pass element to operate within a safe limit and prevents excessive current or power dissipation. The thermal protection block monitors circuit temperature and takes the necessary actions to reduce output current or shut it down, if any overheating is detected.

4.3.5 9A Buck Converter

A 9A Buck converter is a DC-DC switching regulator that steps down a higher output voltage into lower voltage at 9 amperes current. The buck converter consists of inductor, capacitor, switch, diode and a load resistance. The switch controls the flow of power from input terminal to output terminal by turning off and on periodically. When the switch is open zero voltage appears at the output and when the switch is closed all supplied voltage appears at the output. The average output of a buck converter is controlled by either Pulse Width Modulation (PWM) or Pulse Frequency Modulation (PFM). In PWM technique, the overall switching time is kept constant while the on and off time of the switch is varied. Whereas, in PFM technique switch on and off time is kept constant while overall switching time is varied. On the basis of the inductor current mode of operation of the buck converter can be classified into two types. Namely, Continuous Conduction Mode (CCM) and Discontinuous Conduction Mode (DCM). In the Continuous Conduction Mode, the inductor current remains positive throughout the switching period whereas in Discontinuous Conduction Mode the inductor current remains zero for a certain period in a switching period. When the switch of the buck converter is turned on the diode becomes reverse biased to the applied voltage. Therefore, all of the applied current flows through the inductor i.e. $I_{dc} = I_L$. The inductor current is further divided into the capacitor and load current.

$$I_L = I_C + I_o \quad \textbf{(4-3)}$$

The inductor voltage is given by (4-4

$$V_L = V_{dc} - V_o \quad \textbf{(4-4)}$$

The turn on time of a switch for a given duty cycle (D) and switching period (T_s) is given by (4-5).

$$t_{on} = D \times T_s \quad \textbf{(4-5)}$$

The rippling current during turn on time can be calculated in following procedure.

We know, inductor and current relationship is given by (4-6)

$$V_L = L\frac{dI}{dt} \quad \textbf{(4-6)}$$

Now for a small time Δt_{on}, ΔI is given by (4-7).

$$\Delta I_{on} = \Delta t_{on}\frac{V_L}{L} \quad \textbf{(4-7)}$$

$$\Delta I_{on} = D \times T_s\frac{(V_s - V_o)}{L} \quad \textbf{(4-8)}$$

When the switch is turned off, inductor resists the change (according to Lenz Law) by changing polarity of the inductor which makes the diode forward biased. The voltage across the inductor is given by (4-9).

$$V_L = -V_o \quad \textbf{(4-9)}$$

The turn off time can be calculated as:

$$t_{off} = T_s - t_{on} \quad \textbf{(4-10)}$$

$$t_{off} = T_s - D \times T_s \tag{4-11}$$

$$t_{off} = T_s(1 - D) \tag{4-12}$$

Also, for small off time of Δt_{off}, current is given by, (4-13) or (4-14).

$$\Delta I_{off} = \Delta t_{off} \frac{V_L}{L} \tag{4-13}$$

$$\Delta I_{off} = T_s(1 - D) \frac{-V_o}{L} \tag{4-14}$$

To find the maximum and minimum we have, (4-15) and (4-16).

$$I_{Lmin} = I_L - \frac{|\Delta I_L|}{2} \tag{4-15}$$

$$I_{Lmax} = I_L + \frac{|\Delta I_L|}{2} \tag{4-16}$$

By substituting the value of either ΔI_{off} or ΔI_{on} we get,

$$I_{Lmin} = V_o \left(\frac{1}{R} - \frac{1 - D}{2Lf}\right) \tag{4-17}$$

Similarly,

$$I_{Lmax} = V_o \left(\frac{1}{R} + \frac{1 - D}{2Lf}\right) \tag{4-18}$$

Also, the net change of inductor current over a complete cycle is zero. So,

$$\Delta I_{on} + \Delta I_{off} = 0 \tag{4-19}$$

$$DT_s \frac{V_s - V_o}{L} + T_s(1 - D) \frac{-V_o}{L} = 0 \tag{4-20}$$

On simplifying (4-20),

$$V_o = DV_{in} \tag{4-21}$$

$$D = \frac{V_o}{V_{in}} \tag{4-22}$$

On using (4-22), the ripple current and required duty cycle for a buck converter can be determined. The overall circuit diagram of Buck Converter is given by Figure 4-11.

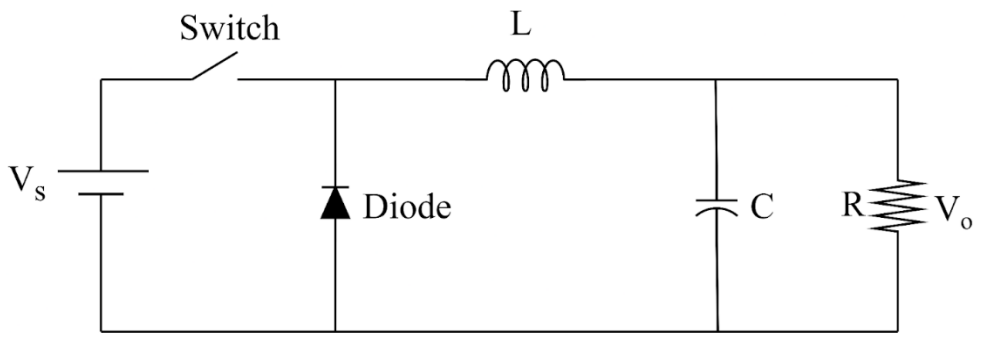


Figure 4-11: Circuit Diagram of Buck Converter

4.3.6 Servo Motor (SG90)

A servo motor is an electric motor that allows precise control of angular rotation. The servo motor consists of a PWM converter, comparator, DC motor, Gearbox, and position sensor. These components are interconnected with each other as shown in the figure to produce controlled output.

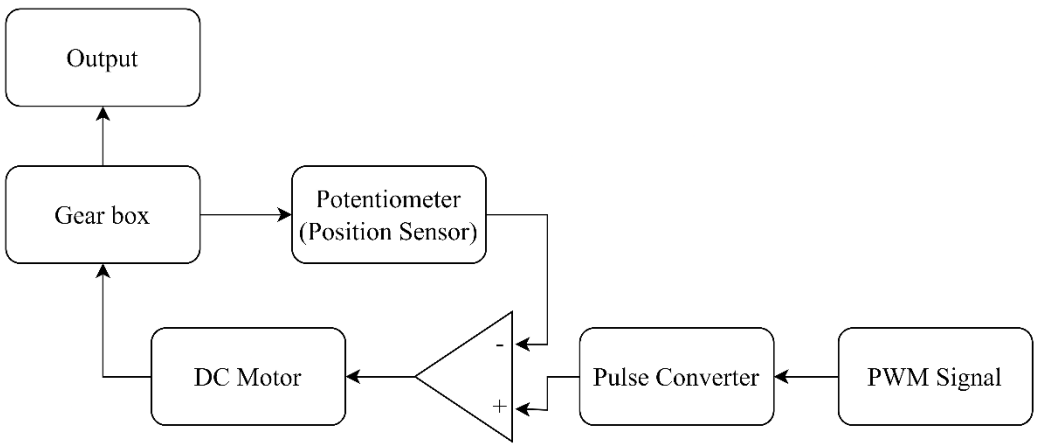


Figure 4-12: Block Diagram of SG90

PWM Signal

A PWM signal stands for Pulse Width Modulation signal. In Pulse Width Modulation technique, the signal's duty cycle is varied to represent the information. A microcontroller/motor controller changes the duty cycle of the digital signal. Due to this, the average voltage supplied to the motor changes, which results in controlled output. Generally, pulses of 20 milliseconds with 5 to 10 percent duty cycle are used to control rotation. A train of 50 pulses are sent at one second. A PWM signal with 5 percent duty cycle represents zero-degree rotation, a PWM signal with 10 percent duty cycle represents 180-degree rotation. The general relationship between the rotational angle and duty cycle is given by (4-23).

$$\boldsymbol{\theta = \frac{x-5}{5} \times 180} \tag{4-23}$$

Where, θ is the rotational angle and x is the corresponding duty cycle of the PWM signal.

PWM Converter

The PWM converter transforms the PWM signal into an analog voltage. This voltage level indicates the desired position to which the motor should rotate. A pulse with 0 percent duty cycle generates minimum output voltage whereas a pulse with 100 percent duty cycle generates a maximum output voltage.

Voltage Comparator

The output from the PWM converter is compared with the feedback voltage from the potentiometer, using a comparator and generates an error signal. The error signal helps the motor to adjust its position. If the error signal is positive, the motor's shaft needs to move in the direction that increases the feedback voltage whereas if the error signal is negative, the motor shaft needs to move in the opposite direction so that the feedback voltage is decreased.

DC motor

The error signal from the comparator is used to control the dc motor. It converts electrical energy into mechanical energy. The motor contains two main parts: a stator and a rotor for the conversion of energy. The stator is a stationary part which provides

a constant magnetic field. When a current is passed through the windings of the rotor, a magnetic field is generated. This generated magnetic field interacts with the stator's magnetic field, which causes the rotor to rotate. The amount of current passing through the rotor windings determines how much torque the rotor produces. The direction of the motor can be changed by changing the direction of the current through rotor windings. To control the direction of the motor a H-bridge circuit as shown in Figure 4-13 is used.

The H-bridge circuit consists of four switches namely S1, S2, S3, S4 and a dc motor. When the switches S1 and S4 are connected and S2 and S3 are kept open, the current flows through Vcc to the ground, which causes the motor to rotate in one direction. When the switches S2 and S3 are connected and S1 and S4 are kept open, then the current flows through Vcc to the ground, which causes the motor to rotate in another direction. When either S1 and S2 or S3 and S4 are connected, a short circuit occurs which may damage the circuit. When either S1 and S3 or S2 and S4 are connected, it causes the same potential difference across both terminals of the motor due to which no current flows through the rotor as a result, the motor stops. When all of the switches are kept open the motor consumes no power and rotates freely due inertia and stops naturally after some time.

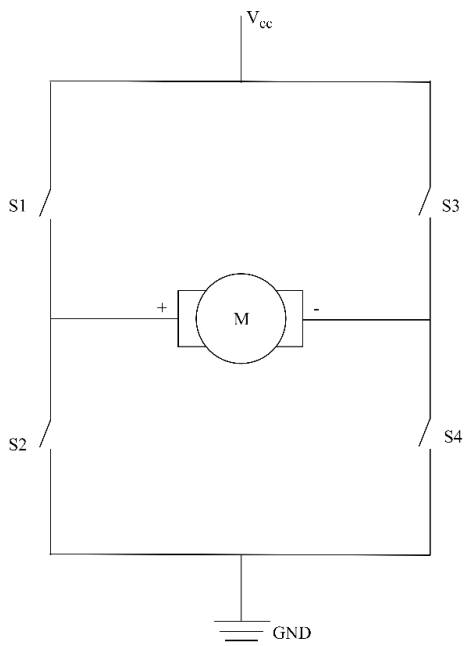


Figure 4-13: Schematic Circuit Diagram of H-Bridge

Gear box

The DC motor is connected to the gearbox to decrease the speed and increase the torque produced by the motor as the gear type and its ratio determines how much torque is multiplied and how much speed is reduced. Typically, smaller form-factor DC motors are connected with smaller gear setup with gear ratio of ~100:1 to ~150:1 to get a torque multiplication factor of ~100-150 times. The larger form-factor servo motors have larger gear box with metal gears having gear ratio of ~200:1 to ~250:1 or more to get torque multiplication factor of ~200-250 times or more depending on the requirement. Gear ratio is calculated from (4-24).

$$GearRatio = \frac{No.ofTeethonOutputGear}{No.ofTeethonInputGear} \quad \textbf{(4-24)}$$

4.3.7 LU9685-20CU 16-Channel Servo Controller

The LU9685-20CU Controller is an easy-to-use 16-channel servo controller that is similar to the PCA9685 (an LED Driver Chip). However, this module doesn't require any additional libraries to communicate with. By sending a data payload of a specific format using the I2C communication protocol, we can simply control any of the 16 motors' shafts to rotate to the required degrees of rotation.

This module uses a microcontroller from the STC8H family of microcontrollers. This series of microcontollers doesn't require an external crystal oscillator but instead uses an internal high-precision RC oscillator circuit to generate clock pulses. They are 8051 microcontrollers with high speed, low power consumption, and strong anti-interference. It is around 12 times faster than the 8051 microcontroller at the same operating frequency. They are single-clock per machine cycle (1T) microcontrollers, whereas the traditional 8051 microcontrollers were 12 clocks per machine cycle microcontrollers. This microcontroller is fully compatible with the traditional 8051 instruction set with an operating voltage of 1.9 to 5.5 volts. Since the resistance and capacitance values depend on the temperature, the RC clock is temperature-dependent. It has 12 channels (0 to 11) ultra-high-speed ADC, which supports 10-bit precision with a maximum speed of 500,000 ADCs per second. It consists of 8-channel advanced PWM timers that can be used as DACs.

The exact microcontroller used in this module is STC8H-1K16-36I-LOFP32. From the naming convention of this microcontroller family, it can be seen that this microcontroller has SRAM of size 1KB and Flash Memory of size 16KB, i.e., 1K16.

This module also supports the UART communication protocol, but for this project only the I2C communication protocol is used with Daisy Chaining fashion with the MPU6050 sensor. The module provides 5 open terminals, namely A0, A1, A2, A3, and A4, which can be used to change the address of the module for the I2C communication protocol. These terminals act as the binary values from 00000 to 11111. Each unbridged terminal represents 0, whereas each bridged terminal represents 1. So, by bridging the terminals, we can generate 2^5, i.e., 32 different addresses starting from (128 to 159 in decimal). This allows us to use multiple modules if necessary.

An external power supply can be provided via pin V+ of this module, which is directly connected to the power pin of the servo motors to drive them. The microcontroller thus only provides the PWM signal, not the power to the motors. We can configure the operating voltage of the microcontroller to our suitable voltage levels using the DVCC pin. This makes the module to be used with different microcontrollers that have different operating voltages (for example, 5V for Arduino Uno R3 and 3.3V for ESP32). In this project, since the main controller is the ESP32 microcontroller, the 3.3V from the ESP32 is connected to the DVCC of the module.

4.3.8 MG996R Servo Motor

MG996R is a metal gear servo motor, which can be programmed to move to in a specific position. The operating principle of the MG996R servo motor and SG90 servo motor is almost the same.

The plastic gear of the SG90 wears out after continuous use for long time. Also, the plastic gears are susceptible to gear stripping. With the repeated use the positional accuracy of the SG90 servo motor decreases. The metal gear MG996R solves these problems. The MG996R is nearly four times heavier than the SG90, so the overall weight of the humanoid robot increases on using it. So, some design consideration to be made. The dead bandwidth of the MG996R is twice less than the SG90 motor which infer that the positional accuracy of the humanoid robot's movement can be increased.

The high stall torque and operating speed of the MG996R servo motor ensures the smooth operation of the motor.

Table 4-1: Specifications of SG90 and MG996R Motors

	SG90	**MG996R**
Gear Material	Plastic gear	Metal gear
Weight (in gram)	14.7	55
Operating Voltage	4.8V-6V	4.8V-7.2V
Dead bandwidth	10μs	5μs
Stall Torque	1.2kgf.cm (4.8V) 1.6kgf.cm (6V)	8.5kgf.cm (4.8V) 10kgf.cm (6V)
Operating Speed	0.12sec/60degree (4.8V) 0.11sec/60degree (6V)	0.2sec/60degree (4.8V) 0.16sec/60degree (6V)

4.3.9 Battery

Lithium ion and Lithium Polymer battery are the most commonly used batteries for the high current application. Both Lithium Polymer (LiPo) and Lithium-ion batteries are the rechargeable energy storage device. Lithium polymer batteries use Polymer (gel like substance) as electrolyte, the lithium metal oxides are used as positive electrode (cathode) and carbon-based material are used as negative electrode (anode). Whereas, Lithium-ion batteries use the lithium metal oxides are used as positive electrode (cathode) and Graphite (C_6) or advanced silicon-graphite composites are used as negative electrode (anode). Lithium salt dissolved in organic solvent are used as electrolyte.

Both LiPo and L-ion batteries operates on the basis of movement of Lithium ions between the positive electrode and negative electrode. On charging Lithium ions from the positive electrode moves to the negative electrode and on discharging the Lithium ions from the negative electrode moves to the positive electrode. The polymer electrolyte allows the lithium ions to move freely during the charging and discharging phase.

The Lithium polymer battery has a very high power to weight ratio i.e. it can store great amount of energy for its weight and size. The gel like polymer electrolyte ensures relatively proper ionic conduction as compared to liquid electrolyte of Li-ion battery. Also, the gel like structure of electrolyte reduces the risk of leakage. Overall weight of the LiPo batteries is generally lighter than the Li-ion battery with comparable power rating. Therefore, LiPo battery is used in this project.

B3 Charger

The B3 charger is a that is specifically designed to charge 2-cell (7.4) and 3-cell (11.4) Lithium Polymer (LiPo) batteries. The B3 charger is a simple plug and play device that does not require any manual current adjustment. It has a LED indicator that indicates the charging status of the battery The B3 charger can be directly powered through an AC input of 110-240V. The B3 charger supports only charging mode. Although B3 charger are simple and easy to use, lack of balance charging and protection from overcurrent, temperature makes it unreliable. One cell of a 3-cell LiPo battery shown in Figure 9-11 was completely damaged by the B3 charger while charging.

IMAX B6 Charger

IMAX B6 charger is a smart battery charger that is widely used to charge and discharge LiPo batteries. It allows the user to control the charging speed and safety requirement based on the battery capacity. The built-in balancing function of B6 charger ensures that each cell in a multi-cell LiPo is equally, preventing overcharging. The IMAX B6 charger also supports discharging and storage modes, which are useful for maintaining the batteries when not used for the long period. It has an LCD display that provides real time voltage, current, charge capacity, and charging status for the precise monitoring. The B6 charger can be powered by either by using a 11-18V or an AC supply. An IMAX B6 charger is used in our project to charge the LiPo battery.

4.3.10 SMPS

SMPS refers to the Switch Mode Power Supply. It converts the electrical power using high frequency switching regulators. A Switching Mode Power Supply switches the transistor on and off to control the electrical power. At first, the SMPS rectifies and the

AC into high voltage DC. The power transistors then switch on and off at high frequency, which generates pulsating DC. The pulsating DC is then fed to high frequency transformer, which steps up or steps down the voltage as required. The transformed voltage is rectified again to using high speed diodes to convert it into DC. A feedback circuit continuously monitors the output voltage and adjust the switching mechanism accordingly. This operation mechanism ensures that the energy loss and heat generation is minimum. Feedback circuit of the SMPS helps to provide the consistent voltage. So, an SMPS with power rating 200W that can deliver an output voltage of 5V and current up to 40A is used in our project.

4.3.11 Data Transmission Mechanism

The whole robotic mechanism is controlled remotely through the Node.js server. The ESP32-CAM hosts the camera data locally in a network in which the server runs. The server can then access that image data and send it to the image processing model. The ESP32 controller that controls the servos is also connected to the server via WebSockets. Websockets is used as the communication protocol as it creates a persistent communication link for real-time communication. The ESP32 then receives the command from the server in real-time and performs actions accordingly to control the servos.

4.3.12 Object Detection Using YOLO

YOLO is the state-of-the-art model for the purpose of object detection. YOLOv8 was initially used for object detection and classification, whose object detection and classification whose blazingly fast performance made it highly suitable for the real-time applications like this humanoid robot project [15]. The architecture of YOLOv8 can be categorized into 3 main components: Backbone, Neck and Head. In simple words, the Backbone is responsible for extracting the features from the images, the Neck is responsible for feature fusion and the Head is responsible for making predictions of the bounding box and the class of the object.

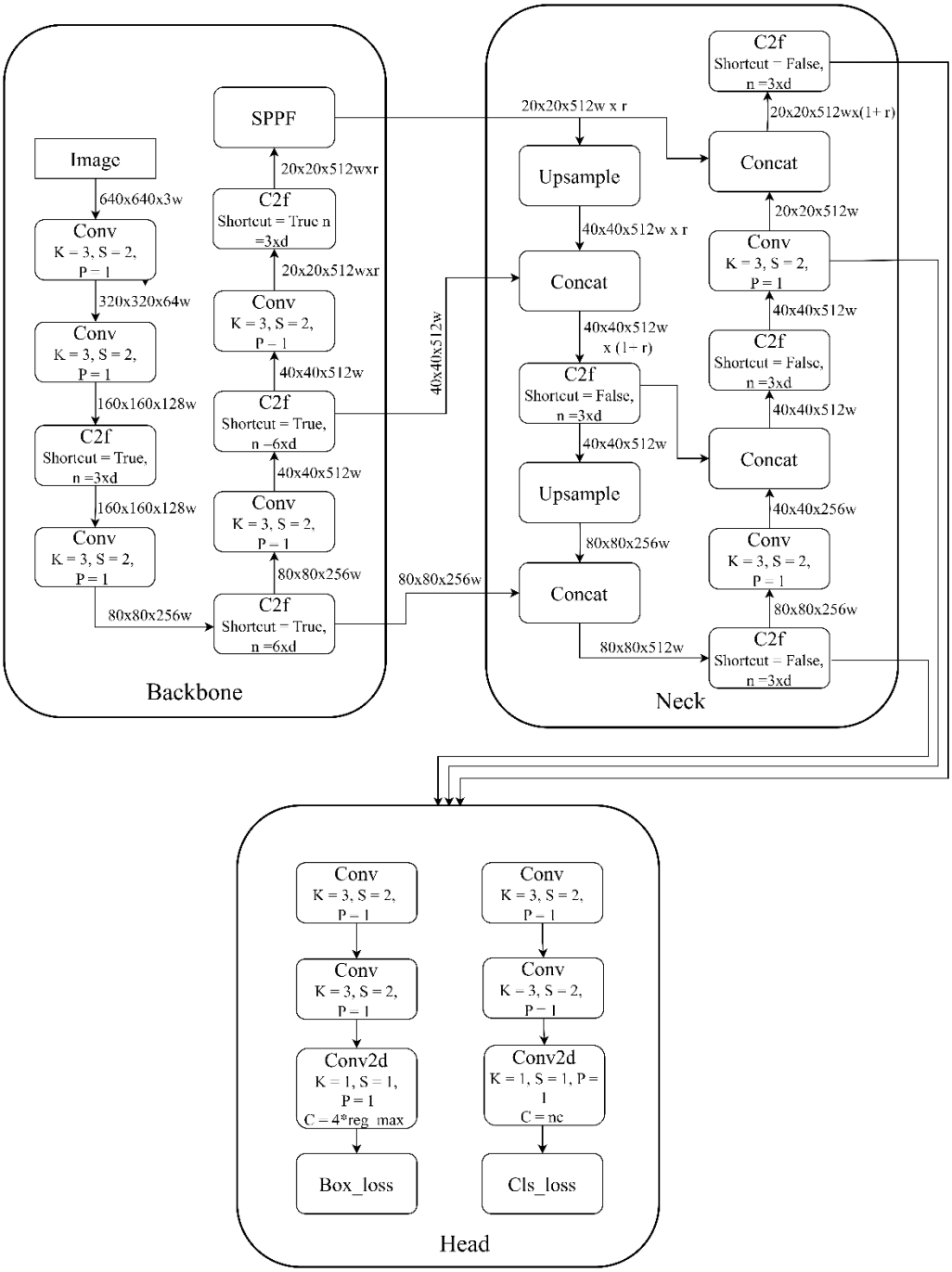


Figure 4-14: Architecture of YOLOv8

Backbone

The YOLOv8 employs the modified version CSPDarknet53 as its backbone, which consists of 53 convolution layers and employs a cross stage partial network for information flow between different layers. The different components involved in backbone are explained below:

Conv Block

The conv block plays an important role in extracting the features of the image. It consists of a 2D convolution layer which performs a 2-Dimensional convolution operation. Batch normalization layer which ensures that the extracted features are in the same scale and a SiLU activation function which introduces the non-linearity in the data.

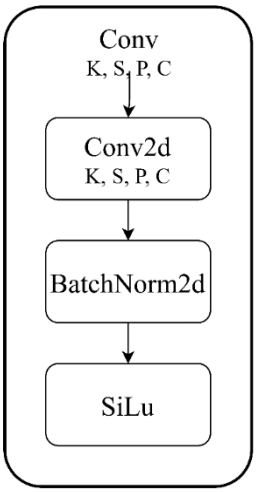


Figure 4-15: Convolution Block

Convolutional Layer

The convolutional layer is a layer that extracts the features from the given input image. The extracted features may range from a low-level feature like edges to a high-level feature like textures of an image. The convolution layer applies a set of kernels (K) with Strides(S) and padding(P) to the image for the feature extraction. A kernel is a small matrix that moves over the input data (image pixels), performs a dot product with the sub-region of input data and gets the output as the matrix of dot product. The kernel matrix is responsible for detecting the edges, textures and complex patterns present in the image. Stride(S) is the value that represents the number of columns or rows of pixel values that are to be slid after each multiplication of the sub-region of input data with kernel matrix. The stride allows to reduce the dimensionality of the input image. Padding(P) is defined as the number of pixels that are added around the input data, so that the kernel matrix to process the edge pixels. If an image of a I x I pixel is convolved with a kernel size of K x K with S stride value and P padding, then the size of the output data after convolution is given by:

$$O = \frac{I - K + 2P}{S} + 1 \tag{4-25}$$

Where, O is the size of the output square matrix.

Batch Normalization

Batch Normalization is a normalization technique which is used to standardize the given input data. The output feature maps from the conv2d layer is normalized using the batch normalization. The steps involved in the batch normalization are:

BATCH NORMALIZATION ALGORITHM	
1	***Calculate the batch mean*** $\boldsymbol{\mu_B}$ *for n number of values per feature map across entire batch from (4-26)*
2	***Calculate the batch variance*** $\boldsymbol{\sigma_B}$ *from equation (4-27).*
3	***Normalize*** *using the batch mean and batch standard deviation, using equation (4-28) where* ϵ *is a small constant introduced to prevent division by zero*
4	***Scale and shift*** *inputs by the factor* γ *and* β *respectively using equation (4-29)*

Batch mean and batch variance for a feature are given by (4-26) and (4-27) respectively.

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4-26}$$

$$\sigma_B{}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2 \tag{4-27}$$

Where, μB represents the batch mean for a feature, xi represents ith value in a feature map, n represents the number of values per feature map across entire batch, and σB represents the batch standard deviation.

Normalized data point is given by (4-28).

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B{}^2 + \epsilon}} \tag{4-28}$$

The final output y_i is given by equation (4-29), where γ and β are scaling and shifting factors respectively.

$$y_i = \gamma \hat{x}_i + \beta \tag{4-29}$$

Activation Function

The normalized output is passed through the Sigmoid Linear Unit (SiLU) activation function to introduce non-linearity in the output. If non-linearity is not introduced the model can only predict linear patterns while not the other complex patterns. The SiLU function is mathematically defined as

$$SiLU(x) = \frac{x}{1 + e^{-x}} \tag{4-30}$$

The nature of the plot between the output and the input to the SiLU is shown in Figure 4-16

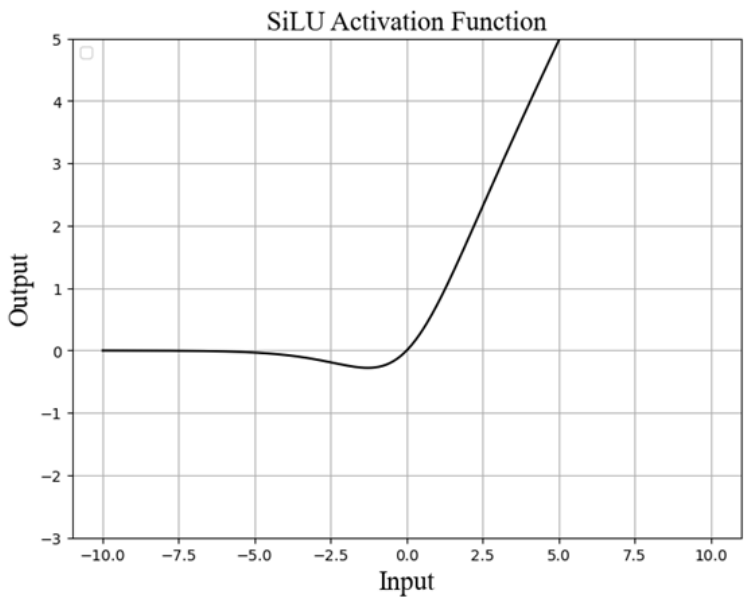


Figure 4-16: SiLU Activation Function Plot

From the graph it can be inferred that when x is positive, the value of SiLU ranges from 0 to x. When x is negative and in the range of 0 to 5, then the output of SiLU is a small negative number and when the x exceeds negative 5 then the output of the SiLU approaches to zero. As the value of the SiLU varies smoothly between 0 and x which makes accurate and robust feature extraction possible.

C2f Block

The C2f is an abbreviation of Convolution to Fully Connected. This block is designed to enhance the feature extraction and representation. The C2f block incorporates the fundamental principles of Cross Stage Partial Network (CSPnet) such as feature partitioning, cross stage connection, bottle neck layer and feature fusion.

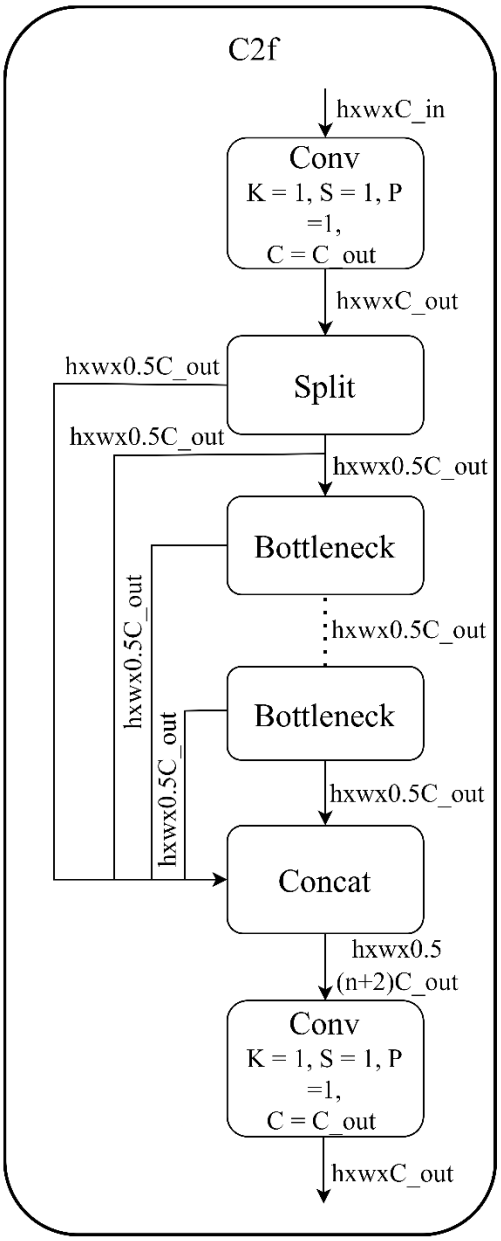


Figure 4-17: C2f Block

Conv block

The conv block performs the convolution operation with kernel of size 1x1 with Stride and padding of 1. The “C = Cout” in the conv block represents the number of desired

channels after convolution operation. When input feature map with ‘h’ height, ‘w’ width and ‘C_in’ number of channel is passed to this block then the output feature map with ‘h’ height, ‘w’ width and ‘C_out’ number of channels is obtained.

Split Block

The Split block divides the feature maps produced by the conv block into equal parts along the channel dimension. The feature map of size ‘h’ x ‘w’ x ‘c_out’ is partitioned into two feature maps with size ‘h’ x ‘w’ x ‘0.5c_out’. These two feature maps are then processed separately through different pathways i.e. one part is processed through bottleneck layer and another part bypasses the bottleneck layer.

Bottleneck Block

A bottleneck layer consists of series of convolution layers that reduce the dimensionality of feature map while preserving the spatial information. The bottleneck block has a parameter ‘shortcut’, which can take either ‘True’ or ‘False’ value. When Shortcut is set to ‘True’, the bottleneck block contains a connection that directly adds the input to its output. It means that the feature maps undergo several convolutions, then the output of the convolution is added with the input feature maps. This helps to retain the important information and lead to more robust and accurate feature representation.

When the Shortcut parameter is set to ‘False’, then the output feature maps obtained through the several convolutions of the input feature maps are not added i.e. output feature map are obtained simply through convolutions. It simplifies the network architecture and may be useful in a scenario where the residual connection doesn’t provide significant benefit.

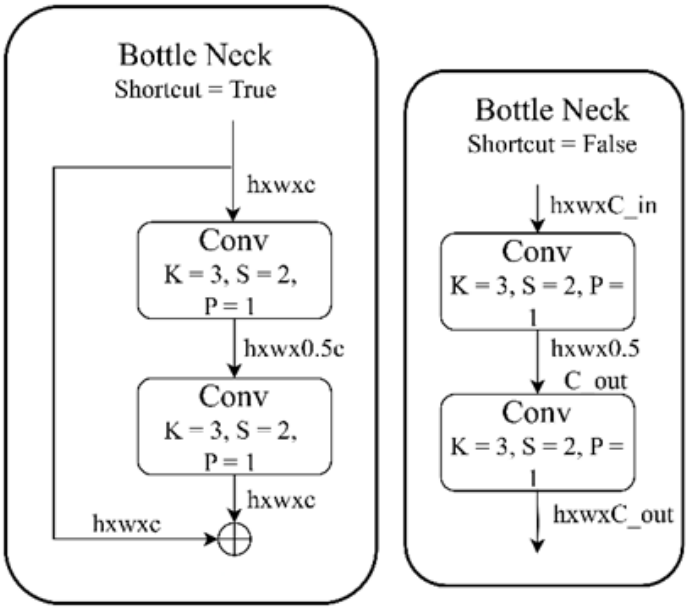


Figure 4-18: Bottleneck Block

Concat block

The concat block performs the concatenation of the feature maps from the multiple sources along the channel dimension. This block helps to retain the original feature along with the transformed features. When the two feature maps of size 'h'x'w'x'0.5c_out' from the split block and 'n' feature maps of size 'h'x'w'x'0.5c_out' form the bottleneck block are concatenated along the channel then a output feature map of a size 'h'x'w'x'0.5(n+2)c_out' is obtained.

SPPF block

The SPPF stands for Spatial Pyramid Pool Fast with Fuse, which is placed at the end of the backbone layer. It is designed to aggregate and process information from input feature maps across multiple scales. It consists of convolution block, MaxPool2d block, concatenation block.

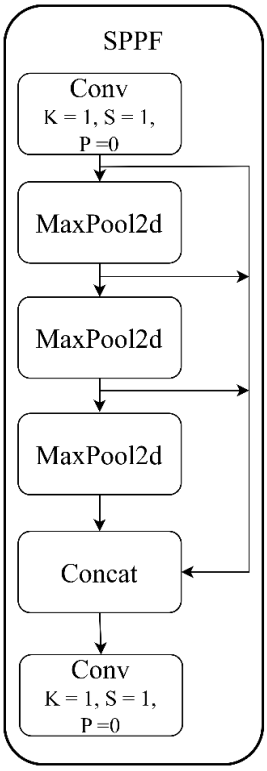


Figure 4-19: SPPF Block

MaxPool2d

The MaxPool2d block is used to down-sample feature maps while retaining the most important features. Each Maxpool2d consists of a different pooling size which determines the dimension of the window used for pooling. The MaxPool2d operation involves dividing the input feature map into non overlapping sub-regions and taking the maximum value within each sub-region. This pooling operation helps in extracting the most prominent features within each region. The output from the each Maxpool2d layer is then concatenated to obtain a rich and diverse set of feature maps.

Neck

The neck in the YOLOv8 is an essential part that improves detection performance by combining features extracted from the backbone. It consists of series of concatenation, up sampling, C2f (Convolution to Fully connect) and Convolution layers. The neck receives the feature maps of different scales from the different stages of the backbone, typically from high, medium and low-level abstraction. The low-level feature maps are up sampled and merged with the medium level feature maps. This feature map is again up sampled and merged with the high-level feature maps. The obtained fused feature

map is then used for object detection in Head. The fused feature map is again down sampled and concatenated with the output of the concatenation of the up sampled low-level feature map and mid-level feature map. This fused feature map is also used for object detection in Head. This fused feature map is again down sampled and concatenated with the low-level feature map; this output is also used for object detection in Head. This technique of feature fusion ensures that detailed low-level features and abstract high-level features are used for object detection.

Up sampling and Concatenation

The up-sampling operations increase the spatial resolution of the feature maps, where the resolution is lower due to previous pooling or striding operations. The up sampled feature maps are then concatenated with higher-resolution feature maps. The concatenation operation combines the deeper, semantically richer features with the shallower, spatially richer features, enabling the network to consider both the finer details of texture and the more general contextual information at the same time.

Head

The head in YOLO is the part where the actual object detection is performed. It consists of two sets of convolution layers followed by conv2d layers. The convolution layers refine the feature maps before making a final prediction. The first set of convolutional layers has a kernel size of 3x3 with a stride of 1 and padding of 1. This operation helps capture spatial information from the input feature maps while maintaining the same spatial dimensions. Following the first set of convolutional layers, there is a Conv2d layer with kernel of size 1x1 with the stride of 1 and a padding of 1and number of output channel equal to 4*reg_max. This Conv2d layer performs a 1x1 convolution operation on the input feature maps. The factor of 4 corresponds to the four bounding box parameters (x, y, w, h). This Conv2d layer is responsible for predicting the bounding box coordinates and regression values associated with the detected objects.

Similarly, the second set of convolutional layers also has a kernel size of 3x3 with a stride of 1 and padding of 0 indicating a 3x3 spatial convolution operation. It is followed by another Conv2d layer with kernel size of 1x1 with a stride of 1 and padding of 1 and number of output channel equal to ‘nc’, where ‘nc’ represents the number of classes or categories that the network is trained to detect. This Conv2d layer performs

a 1x1 convolution operation and outputs a tensor with 'nc' channels, representing the predicted class probabilities for each bounding box.

Loss Calculation

Bbox Loss

The Bbox loss calculates the loss for the predicted bounding boxes during training. It ensures that the predicted coordinates are as close as possible to the ground truth. The Bbox loss should be minimized to predict the location and the size of the object accurately. Some of the most commoly used Bbox loss functions are L1 loss, L2 loss, IoU loss, GIoU, CIoU, etc.

L1 loss: The L1 loss measures the absolute difference between the coordinates of predicted bounding box and the ground truth. It is given by (4-31).

$$\boldsymbol{L1} = \sum_{i=1}^{n} |x_i - y_i| \tag{4-31}$$

Where x_i represents the predicted value and y_i represents the ground truth value.

L2 loss: The L2 loss measures the mean square error between the coordinates of the predicted bounding box and the ground truth. It is given by(4-32).

$$\boldsymbol{L2} = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{4-32}$$

IoU loss

Intersection over Union loss is a measure that measures the overlap between predicted bounding box and ground truth. The IoU between the two boxes is given by the ratio of the area of the overlap to the area of the union between the boxes. Mathematically, IoU loss is given by (4-33).

$$\boldsymbol{IoU\ loss} = 1 - \frac{\boldsymbol{Area\ of\ overlap}}{\boldsymbol{Area\ of\ Union}} \tag{4-33}$$

Where, Area of Overlap is the area where the predicted and ground truth bounding boxes intersect and Area of Union is the total area covered by both the predicted and ground truth bounding boxes.

GIoU loss

Global Intersection over Union is an extension of the IoU, it considers the distance between the predicted and ground truth box. It is given by (4-34).

$$\boldsymbol{GIoU\ loss = 1 - IoU + \frac{C - (AUB)}{C}} \tag{4-34}$$

Where, A is the area of the predicted bounding box, B is the area of the ground truth box, C is the smallest box enclosing both A and B and IoU is the intersection over Union between A and B.

CIoU: Complete Intersection over Union is also an extension of IoU, it improves the limitations of IoU by considering not only the overlap but also the distance between bounding box centers and the aspect ratio of the bounding boxes. It is given by (4-35).

$$\boldsymbol{CIoU\ loss = 1 - IoU + \frac{d^2}{c^2} + \alpha\vartheta} \tag{4-35}$$

Where, d is the Euclidean distance between the centers of predicted bounding box and ground truth box, c is diagonal length of the box enclosing the both predicted and ground truth boxes, ϑ is a measure of the discrepancy between the aspect ratios of the predicted and ground truth bounding boxes and α is a trade-off parameter to balance the aspect ratio term.

Cls loss

The Cls loss refers to the classification loss, which measures how well the model classifies the object within the detected bounding boxes. The Cls loss should be minimized to predict object's class accurately. Some of the widely used class loss functions are cross entropy and focal loss.

Cross entropy

Cross entropy loss also known as the log loss measures the difference between the predicted probability distribution and the ground truth distribution. The cross-entropy loss is given by:

$$\boldsymbol{Cross\ entropy} = -\sum_{i=1}^{N} \boldsymbol{y_i \log\left(P_i\right)} \tag{4-36}$$

Where, N is the total number of classes, y_i is the ground truth label and P_i is the predicted probability of ith sample.

Focal Loss:

The focal loss is designed to address the imbalanced class problem. It focuses on hard-to-classify examples by down-weighting the loss for well-classified examples. The focal loss is given by:

$$\boldsymbol{Focal\ loss} = -\sum_{i=1}^{N} \boldsymbol{\alpha_i (1 - P_i)^{\gamma}\, y_i \log\left(P_i\right)} \tag{4-37}$$

Where, N is the total number of classes, y_i is the ground truth label and P_i is the predicted probability of ith sample, α_i is weighting factor for i^{th} class, γ is a focusing parameter that adjusts the rate at which easy examples are down-weighted.

4.3.13 YOLOv11

YOLOv11 state of art model was used instead of the YOLOv8 for the object detection. YOLOv11 introduces architectural changes to optimize processing and feature extraction. The Floating-Point Operation Per second (FLOPs) that represents the number of operations required to process a single batch of inputs through the model. It was found that the FLOPs value of YOLOv8n was 8.7 whereas for the YOLOv11n it was 6.5. The reduction in the FLOPS values infers that the YOLOv11 is computationally superior than YOLOv8. YOLOv11 introduced several architectural changes in backbone and neck layer which are discussed below:

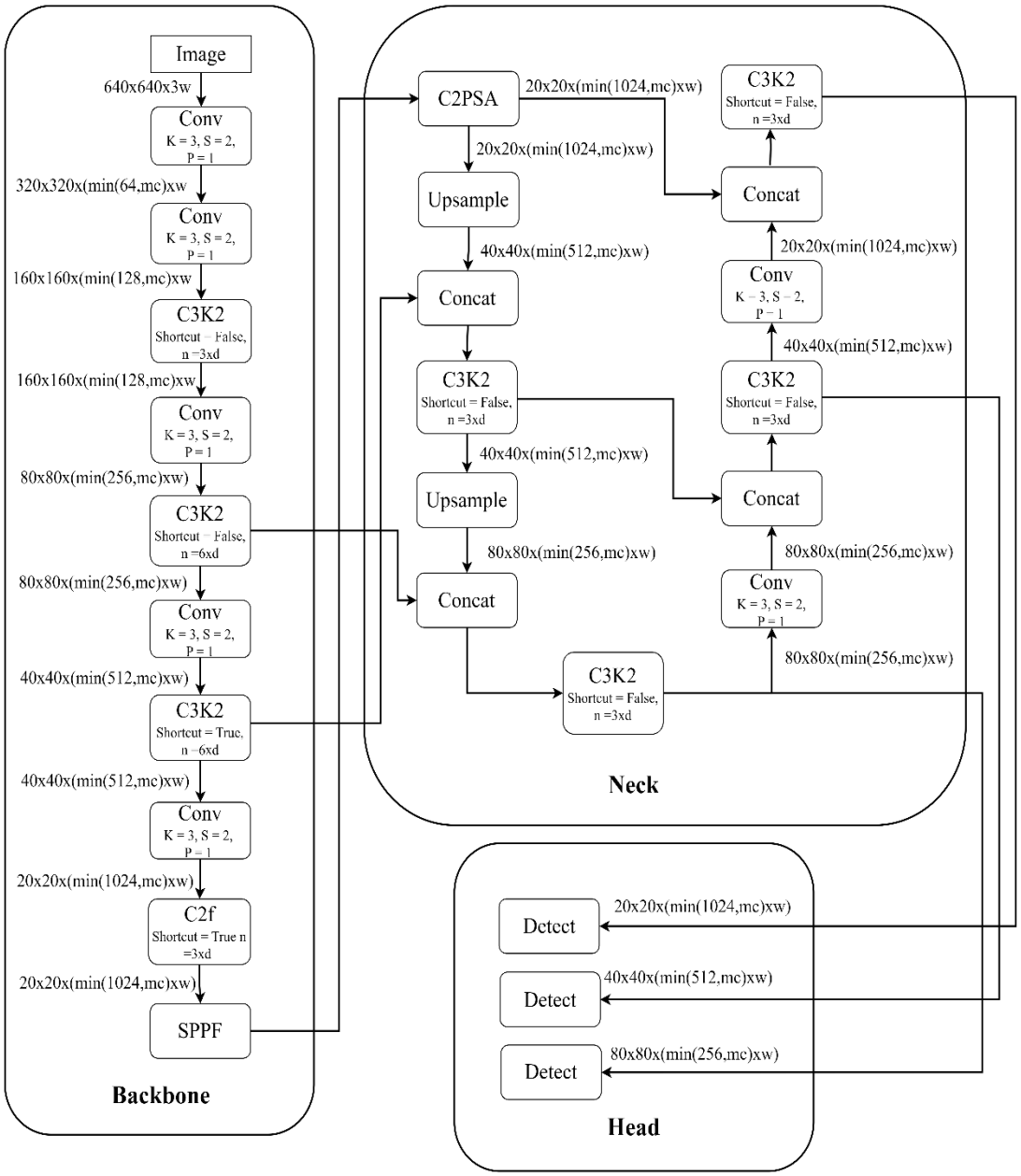


Figure 4-20: Architecture of YOLOv11

C3K2 Block

YOLOv11 replaced the C2f block that was used in the YOLOv8. This block is used for the feature extraction at different stage of the backbone. The C3K2 block uses C3K block to process the information. The C3K2 block has two convolutions block one at the start and another at the end. These series of blocks maintain a balance between the speed and the accuracy. The one and only difference between the C2f block and C3K block is that C2f block uses split block to split the feature maps into different size to process them separately. Whereas, no split block is used in C3k block.

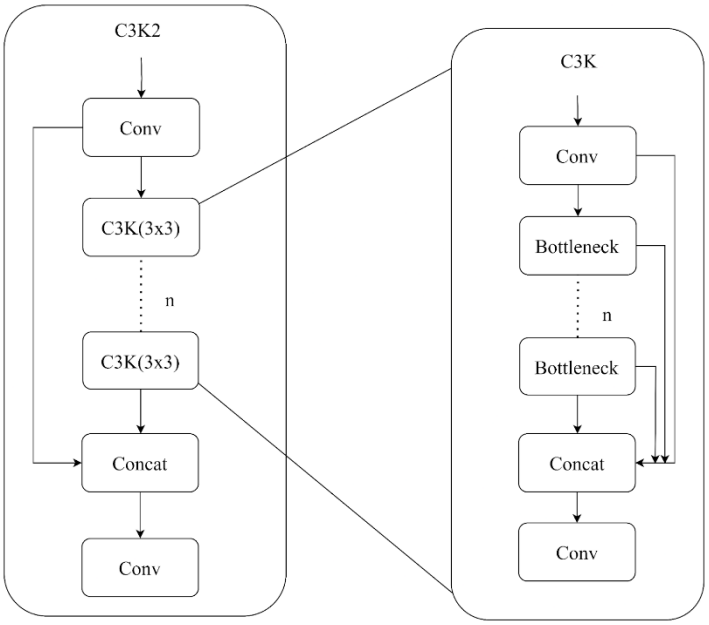


Figure 4-21: C3K2 Block

When an image enters into the C3K2 block. It is replicated into the two copies along the channel dimension. This splitting ensures that some of the raw information is preserved. The other half of the image split along the channel dimension is passed through the C3k block. When the feature maps are passed through the 'n' number of C3k a richer and deeper feature maps is obtained which are concatenated and convoluted to produce a cohesive feature representation that is ready to be passed to the subsequent layer.

C3k block uses different kernel sizes in its bottleneck layer for the proper feature representation. The 'n' number of repeated bottleneck layer deepen the feature representation which helps the model to learn patterns. The concat block then merges the features processed by the 'n' bottleneck block and the partial path that skipped the bottleneck. The output from the concat block is again passed through a convolution layer to mix the concatenated feature maps and produce a cohesive feature representation.

4.3.14 Estimating Distance via Stereo Camera

In absence of a depth camera, a stereo camera setup is required to calculate the distance the robot needs to move to reach a certain destination. A single camera setup cannot be

used to calculate the distance because it lacks a way to compare the distance in real world with the distance in the image pixels.

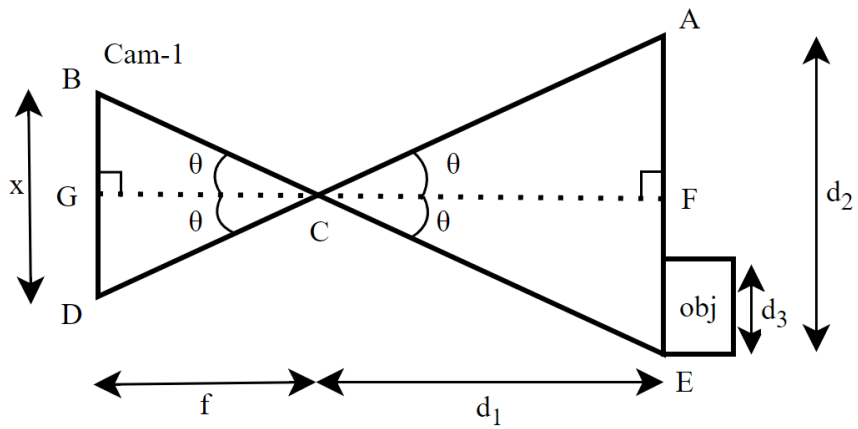


Figure 4-22: Setup to Calculate Distance using Single Camera

Let C be the camera lens that projects the real-world image plane AE to the digital image plane BD. The plane AE contains the front surface of the object whose distance is to be found relative to the camera. The distance of the object from digital image plane is given by:

$$\boldsymbol{d = d_1 + f} \tag{4-38}$$

From similar triangles ACF and BCG,

$$\frac{\left(\frac{x}{2}\right)}{\left(\frac{d_2}{2}\right)} = \frac{f}{d_1} \tag{4-39}$$

$$\boldsymbol{Or,} \qquad \boldsymbol{d_1 = \frac{f \times d_2}{x}} \tag{4-40}$$

Moreover, from the property of right-angled triangle,

$$\boldsymbol{d_1 = \frac{d_2}{2tan\theta}} \tag{4-41}$$

If the width of the object is p and the width of the field of view of the camera is N in pixels, the ratio of the real-world object width to the real-world image width is equal to the ratio of captured object width to the image width in pixels.

$$\frac{d_3}{d_2} = \frac{p}{N} \tag{4-42}$$

From equation (4-38) and (4-41),

$$d = f + \frac{Nd_3}{2p\tan\theta} \tag{4-43}$$

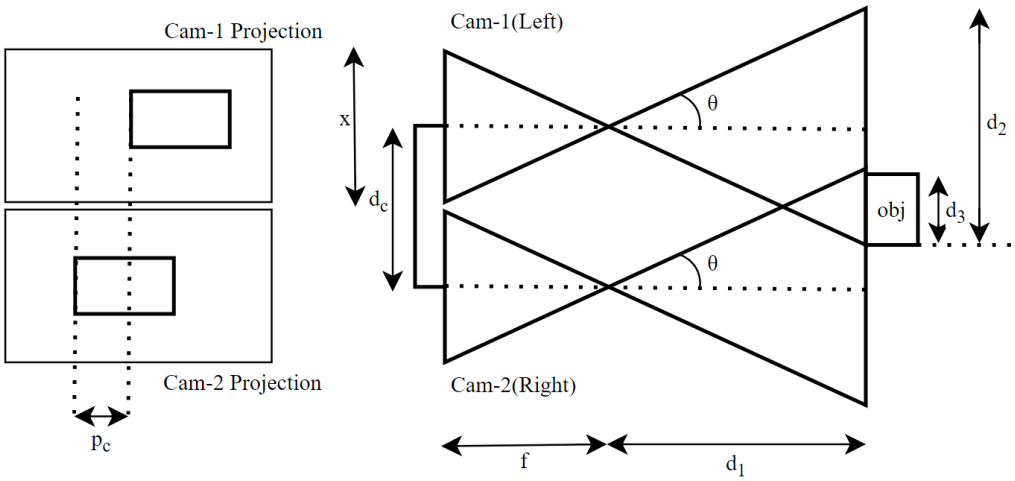


Figure 4-23: Setup to Calculate Distance using Stereo Camera

The distance d_3 is not comparable with pixel width p. So, equation (4-43) is not enough to calculate the distance. To overcome this problem, the principle of stereo camera is used as shown in Figure 4-23. For a parallel two camera setup, the objects in the image differ by some horizontal distance p_c. In order to calculate accurate distance, the perspective difference of the image captured by both cameras need to be taken into account.

If the cameras are separated by distance d_c, the shift in pixels of object in image corresponds to p_c. Now, these values are directly comparable as the shift in pixel is

relative to the distance of separation of two cameras. Then the equation for distance between stereo camera setup and the object can be written as:

$$\boldsymbol{d = f + \frac{Nd_c}{2p_c tan\theta}} \tag{4-44}$$

In order to calculate the unknown parameters: angle θ and focal length *f*, the corresponding distance formulas can be solved by calibrating the distance using two known distances.

For first calibration:

$$\boldsymbol{d_1 = f + \frac{Nd_c}{2p_1 tan\theta}} \tag{4-45}$$

For second calibration:

$$\boldsymbol{d_2 = f + \frac{Nd_c}{2p_2 tan\theta}} \tag{4-46}$$

Where, d_1 and d_2 are known distances from camera respectively for calibration and p_1 and p_2 are the object's pixel shift captured on d_1 and d_2 distances respectively.

Subtracting (4-46) from (4-45), we get,

$$\boldsymbol{d_1 - d_2 = \frac{N.dc}{2tan\theta}.\left(\frac{1}{p_1} - \frac{1}{p_2}\right)} \tag{4-47}$$

$$\boldsymbol{tan\theta = \frac{N.dc}{2(d_1 - d_2)}.\left(\frac{1}{p_1} - \frac{1}{p_2}\right)} \tag{4-48}$$

Substituting all the values in (4-48) the value of tanθ is obtained. Now, putting the value of tanθ in (4-45) or (4-46), the value of focal length is obtained.

4.3.15 Bot Movement Planning

For the bot to move to the desired position and pick/drop the object, it has some predefined set of movements that are written once and abstracted as a function to be used from the upper layer to reduce the complexity. These set of movements include:

Step Movement

This function takes 2 arguments namely 'direction' and 'leg'. The 'direction' argument defines if the step is to be taken in forward or backward direction whereas the 'leg' argument defines whether the left or right leg is to be moved.

Side Movement

This function takes only one argument 'direction' which defines if the sideways movement should take place in left or right direction.

Rotation Movement

This function takes two arguments namely 'direction' and 'angle'. The argument 'direction' defines if the bot should rotate in clockwise or anti-clockwise direction and the 'angle' defines the angle in degrees the bot should rotate.

Bend and Pick/Drop Movement

This function is useful in picking/dropping the object. Then the kinematics to pick/drop the object is calculated and sent to the ESP32 to perform the desired action.

Fall Recovery Movement

This movement is solely responsible for making the bot stand up if it falls due to certain conditions. The fall is detected using the angle data from the MPU6050 Gyroscope sensor. This method takes any argument and contains a hardcoded procedure that makes the bot wake up following a predefined set of movements under certain fall conditions determined by the angle data from gyroscope sensor. Since the bot has less surface area sideways and more surface area in front and back portion, the fall generally results in the bot falling on its front and back portion due to stability reasons. In case if the bot falls sideways, some hand movements and leg movements are used to make the bot rest

on the front or back portion and the same set of movements is used to make the bot stand up again.

The kinematics required for the movements is calculated using the different kinematic functions as described in the kinematics section.

4.3.16 Control System

The control system for the humanoid robot consists of a combination of both closed-loop and open-loop control systems. The robot requires a closed-loop control system as the motions are not perfect and consistent. During the motion, the initial position of the robot plays a very crucial role in how the robot runs the whole motion. At the end of the motion, ideally the robot should return back to the standing initial position, but due to the mass shifts during the motion and the small wobbling issue of each motor in its each rotational position, the robot's position differs a bit from the actual initial position at the end of each motion (however the code is written to bring back to its initial position). As a result, the robot's walk and other motions are not always completely the same. And the motion is also dependent on the walking surface friction. So, the distance covered by the robot in motion cannot be exactly the same all the time. This induces the need for a closed-loop control system where the robot, after each/multiple motions, takes feedback from the camera and the distance estimation unit to locate the object that it is intended to pick.

The robot requires an open-loop control system due to some of the physical limitations of the robot. The robot has only one degree of freedom in its head that allows it to rotate in the z-axis only so that the robot can rotate its head and see the objects around it. The robot has no degree of freedom to bend its head up/down. As a result, when the robot approaches the object, the view of the object becomes smaller and smaller for the robot, and at a certain distance, the object is completely below the view range of the object. In that case, the robot has no idea where the object is, and the control system fails. So, an open-loop control system is setup so that after a certain distance, the robot takes one cycle of forward walking and then picks up the object without exact calculation. This open loop control system is accurate if the motions of the robot are stable and consistent enough.

Overview of Closed Loop Control System

In this control system, the camera unit and distance estimation unit combinedly act as the feedback element. The feedback element can provide feedback about the presence of the object in the robot's view range and also the distance at which the object is present. The distance estimation is quite accurate in the range of 10–60 cm. So, the closed-loop control system, as shown in Figure 4-24, is designed in such a way that the robot tries to bring the object within the accurate range of the robot.

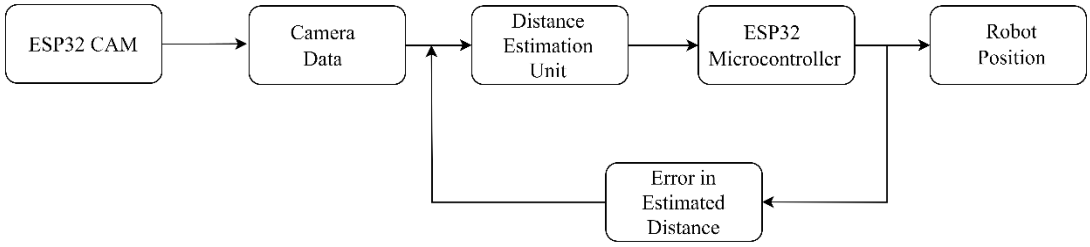


Figure 4-24: Control System with Feedback

First of all, the robot tries to bring the object into the frame by rotating itself towards the object. For that, the robot rotates its head and tries to find the object. When the object is found, the robot then records the head rotation angle and then rotates its body to that angle to align itself with the object. Then the object is again checked for presence. If the object is present, the rotation motion is stopped, and the robot checks if the objects are in the center of frame for both cameras. If not in center, the robot performs the side movement (left or right depending on the necessity to but object back to frame) to put the object in the center of frame. This ensures that the robot is aligned straightly towards the object. Then the distance estimation is performed to find the distance of the object. If the object is at a distance above 50 cm, the robot takes two steps in the forward direction and again estimates the distance and repeats the process. When the object comes within the distance of 50 cm the robot takes one step at a time and calculates the distance to see if it is on the right path to see the object and repeats. During this process, the robot also tries to put the object in the center of the frame by using sideways motions too. Once the object is found to be at a distance of below 15 cm, the robot then leaves the closed-loop control system and shifts to the open-loop control system.

Open Loop Control System

The robot enters the open loop control system when the object is detected at a distance less than 15 cm. It is fed to the system that the robot travels a distance of 5.6 cm on one forward motion. And the robot can pick the objects at about the distance of 10 cm far from the object. Considering these facts, the open loop control system, without any feedback from its motion, takes one step forward and then initiates the pickup motion.

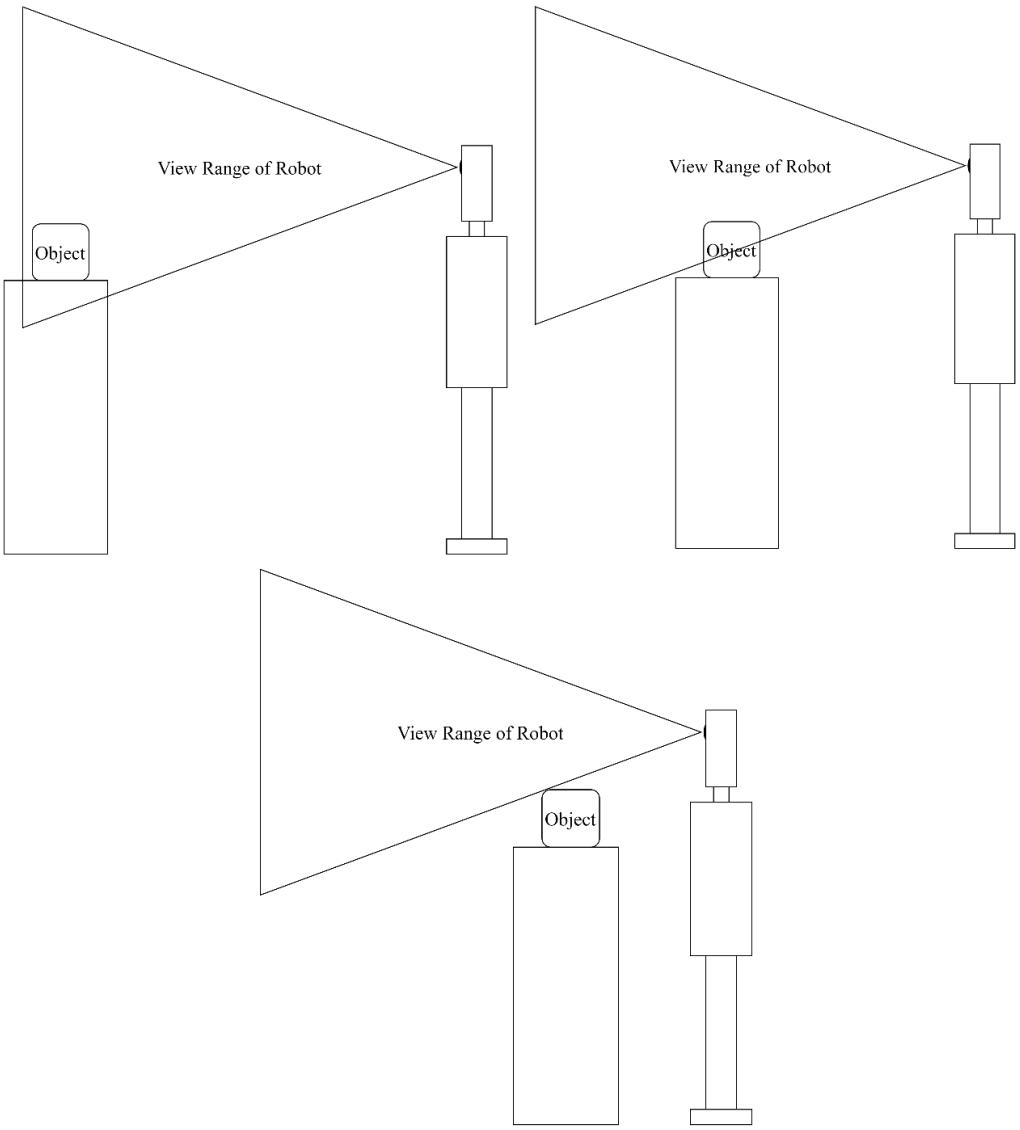


Figure 4-25: Three Different Scenarios of Object and View Range of Robot

How does the Robot determine if the object is in center or not?

The YOLOv11n model provides the bounding boxes coordinates of the object, if present in the image. Depending upon the object detection and bounding boxes

predicted, the object can be in following states in reference to the robot. The cases are described as follows:

CASE-I: Both cameras show the bounding boxes to be 0.

This is the case when the object is not in the frame of the object. Means that the object can be at the side of the object or behind the object. In this case, the robot determines the object is not in frame and object is not in center.

CASE-II: Right Camera detects the object but the left camera doesn't detect the object.

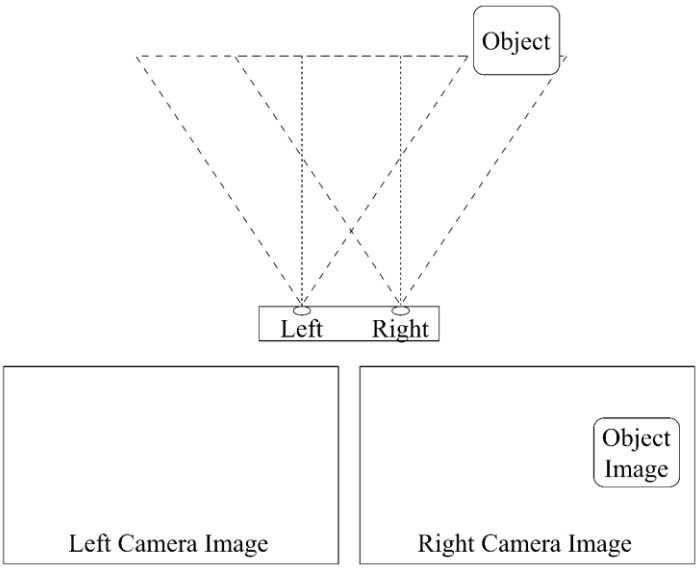


Figure 4-26: Detection by Right Camera Only

This means that the object is on the right side. The visual description of the scenario is shown in Figure 4-26. In this case the robot moves to the right side to place the object in the frame.

CASE-III: Left Camera detects the object but the right camera doesn't detect the object.

This means that the object is on the left side. The visual description of the scenario is shown in Figure 4-27 .In this case, the robot moves to the left side to place the object in the frame.

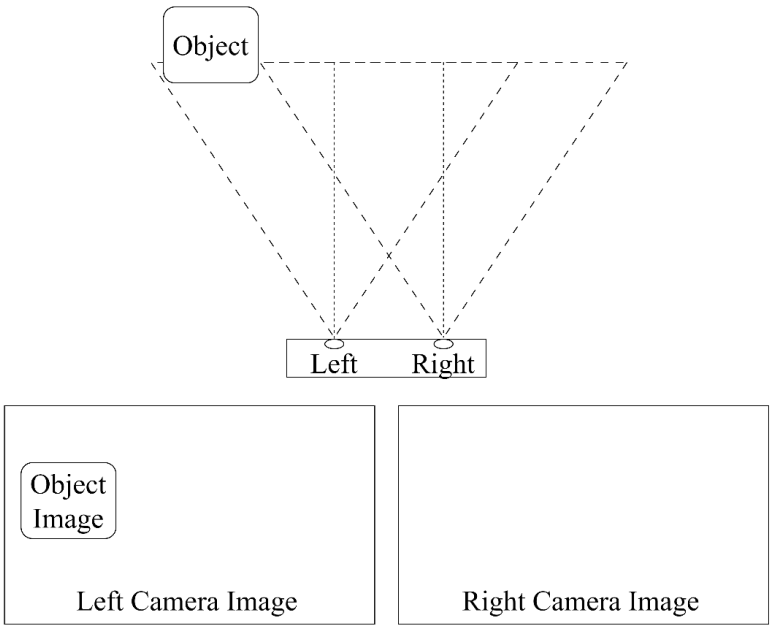


Figure 4-27: Detection by Left Camera Only

CASE-IV: Both cameras detect the object but the object is still on left side

This case is detected when the object difference of the object's center and right cam's center (in terms of pixel) is above a threshold. The camera's center is at 240 pixels (FC_R) as the width of camera frame is 480 pixels (Resolution being 480x320). The object's center is calculated from the bounding boxes coordinates. The bounding boxes coordinates gives the coordinates top-left corner (x1, y1) and bottom right corner (x2, y2). Then the object's center is calculated by taking mean of x1 and x2.

The threshold is set to a value of 130 pixels.

$$OC_R = \frac{x_1 + x_2}{2} \tag{4-49}$$

Then the distance between the camera center and the object center is calculated.

$$D_R = FC_R - OC_R \tag{4-50}$$

If the value of D_R is less than the threshold, the object is considered to be in center. Else, the object is considered to be in the Left side.

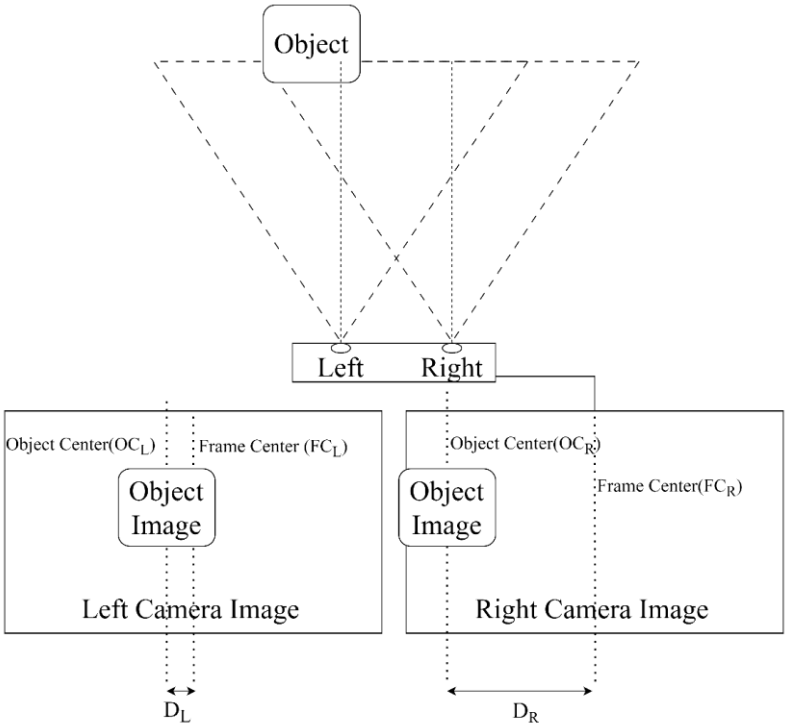


Figure 4-28: Object on Left Side

The value of threshold can be set to a smaller value in order to more accurately place the object in the center of the frame but that can lead to another problem. Due to the separation between the cameras, the object cannot naturally be at the center of both camera frames, when one is in center, other is a bit far from the center. If the threshold is set to a smaller value, this might lead to oscillation where centering one frame disrupts the centering of another frame and vice versa.

This leads to the failure of the control system leading to infinite oscillation of left and right sideways motion. The threshold of 130 pixels worked perfectly for this project.

CASE-V: Both cameras detect the object but the object is still on right side

The process of detecting this case is same as the CASE-IV. The difference in centers is calculated from (4-51).

$$D_L = OC_L - FC_L \tag{4-51}$$

This value if exceeds the threshold of 130 pixels, the object is considered to be at the right side. Else the object is considered to be in the left side.

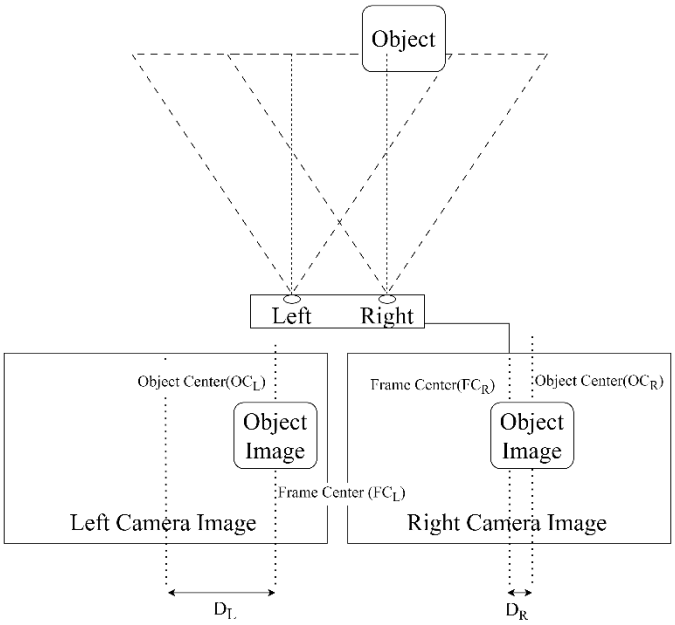


Figure 4-29: Object on Right Side

4.3.17 Kinematics

Kinematics is the study of motion without considering the torque and force causing it. It involves the study of both Forward and Inverse kinematics which is crucial for smooth control of the Leg's movement and to balance the robot while moving. Forward kinematics calculates the position and orientation of an end effector when the joint angles and length of links are known. Inverse kinematics calculates the joint angles when the position and orientation of an end effector are known.

The basic kinematic structure of joints and links is shown in Figure 4-30. The Denavit-Hartenberg method forms the basis for the calculation of Forward kinematics and manipulate the joints accordingly. It also provides the framework for formulating equations necessary for Inverse kinematics. Some parameters involved in Denavit-Hartenberg are shown in Table 4-2.

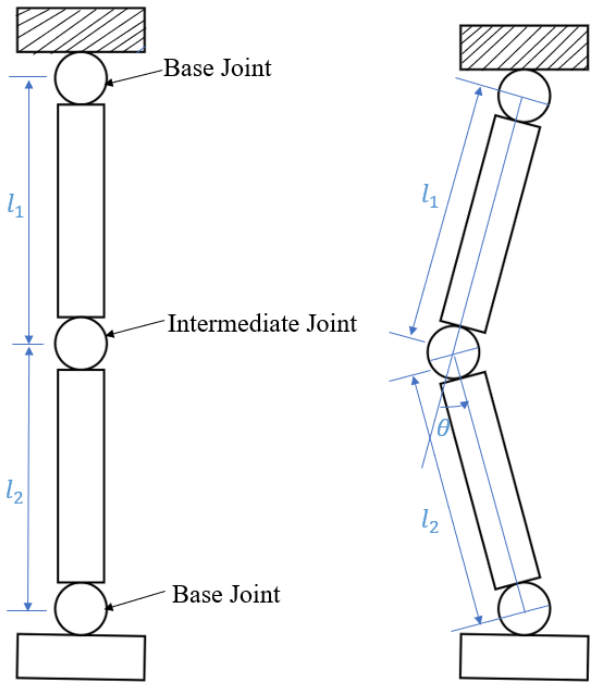


Figure 4-30: Basic Kinematic Structure

Based on the orientation and position of the joints, D-H parameter table is computed. From the D-H parameter table, a transformation matrix can be formulated:

$$\boldsymbol{H_n^{n-1}} = \begin{bmatrix} \boldsymbol{c(\theta_n)} & \boldsymbol{-s(\theta_n)c(\alpha_n)} & \boldsymbol{s(\theta_n)s(\alpha_n)} & \boldsymbol{r_n c(\theta_n)} \\ \boldsymbol{s(\theta_n)} & \boldsymbol{c(\theta_n)c(\alpha_n)} & \boldsymbol{-c(\theta_n)s(\alpha_n)} & \boldsymbol{r_n s(\theta_n)} \\ \boldsymbol{0} & \boldsymbol{s(\alpha_n)} & \boldsymbol{c(\alpha_n)} & \boldsymbol{d_n} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-52}$$

Table 4-2: Denavit-Hartenberg Parameters

Symbols	Description
θ_i	The rotation angle between X_i and X_{i-1} measured about Z_i.
α_i	The angle between lines along joints i-1 and i measured about Z_{i-1}.
r_i	Total displacement from $frame_{i-1}$ to $frame_i$ measured in x_i direction
d	Total displacement from $frame_{i-1}$ to $frame_i$ measured in Z_{i-1}
C_i	$cos(\theta_i)$
S_i	$sin(\theta_i)$
C_{ij}	$cos(\theta_i + \theta_j)$

S_{ij}	$sin(\theta_i + \theta_j)$
C_{ijk}	$cos(\theta_i + \theta_j + \theta_k)$
S_{ijk}	$sin(\theta_i + \theta_j + \theta_k)$

Then for each joint shown in the figure, a transformation matrix can be computed. The angle of rotation of the ith joint and the length of the link act as inputs to compute the transformation matrix of the ith joint. The overall transformation matrix for end effector relative to the base frame is computed as:

$$\boldsymbol{H_4^0 = H_1^0 \times H_2^1 \times H_3^2 \times H_4^3} \tag{4-53}$$

Forward Kinematics calculates the cartesian coordinates of end effector relative to the base joint based on current joint angle. Given the current joint angle FK gives cartesian coordinates (x, y, z) which is then used by inverse kinematic (IK) model to perform inverse operation of the FK to calculate joint angles from required cartesian coordinates.

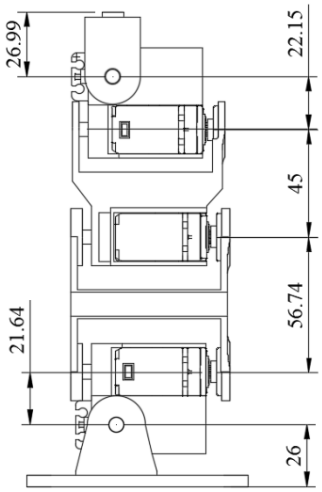


Figure 4-31: Inverse Kinematics Structure and Link Length of Right Leg

Each leg (left and right) forms their own IK structure with respective DH-parameters. The DH parameters of left and right legs can be expresses and below:

Table 4-3: DH-Parameters Configuration for Legs

Joint Pair	d [m]	θ [deg]	r [m]	α [deg]
Hip_Yaw - Hip	d_1	θ_1	0	90
Hip - Thigh	d_2	θ_2	a_2	-90
Thigh - Knee	0	θ_3	a_3	0
Knee - Ankle	0	θ_4	a_4	0
Ankle – Feet	d_5	θ_5	a_5	90
Feet - FootTip	d_6	θ_6	a_6	90

Similar DH-Parameters configuration is followed for both left and right legs as on leg is exact mirror of another leg.

The transformation of two consecutive joints for a link for above DH-Parameter are expressed as:

$$\boldsymbol{H_H^{HY}} = \begin{bmatrix} \boldsymbol{c(\theta_1)} & \boldsymbol{0} & \boldsymbol{s(\theta_1)} & \boldsymbol{0} \\ \boldsymbol{s(\theta_1)} & \boldsymbol{0} & \boldsymbol{-c(\theta_n)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{d_1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-54}$$

$$\boldsymbol{H_T^H} = \begin{bmatrix} \boldsymbol{c(\theta_2)} & \boldsymbol{0} & \boldsymbol{-s(\theta_2)} & \boldsymbol{a_2 C(\theta_2)} \\ \boldsymbol{s(\theta_2)} & \boldsymbol{0} & \boldsymbol{c(\theta_2)} & \boldsymbol{a_2 S(\theta_2)} \\ \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{c(0)} & \boldsymbol{d_2} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-55}$$

$$\boldsymbol{H_K^T} = \begin{bmatrix} \boldsymbol{c(\theta_3)} & \boldsymbol{-s(\theta_3)} & \boldsymbol{0} & \boldsymbol{a_3 C(\theta_3)} \\ \boldsymbol{s(\theta_3)} & \boldsymbol{c(\theta_3)} & \boldsymbol{0} & \boldsymbol{a_3 S(\theta_3)} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-56}$$

$$\boldsymbol{H_A^K} = \begin{bmatrix} \boldsymbol{c(\theta_4)} & \boldsymbol{-s(\theta_4)} & \boldsymbol{0} & \boldsymbol{a_4 C(\theta_4)} \\ \boldsymbol{s(\theta_4)} & \boldsymbol{c(\theta_4)} & \boldsymbol{0} & \boldsymbol{a_4 S(\theta_4)} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-57}$$

$$\boldsymbol{H_F^A} = \begin{bmatrix} \boldsymbol{c(\theta_5)} & \boldsymbol{0} & \boldsymbol{s(\theta_5)} & \boldsymbol{a_5 C(\theta_5)} \\ \boldsymbol{s(\theta_5)} & \boldsymbol{0} & \boldsymbol{-c(\theta_5)} & \boldsymbol{a_5 S(\theta_5)} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{d_5} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-58}$$

$$\boldsymbol{H_{FT}^{F}} = \begin{bmatrix} \boldsymbol{c(\theta_6)} & \boldsymbol{0} & \boldsymbol{s(\theta_6)} & \boldsymbol{a_6 C(\theta_6)} \\ \boldsymbol{s(\theta_6)} & \boldsymbol{0} & \boldsymbol{-c(\theta_6)} & \boldsymbol{a_6 S(\theta_6)} \\ \boldsymbol{0} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{d_6} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \tag{4-59}$$

The overall transformation matrix for Foot Tip relative to the Hip Yaw frame is obtained as:

$$\boldsymbol{H_{FT}^{HY}} = H_H^{HY} \times H_T^H \times H_K^T \times H_A^K \times H_F^A \times H_{FT}^F \tag{4-60}$$

By substituting the value of d_i, $\boldsymbol{\theta}_i$ and a_i of a particular link, the specific transformation matrices of the legs can be calculated. Multiplying these transformation matrices, Forward kinematics is implemented to find where the end-effector (FootTip) is, based on joint angles. After that, the required joint angles to reach a desired position and orientation is calculated using Inverse Kinematics [16]. It can also further be used for Jacobin analysis to relate joint velocities to end-effector velocities.

The calculated joint angles for each individual joint is mapped to the servo rotation to manipulate each joint of the humanoid robot to make the robot walk using it legs and pick defined objects using its arms.

4.3.18 Balancing the Robot

For balancing of robot, there are two qualitative things that can be done to improve the balance.

Lowering the Center of Gravity

The COG of non-regular body is calculated from (4-61).

$$\begin{Bmatrix} x_{COG} \\ y_{COG} \\ z_{COG} \end{Bmatrix} = \frac{1}{\sum_{i=1}^{n} m_i} \begin{Bmatrix} \sum_{i=1}^{n} m_i x_i \\ \sum_{i=1}^{n} m_i y_i \\ \sum_{i=1}^{n} m_i z_i \end{Bmatrix} \tag{4-61}$$

Where $x_{COG}, y_{COG}, z_{COG}$ are the positions of the mass center of the robot: and x_i, y_i, z_i are the positions of the mass center of the link I; m_i is the mass of the link i and n is the total number of the components of the robot.

As computing the COG of the complex body manually is cumbersome quantitative method is used to lower the COG. The COG of the robot is lowered by placing the joints only up to the height that it actually needs for motion. Since the humanoid robot has its most of the weight on the upper part of body, it brings the COG significantly higher. To counter that, cutouts are made so that it can act as wire passthroughs and also reduce the weight at upper part. Also, the leg parts are printed with 100%-layer infill so that the lower weight contributes to lower the COG.

Increasing the Base Area

The foot of the robot is made wider to cover as much surface area as possible to increase the stability. The foots are made wider to an extent that it just doesn't overlap with each other during the motion.

Also, for stability, the two legs of the robot are placed very close to each other so that when the robot puts up all its weight in one leg during motions, the COG doesn't move outside the convex hull. This ensures that the motion frame is stable when robot puts all its weight on one leg.

Another method to measure stability is to find the ZMP (Zero Moment Point) for the robot. ZMP point is the point on the ground where the total moment (torque) due to gravity and inertia is zero. The ZMP of the robot can be calculated using the formula:

$$\begin{Bmatrix} x_{zmp} \\ y_{zmp} \end{Bmatrix} = \frac{1}{\nu} \times \begin{Bmatrix} \sum_{i=1}^{n} m_i(\ddot{z}_i - g_z)x_i - m_i(\ddot{x}_i - g_x)z_i - J_{y,i}\ddot{\phi}_{y,i} + SF_x \\ \sum_{i=1}^{n} m_i(\ddot{z}_i - g_z)x_i - m_i(\ddot{y}_i - g_y)z_i - J_{x,i}\ddot{\phi}_{x,i} + SF_y \end{Bmatrix} \quad \textbf{(4-62)}$$

Where,

$$\nu = \sum_{i=1}^{n} m_i(\ddot{z}_i - g_z) - \sum_k F_{z,k} \tag{4-63}$$

$$SF_x = \sum_k \left(z_k F_{x,k} - x_k F_{z,k}\right) \tag{4-64}$$

$$SF_y = \sum_k \left(z_k F_{y,k} - y_k F_{z,k}\right) \tag{4-65}$$

Where, $J_{y,i}, J_{x,i}$ denote the moment of inertia in the x and y direction of link i; $\phi_{y,i}, \phi_{x,i}$ denote the pitch angle of link; $F_{y,k}, F_{z,k}$ are the y and z-direction of the external force and g_x, g_y, g_z are parts of the gravity acceleration.

ZMP calculation is also very complex and computationally expensive too for the prototyping hardware used for the project. As a result, again a qualitative approach is used. Since the moment due to gravity and inertia is zero in ZMP point, the robot should be stable in the position without any forces acting on it by the motor. If this is achieved the ZMP can be considered to be in the area of the convex hull of the connection points between feet and ground. This method was adopted for finding some of the stable positions of the robot. But in most of these cases the position was not able to be found. As a result, some manual adjustments were made to make the robot stable and ready to move to next frame of the motion stably.

4.3.19 Incorporating into Simulation

The CoppeliaSim provides a world with built in physics engine for simulating the robot activities. The world contains certain nodes, which are shapes and objects that can be programmed through a programming interface. These nodes are of two types: Base nodes and Proto nodes. The Proto Nodes are built in nodes which are pre-programmed and have existing controller code.

The base nodes are the user-defined nodes that can be imported from an external file. For the purpose of Humanoid Robot Simulation, the 3D model from Fusion360 is

imported as a “Robot node” into the world. Under this node, the controller arguments and code are provided and the simulation is run.

The controller code in simulation software controls the states of the robot dynamically. It is based on feedback system so that the robot can adjust its states automatically as seen from Figure 4-32.

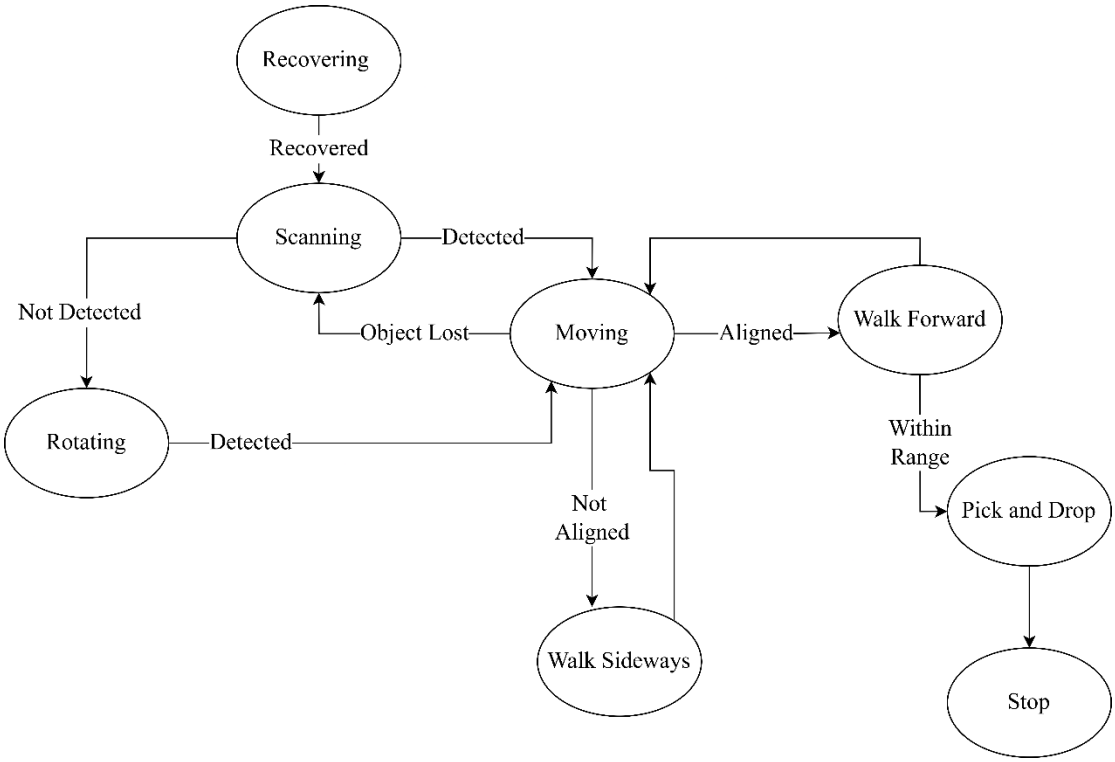


Figure 4-32: States of Robot

Depending on the conditions like Recovered, Detection, Alignment and Range of Object, the controller changes the states of the robot.

The states are as follows:

Recovering: Robot is recovering from fall. If the robot is bent more than threshold angle, this state is set.

Scanning: The robot is scanning the environment for object detection.

Rotating: The robot is rotating clockwise or counterclockwise, in the direction of the detected object. It is always set after object comes in view after scanning.

Moving: The robot is in motion and covering a distance. Depending on alignment, it sets its state to Walk Forward or Walk Sideways.

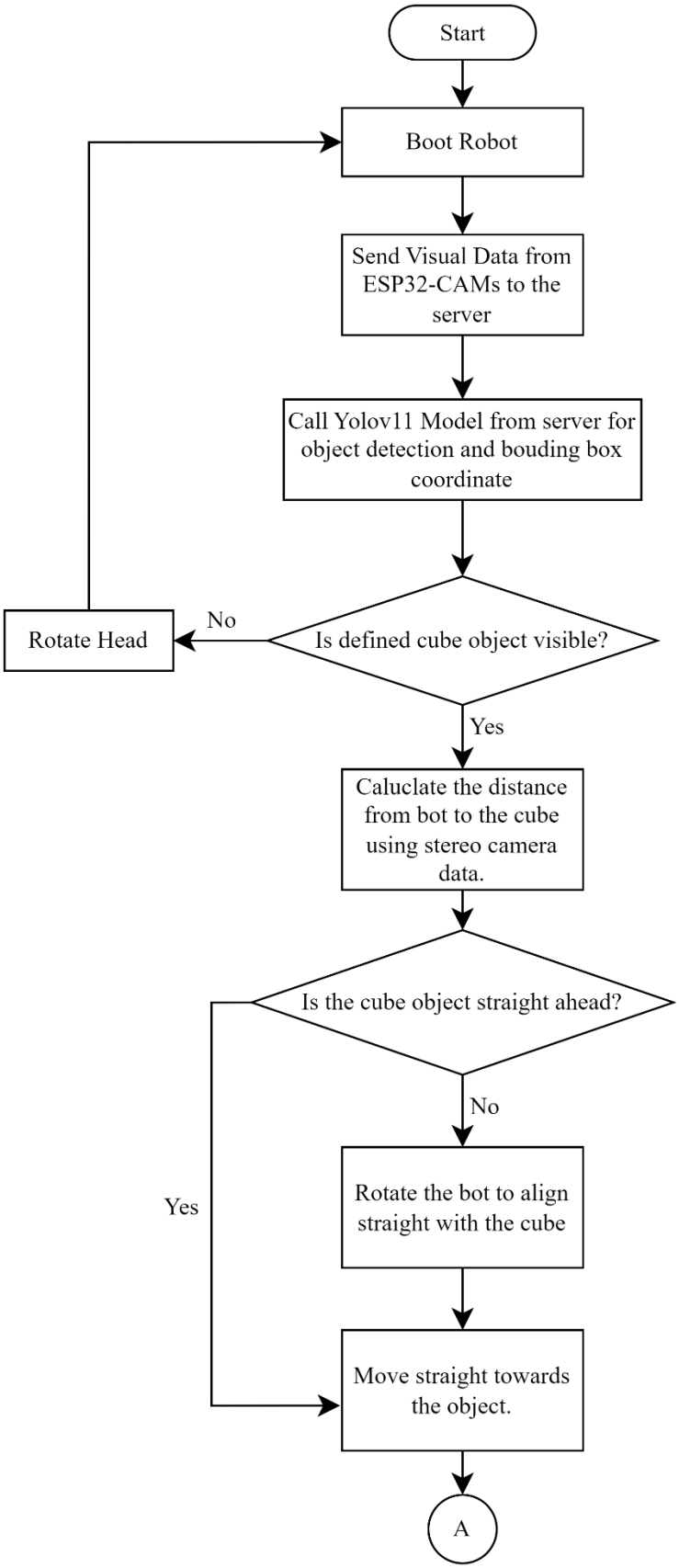
Walk Sideways: The robot is in state of walking sideways to align to the object.

Walk Forward: The object is aligned in this state and robot is walking towards the object straight ahead.

Pick and Drop: After reaching the picking range, the robot sets its state to Pick and Drop and it states to pick the object in front.

Stop: The robot stops its operation in this state.

4.4 System Flowchart



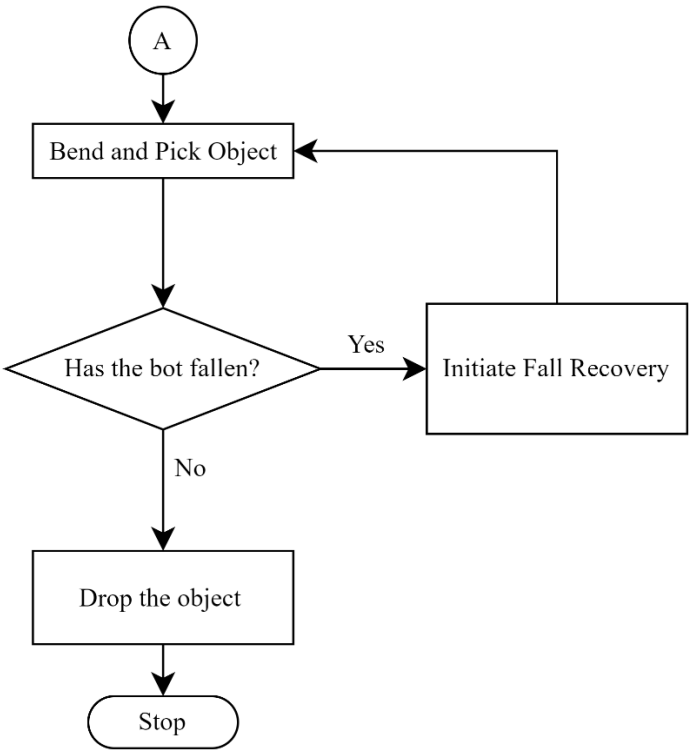


Figure 4-33: Proposed System Flowchart

4.5 Performance Evaluation

4.5.1 Object Detection

The performance of object recognition and detection of the model is evaluated from multiple quantitative evaluation metrics calculated from obtained data as well as some readily available evaluation metrics for the YOLOv11 model. The evaluation and performance metrics that are used are:

Mean Average Precision (mAP): This is a primary metric for evaluating an object detection model. On the basis of IoU threshold of 0.5 and 0.5-0.95 there are two types as mAP50 and mAP50-95. A good value of mAP50 and mAP50-95 is considered above 80% and 50% respectively and a bad value is considered below 50% and 30% respectively. On the basis of this, the trained object detection and recognition model is evaluated for detection precision.

Precision and Recall: Precision is a ratio of true positive detections to the total number of detections made. High precision means that the model makes fewer false-positive errors. The value above 85% for precision is considered good and value less than 70%

is considered bad. Based on this the instance of false-positive detections can be evaluated. Similarly, recall is a ratio of true positive detections to the total number of actual objects present in the images. High recall means fewer true objects miss. With a good value of 80% and a bad value of 70%, this metric is used to evaluate true detection misses.

4.5.2 Navigation

The navigation feature is evaluated based on how accurately the robot moves towards the object, aligning itself in a position for effective object pickup. This feature can be evaluated by conducting an evaluation test where the robot's distance (d) from the object before it initializes the pickup mechanism. The robot requires to be at a specific distance (d_s) to pick up the object. The difference in this robot's stopping distance (d) and predefined specific distance (d_s) can be used to calculate the error in distance ($d_e = d - d_s$) for navigation. The average of these two errors in multiple test setups is used as a metric to evaluate the navigation performance. The metric measures the error as "distance error of x cm".

4.5.3 Fall Recovery

The performance of the robot on fall recovery is evaluated based on its ability to recover from different fall angles and the movement it generates to recover from the fall. Although the fall recovery procedure for the robot for forward and backward fall is hard-coded, the movement that the robot generates to position itself from sideways fall to the predefined fall recovery position is dynamic. This angle of fall and its recovery status is recorded in a table and the percentage is calculated based on the number of times it recovers from the fall.

4.5.4 Pick and Drop Objects

The detection of object pick/drop state is determined through object detection. For evaluating the pick and drop accuracy, the number of times the object the robot picks/drops is compared to the number of experiments conducted.

4.5.5 Distance Estimation

The estimated distance from the camera to the object is evaluated by measuring the actual distance between the camera and the object using a measuring tape beforehand during the experimentation stage and comparing it with the estimated distance. The percentage error produced by the distance estimation methodology for each distance is calculated as:

$$\%\boldsymbol{error} = \left| \frac{\boldsymbol{measured\ distance - actual\ distance}}{\boldsymbol{actual\ distance}} \right| * \mathbf{100}\% \qquad \textbf{(4-66)}$$

The overall error can be calculated as the mean of the errors of each points. Similarly, the overall accuracy of the estimation is calculated as:

$$\boldsymbol{Accuracy} = \mathbf{100}\% - \boldsymbol{average\ error}(\boldsymbol{in}\ \%) \qquad \textbf{(4-67)}$$

For easy visualization, a plot is made by plotting the curve of actual distances and predicted distance in single figure. This allows to visualize how much the estimated distances are deviating from the actual distances.

To get further insights on the distances in which the error percentage is maximum or minimum, the curve of error at each distance is plotted.

5. IMPLEMENTATION DETAILS

5.1 Network Setup

The mobile application, ESP32-CAMs and the Node.js server are connected to each other through a local network. The static IP addresses are assigned to each component. The static IP addresses used are:

ESP32-CAM-1: 192.168.254.14
ESP32-CAM-2: 192.168.254.15
Node Server: 192.168.254.16
Default Gateway IP: 192.168.254.1
Subnet Mask: 255.255.255.0

The subnet mask of 255.255.255.0 is used for the class C type network used by home networks. These static IP addresses ensure that the applications do not need to recompile with different IP addresses each time they need to run.

Since the ESP32-CAMs only supports 2.4GHz network but not the 5GHz network, the overall local network is in 2.4GHz. The overall maximum speed of the 2.4GHz network i.e. up to 100 Mbps is enough for the transmission of the images 10 frames per second by the ESP32-CAMs and the images hosted by the server to the flutter application where each image of size 480x320 are of approximate size of 3-6 KBs.

5.2 Dataset Generation

When the 'save' command is received from the application, the server detects the data received into the two frame streams. The data is saved as an individual frame on corresponding image streams, where a frame counter counts the number of frames received. The raw data is read by the Jimp module. Finally, the frame count is mapped to each image and saved to a directory in jpg format. Altogether, a total of 4432 images were captured to train the model on a cubical object.

5.3 Data Augmentation

Data is augmented to generate new training examples for the model to learn from different scenarios and make the model able to detect the object at most of the situations. First of all, the images are stretched to 640 x 640 resolution as a preprocessing step and then various data augmentation steps are applied simultaneously.

5.3.1 Data Augmentation Steps

1) Flip Image:
    - If 'flipAxis' is horizontal reverse the order of pixel in each row
    - If 'flipAxis' is vertical reverse the order of pixel in each column
2) Rotate Image by a random angle between -14° and 14°
3) Adjust the Saturation by a random factor between -10% and 10%
4) Apply Gaussian blur with a random sigma up to 1.4px
5) Return augmented image

After data augmentation, all the images are exported as a dataset. The final dataset contains 11578 images for cubes without considering the data split for train, test and validation set.

5.4 Model Training

The YOLOv11 model is trained using its nano model. The nano version of the YOLOv11 model; YOLOv11n, although being a pre-trained model, is not able to detect custom objects so, it is trained on the dataset prepared above. The model is trained in Google Colab using T4 GPU provided by the Google. First, the weight file (.pt) and the configuration (.yaml) file of the YOLO are imported into the environment. Then, the prepared dataset with bounding box coordinate data is also imported in the training environment. The dataset is split into three parts as training set, test set and validation set with 70%, 20% and 10% distribution for cubes. All the images in the dataset are annotated using a combination of automatic and manual labeling. First, all the images are passed through an auto labeling model using a prompt-based auto-labeler. After that, the auto-labeled images are verified and approved manually and corrected during the process.

Bounding box coordinates format: [class ID, center X, center Y, width, height]
Example: [0, 0.59609375, 0.41796875, 0.25, 0.40625]
0 : *class ID*
0.59609375 : *center X*
0.41796875 : *center Y*
0.25 : *width*
0.40625 : *height*

The model is trained on images with some training hyperparameters set to following values:
learning_rate : lr0 = 0.01, lr1 = 0.01
train batch_size: 64
tnference batch_size: 16
tmage_size: 640 x 640
epoch : 110
momentum : 0.937
weight_decay: 0.0005

5.5 Server Implementation

The server performs multiple functions based on the request it receives from the client application (Humanoid Robot App). The dataset generation feature of the server is described in section 5.2. Similarly, the other functionality of the server and their implementation process are as follows:

5.5.1 Receiving Image Stream

The server receives the image stream from both the ESP32-CAMs and creates a stream of received image buffer data using the *'ffmpeg'* library. This library requires the *ffmpeg* executables to be installed and set up on the device's environment variables correctly on the local machine on which the server runs. Both the streams are then piped to a main stream from where they are piped to different streams based on the tasks that need to be performed by the server:

Save Image Stream

This stream is responsible for saving the images from the both streams to generate the dataset as explained earlier.

Calibration Stream

This stream is responsible for capturing and calculating the value of the unknown parameters (focal length and $\tan\theta$) which are required for distance calculation.

Output Stream

This stream is responsible for streaming the received images to the mobile application. Also, the distance is calculated using the data from this stream.

5.5.2 Image Processing

This functionality is achieved in the server using the *'onnx-runtime'* which is a 'Cross Platform Machine Learning Model Accelerator'. Since the YOLO model from Ultralytics is not directly available in the Node.js, the weights from the YOLO training cannot be directly used for bounding boxes prediction. As a workaround, Ultralytics offers the tool that converts the *'.pt'* files of YOLO to the *'.onnx'* format which can be used by *onnx-runtime*. The converted *'.onnx'* file is then imported using the *onnx-runtime* to create a model. The overall image processing part can be categorized into two major steps:

Image Preparation

The images received from the ESP32-CAMs are of resolution 480x320 which is not the image size supported by the YOLOv11 model which is of resolution 640x640. So, the initial dimension of the image is recorded and the image is first resized to a resolution of 640x640. Also, the image data format is not directly supported by the model.

Input Image Format: ***[r, g, b, r, g, b, r, g, b, ...]***

Image format supported:

[[red_values(640x640)], [green_values(640x640)], [blue_values(640x640)]]

So, the input image is converted to the form supported by the model using "*sharp*" image processing library's *resize* function, which in turn uses resampling and interpolation methods to resize the image. The final prepared input image is a tensor of shape (1, 3, 640, 640). This prepared input now can be directly fed to the model for prediction.

Bounding Boxes Calculation

The prepared image, when fed to the model, outputs 8400 bounding boxes. To find out the bounding boxes in which the object is present, the bounding boxes need to be filtered out.

The bounding boxes are first sorted in descending order and the bounding boxes that have the confidence score below 70% are discarded. From the remaining bounding boxes, the first bounding box is pushed to the filtered bounding boxes list. Then the other bounding boxes are only pushed to the filtered bounding boxes list only if they have an IOU value of less than 70%. This ensures that the model doesn't classify the same object multiple times.

5.5.3 Serving Clients

The server consists of different endpoints that allow the client application to send request for performing different tasks by the server. The endpoints and their tasks are as follows:

/control : Allows to turn on/off the image stream receiving process.

/control/save : Allows to turn on/off the saving image for dataset generation.

/control/calibration : Allows to send the distance value and generate bounding boxes for the image when placed at that distance and use those distance values to calculate the parameters.

/control/distance : Allows to calculate and get the distance of the object in the frame.

5.6 Hardware Control

The basic movements, like the object pickup and walking movement, have been implemented on a physical level. The ESP32 is programmed using the Arduino IDE using C++ as the programming language. The ESP32 sends the control motor movement signals to the LU9685-20CU controller, which in turn sends the required PWM signal to the servo motors to control them. Since the LU9685-20CU module works directly by receiving I2C commands, the simple I2C library called *'Wire.h'* is used to send commands to the module from ESP32. The process of sending the commands is also simple.

1. First, begin the transmission by providing the address of the module.

   Wire.beginTransmission(128);

2. Then write the command to the module.

   Wire.write(buffer,2);

3. Finally, end the transmission with the module.

   Wire.endTransmission();

Here, the buffer object is an array of *byte* datatype, which is equivalent to the *unsigned char* datatype of C++. The first value of buffer must be the port number to which the servo is connected in the module, and the second value must be the angle of rotation of the servo in that port. The second argument in the '*write*' function is the length of the buffer to be sent, which here means that only the first 2 data points in the buffer array are to be sent.

The OOP approach has been used to write code to control different parts of the robot, and motion controls are as abstracted as the methods of the Robot Simulation object. This allows us to make fundamental movements and then compile up the fundamental movements to generate more complex movements from a higher level of code. Considering the complexity of the robot movement, this approach has been utilized.

5.6.1 Smooth Inter-Frame Interpolation

The walking motion implemented in the physical robot is the positioning of the robot's joint angles frame-by-frame. Each frame consists of joint angles of the robot, which defines a position of the robot. Each frame is allocated a specific time to reach the described position. But if the angle is directly assigned to the joint motors without interpolation, all of a sudden, all motors try to move to the described position which has two drawbacks.

To move a larger angle with the load with very less time, the motor requires higer torque which is explained by (5-1). Moreover, Force is rate of change of momentum as evident from (5-2).

$$\boldsymbol{Torque = Force \times Perpendicular\ Distance\ from\ Fulcrum} \tag{5-1}$$

$$\boldsymbol{Force = \frac{mv_f - mv_i}{t}} \tag{5-2}$$

Where

m = Mass

v_f = Final Velocity

v_i = Initial Velocity

t = Time

On a stable frame, $\boldsymbol{v_i = 0}$

$$\boldsymbol{Force = \frac{mv_f}{t}} \tag{5-3}$$

On sudden and fast change in motor position, the value of ‘t’ is extremely low and also the motor is acting to move a larger mass. As a result, the force/torque required by the motor to reach the desired position is very high. These values are so high that it surpasses the maximum torque that can be supplied by the motor by multiple folds. This can also damage the motor.

For high torque to be supplied by the motor, the current requirement of each motor also increases drastically. This can cause the module LU9685 used to control the motor to be damaged too.

So, to overcome these problems, the process of inter-frame interpolation is performed.

ALGORITHM: INTER-FRAME INTERPOLATION

1 *The Robot is initialized with a current frame angle.*

2 *A smaller incremental time frame of 50ms is defined*

3 *The time defined for the robot to reach the next frame is divided by the 50-ms time to get the number of incremental steps to be interpolated as shown in equation (5-4)*

4 *For each joint the change in angle to reach the next frame is divided by this number of steps to get the small incremental angle for each frame. Equation (5-5) shows for left leg, for instance.*

5 ***Loop*** *for number of steps:(1 to No. of steps)*

6 *Increase the Current Angle of each joint by the fixed incremental angle. For the above left leg knee example, see equation (5-6).*

7 *Write the Next Step Angle values to the motor.*

8 *Assign the Next step Value to the Current Angle.*

9 *Add delay equal to the steptime.*

10 ***end***

$$\boldsymbol{No.ofSteps = \frac{T_n}{T_i}} \tag{5-4}$$

Where,

T_n = Time to reach next framw

T_i = Incremental Time (50 ms)

$$\boldsymbol{\theta_i = \frac{\theta_{lkc} - \theta_{lkn}}{No.ofSteps}} \tag{5-5}$$

Where,

θ_i = Incremental Angle for Left Leg Knee

θ_{lkc} = Left Leg Current Angle

θ_{lkn} = Left Leg Knee Angle in Next Frame

$$\boldsymbol{\theta_{lkn} = \theta_{lkc} + \theta_{i}} \tag{5-6}$$

The Source code for this interpolation is shown in Appendix K:

5.7 App Implementation

The mobile application for this project is implemented using Flutter. The basic '*http*' package is used for handling the network requests. The '*bloc*' package is used as a state management solution for the application. The application incorporates effective exception handling techniques for any exception thrown during a network request. The overall UI of the application is shown in Figure 9-17.

5.7.1 Motion Planning Interface

The frame data imported directly from the simulation doesn't comprehend directly to move the physical robot stably. For the process of making the frames stable and also to manually design the intermediary frames, the manual control of the motor is necessary. The motion planning interface of the robot helps to achieve the same functionality. The interface allows to control each of the robot's joints manually. Also, it allows to send the data frame by frame from the server to the robot to analyze its stability. The same page allows to initialize the motors to the initial position for the robot to move to a stable standing pose. The manual control of the motor is shown in Appendix L:

5.7.2 Use Case of Application

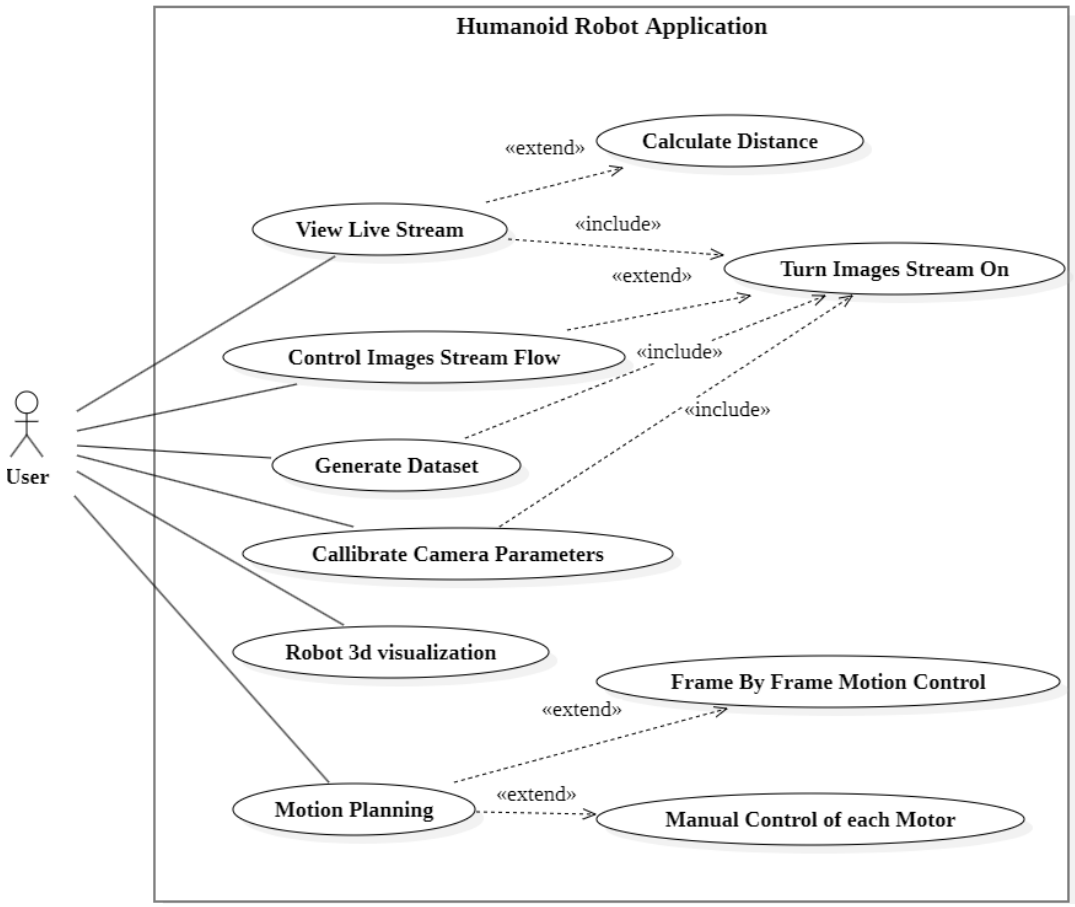


Figure 5-1: Use Case Diagram of Robot Mobile Application

5.8 Robot Model

For the bot movement, the two legs of the robot were grouped as two kinematic chains. Each chain was built from the hip all the way down to the foot. The rigid parts of each chain were connected with rotational motors, providing one degree of freedom to each joint as shown in Figure 5-2. Each joint in each kinematic chain moves its position relative to its parent joint.

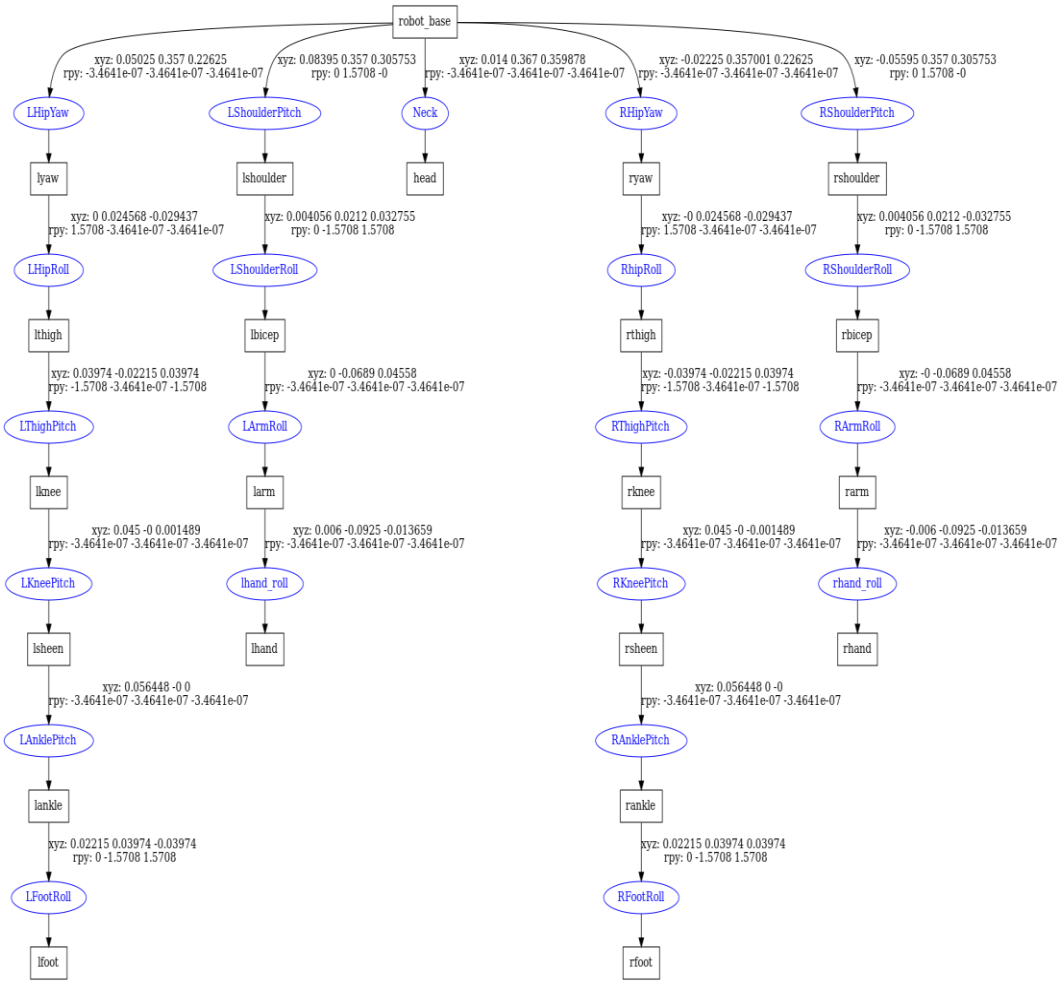


Figure 5-2: URDF of Robot Model

Moreover, for the links to be actually connected to each other, their position of anchors was set to the coordinate of the corresponding link connector, which is the position of the joint.

5.8.1 Setting Up Simulation Environment

The simulation model of the robot was imported and simulated in the simulator with approximated values of different parameters of the actual robot.

5.8.2 Importing Robot Model in the Scene

Before importing the simulation model in the CoppeliaSim, the model first needs to be exported as a mesh from the design software. It can be exported as .OBJ or .STL file after merging non-significant and minor parts to reduce the simulation overload. So,

the simulation model of the robot was imported as a mesh in .OBJ format in the CoppeliaSim.

By default, when importing the mesh file, there is auto scaling but a custom scaling parameter can be added to match the size of actual 3D design close to the real size.

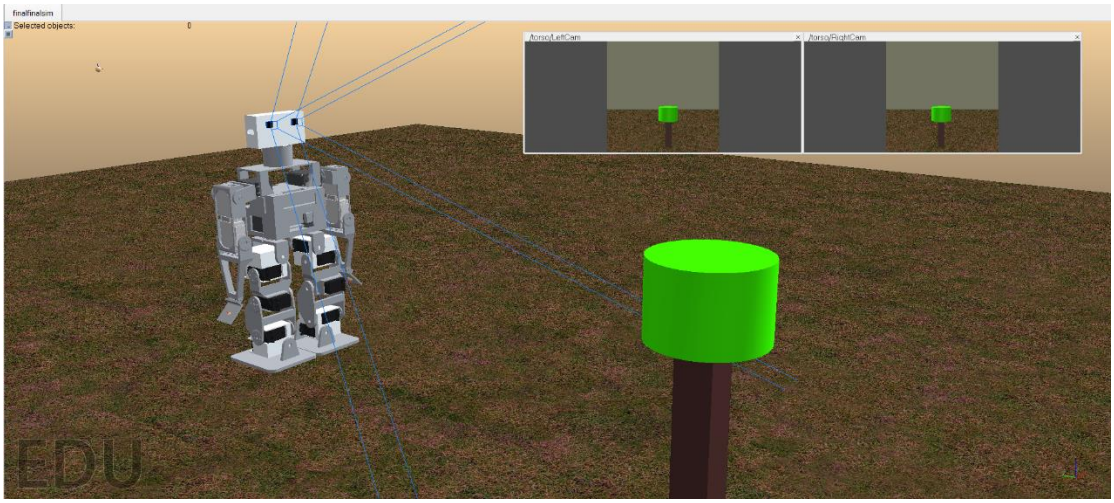


Figure 5-3: Robot Model in Simulation

Convex Decomposition of Mesh

The imported mesh file can be simulated by the available physics engine but is not recommended because of the extreme computational requirements and unwanted behavior in simulation. At the import the meshes of the parts are considered as non-convex shapes rather than convex shapes by the CoppeliaSim. The non-convex mesh file is made up of a large number of triangles which raises computation complexity. So, the non-convex shapes need to be decomposed into a convex shape and some simple shapes to primitive shapes like cube, cylinder, plane.

In CoppeliaSim, convex shapes perform faster and are more stable than random shapes, in regards to dynamic collision and response calculation.

The complex parts of the robot model are convex decomposed using the inbuilt convex decomposition tool provided by CoppeliaSim and a cube was extracted for the body of the robot. After that, the view can be changed between using the layers tab and putting the meshes in the top layer with convex-decomposed in the bottom layer.

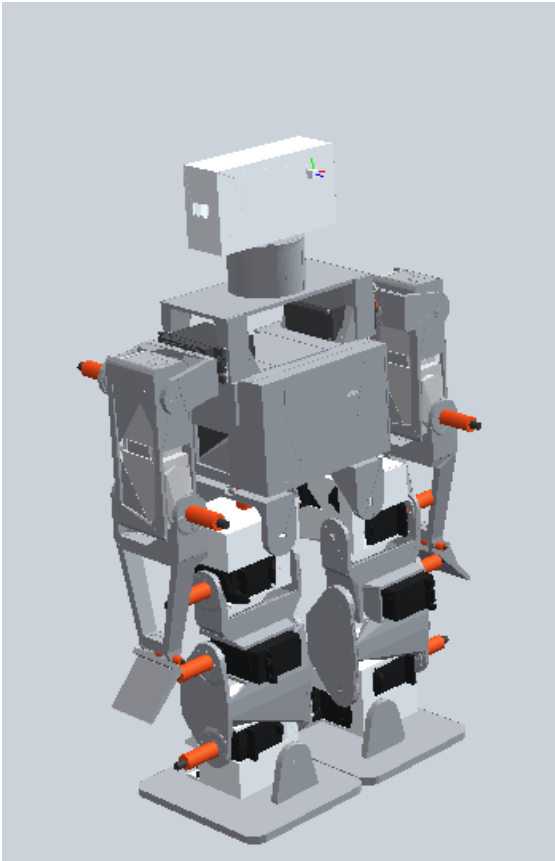


Figure 5-4: Convex Decomposed and Visibility Layers of Model

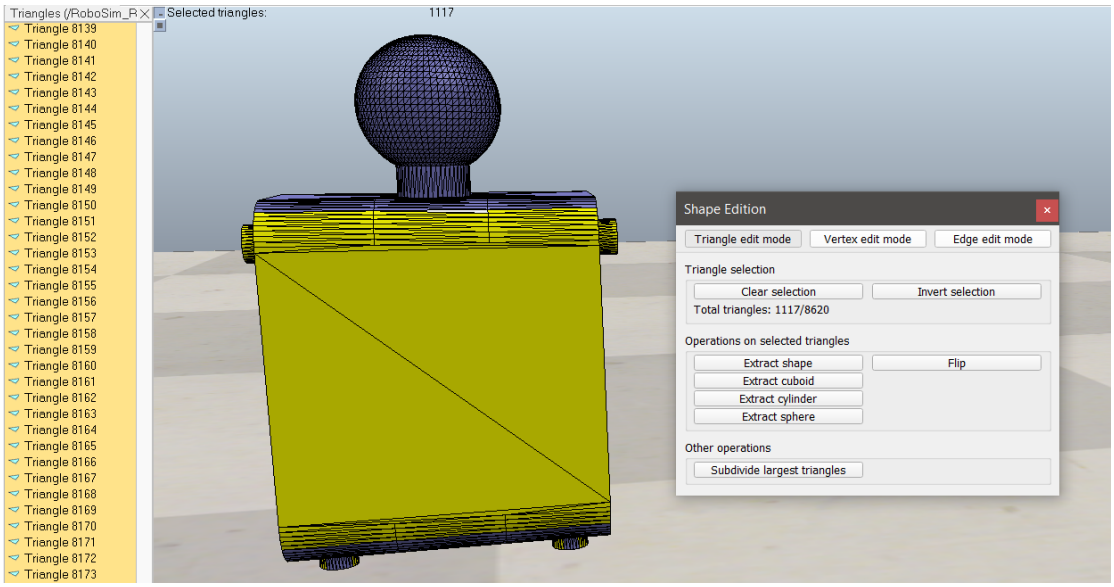


Figure 5-5: Convex Shape Extraction of Model

Defining Joints

In order to simulate the joints of the legs properly, the joints need to be placed in their proper position in the leg of the correct type. Each joint can be added using the add joints feature and positioning them in their proper position in the robot model.

A total of 6 joints are placed in each leg with 12 joints for both legs and the joint's type is set to position control type to simulate the working of a servo motor.

Scene Hierarchy

To let the simulator know the relationship between leg links and joints, the shapes, mesh and joints are grouped together in a hierarchical manner. After creating joints and decomposed shapes, all of them can be seen in the scene hierarchy. All of the links and joints are arranged in parent-child relationship to properly attach joints to the parts and parts to the joints. This defines the joints of the robot for accurate movement simulation.

The joints are made parent of the link that follows the joint and the meshes are made child to the link's decomposed shapes to make them non function in the simulation and are just for visuals.

Grouping and Masking

In the scene, each link is connected to its respective joint in a hierarchical manner. Also, there are some parts that are not simulated but need to be visible or be responsible as an integrated part of another object. For this, multiple shapes can be grouped together so that they behave as a single shape.

Each link of the leg is responsible and dynamic, which means each link can collide with each other and respond to the physics engine. This causes some issues in simulation when unwanted collision occurs during simulation as the convex decomposed shape does not have the same geometry as the designed mesh. To make the links move freely without any unwanted collision, each link is placed in their own local masking group and alternate between joints. Two consecutive links are placed in different local responsive masks.

Mass and Torque Assignment

In order to simulate the robot with real world parameters, the mass of each part and torque of the joints need to match with the real values. This is done by changing the dynamic properties of each part from body to the links. In simulation, by assigning realistic mass to each component of the robot, including motor and battery mass, the total mass of the body is set to about 2 kg. Similarly, the torque on the joints is set to 6 Nm.

5.9 Robot Movement

Three major functionalities for Robot's movement were implemented.

5.9.1 Forward Movement

The movement of the robot in forward direction is implemented by continuous trial of balancing center of mass and observations in multiple experiments. From the simulation data, the generic walking cycle was observed and a table of motor angles were recorded.

The motor angles at various frames were then obtained. After obtaining motor angles, adjustments were needed to balance the movement. In order to stabilize the walking motion, the center of mass needed to be stabilized so that the robot did not collapse due to the robot's weight distribution during movement. The first step was to raise the right leg, by changing the position parameter of the motors. To counter the absence of supporting structure on the right side of the sagittal plane, the body was tilted on the left side and the robot was balanced entirely on the left leg. The motor located at the left foot's roll position proved crucial for taking this step.

Then, the motor located at the right thigh's pitch position was manipulated to move the right leg forward and using left foot's roll motor, the body was once again brought to a position where it was supported by both legs.

Similarly, this process was applied, inversely, for moving forward the left leg as well.

5.9.2 Sideways Movement

The sideways movement was implemented in simulation using Forward Kinematics. First of all, a set of joint angles for each six joints on each of the legs were calculated through hit and trial to develop a relatively stable and human-like movement. Eight sets of such joint angles were calculated in order to divide the movement into eight frames, which means, a total of 96 joint angles were calculated in order to make the robot take sideways steps in one direction.

For smooth transition from one frame to another, the angles were interpolated between the start angle and end angle. The interpolation ratio was the time elapsed between the start of one frame to the beginning of the consecutive frame.

5.9.3 Rotation Movement

The rotation movement is achieved with the manipulation of yaw motors present at the hip of the robot. Depending on the direction of the rotation, the yaw motor was operated to rotate the leg. But before rotating the yaw motors, the leg needed to be lifted. So, the center of mass was shifted on another leg, the robot was brought to a stable state and finally the yaw motor was moved. Then gradually the legs were placed on ground and the resulting rotation motion was implemented.

5.9.4 Pick and Drop Movement

Picking up motion required a cubical object to be placed in front of the robot. The object was placed on an elevated position so that the object was within the reach of the robot. Picking up object requires bending of the body. Humans normally bend with the help of flexible spinal bone. But, due to lack of freedom in movement in upper body of the robot, bending position was obtained entirely through the motors in legs. The pitch motors of thigh, knee and feet were collectively manipulated to achieve a suitable bending position.

To grab the object, the three motors in each arm were collectively manipulated to reach out to the object. When in the grabbing position, the motors in roll position of the arm were bent in such a way that the distance between the tips of both arms was less than the width of the object to the picked.

To implement the drop motion, the pick motion was reversed. First, the body was bent, and the object was put to the level of the elevated position as before. The grip was released and all the motors were reset to original position.

5.10 Physical Implementation of Forward Walking Motion

5.10.1 Motor Initialization

This is the primary step where the motor maintains its angle to hold the robot's body in the upright position. All the motors need to be set to their angle from the LU9685 to prevent their position from getting disrupted. In the absence of the motor initialization, the robot's motor won't receive the power from the circuitry and cannot hold the body due to heavy torque acting due to the robot's body, and thus the robot collapses. This is the primary step in the robot's initialization before any other tasks.

This is done via setting the initial frame angle to all motor joints one by one from the Humanoid Robot application as shown in Figure 9-18.

5.10.2 Dynamic Environment Implementation of Data from Walking Simulation

The dynamic implementation of simulation data means that the robot is left to interact with the surroundings and perform the action it needs to (here walking simulation). In this environment, the robot gets effected by gravity, friction, and other physical parameters of the real world. In simulation, it is done by just keeping the environment dynamic, whereas in the physical implementation, it is done by placing the robot on the ground or a surface that it needs to walk on.

The robot was stable in the simulation with the same mass and joint position as in the real world, but the time allocated for each frame was so short that the joint angles moved with higher velocity, causing the momentum to increase drastically.

$$\boldsymbol{Momemtum} = \boldsymbol{Mass} \times \boldsymbol{velocity} \tag{5-7}$$

As known from (5-7), due to that increased momentum, the robot overthrew its body, causing the disbalance from the initial frame.

To solve that issue, the time allocated for each frame to complete was increased by fourfold. As a result, the problem of overthrowing was solved.

Then some frames of motion were directly stable when implemented in the robot's body, but most of the frames were not stable and caused the robot to fall due to disbalance. On deeper inspection, it was found that the frame position was itself not the issue, but the steps that the robot took to reach that frame position were causing the disbalance. It is due to the fact that the robot divides the time allocated for those frames into 50ms time slots and then calculates the incremental angle to be taken in each 50ms time slot for each joint and moves each joint almost at the same time. This caused the unpredictable intermediate frames in which the balance of the robot collapsed and resulted in the robot falling.

To solve this issue, some intermediary frames were added to balance the robot in between the simulation frames where it lost its control. This step was a manual approach, and intermediary frames were obtained by controlling each motor manually via the Humanoid Robot application's motion planning page.

The addition of intermediary frames balanced the robot to a greater extent.

5.11 Physical Implementation of Pick-Drop Movement

The direct implementation of simulation data of the pick-drop movement from the simulation was not workable in real environment and caused the robot to flip directly from the first frame. Then the robot bending angle needed to be adjusted to a level where the robot doesn't fall. Then the pickup motion of hands was almost similar to the simulation data.

5.12 Physical Implementation of Rotation Movement

Despite some environmental differences, the ideal rotation motion in simulation was almost consistently implemented in real world after some tweaking angles of the motors. Slowing down the transition time between frames aided in smooth and consistent movement.

5.13 Hardware Challenges and Solutions

The humanoid robot was initially designed using micro servo motors (SG90) as shown in Figure 9-5. After assembling the complete robot, few challenges and issues arised; specifically related to the designed parts tolerances, torque provided by the servo motors and space for electronic components. The connection method used to mount the SG90 micro servo to each joint of the robot was not strong enough using 3D printed servo horns and also the motors could not provide enough torque (claimed torque of 1.8 kg cm at 4.8V) to handle all the joints which caused the plastic geared motors to strip their gears.

All these problems were tackled in the later design iterations using larger and more powerful MG996R servo motors with claimed torque of 12 kg cm. The first design of the robot using a 12 kg Servo motor was successfully simulated but when implemented in hardware produced its own issues with design and 3D printing tolerances, link length between joints, mass distribution and part strength. In this design also, the used 12 kg metal gear servo motor was not able to provide expected torque (claimed torque of 9.4 kg cm at 4.8 V) as shown in Figure 5-6 and Figure 9-16. Also, with all the circuits, battery and camera and microcontroller in the torso, the robot was extremely unstable with added DOF in the torso. In later iteration, all these problems were tackled with various major design changes and power distribution decisions.

To solve all the issues in previous designs, the overall height of the motor was reduced. To do so, a whole new torso was designed with adjusted placement for electronics components, motors and camera position. In order to lower the CG, the length of the leg was significantly reduced by reducing the length of Hip Yaw and the mass distribution was improved by decreasing the width and height of the torso with careful component placement for required mass distribution in front and in rear.

The heavier components were placed at the lower part of the torso to help lower the CG as well. The torso was designed to be as light as possible without compromising its structural integrity and then printed to assemble the robot. The cameras which were placed in the torso in previous designs were placed in a simple head to free the space in the torso considering its less impact on the robot's dynamics. In previous iterations, foot

was also a major issue that caused imbalance in robot's dynamics, this was solved by making the foot plane considerably larger for enhanced balance of the robot.

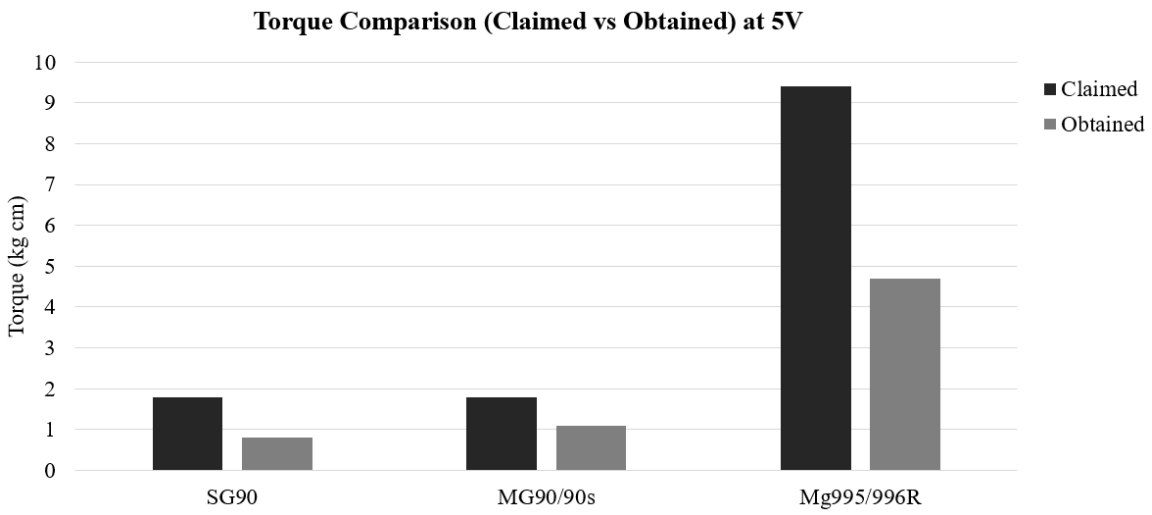


Figure 5-6: Torque Comparison of Servo Motors

With design and fabrication issues tackled, power requirement issue was another major issue to solve. The initially designed and created circuit using 4 LM7805 voltage regulators could not provide enough current for larger motors. So, a new circuit was developed using 12 LM7805 with 3 power channels having 4 regulators in each channel for each leg and arm. This approach was successful for some time but failed due to rapid degradation of the LM7805 voltage regulator caused due to thermal stress and stress induced by rapid variations in load current rendering the circuit useless for the application.

This was solved by using a 200 watt (5V 40A SMPS) external power supply to meet the power requirement of the whole robot. This approach solved most of the issues and challenges with robot's power supply but brought with it another challenge. Due to transient response of the external power supply, the Esp32 microcontroller and two Esp32 Cams could not stay powered on and failed to initialize the robot. This had to be solved by using a second isolated power source for microcontroller and cameras using a buck converter module capable of providing needed power to sensitive components.

6. RESULT AND ANALYSIS

6.1 Model Training

The YOLOv11n (nano) model was trained on the Cube dataset, the object to be manipulated. The model was trained on the coffee bottle and other cubes made of foam material initially to test the performance of the model and later was trained on the actual cube dataset for further implementation in the hardware after analyzing the performance and shortcomings of the model.

6.2 Cube Dataset

The prepared cube dataset contains 10719 total training images, 602 validation images and 257 test images of objects, specifically cube/cuboids of annotated manually with assisted annotation tools. The prepared dataset was passed through pre-processing and augmentation stages for different scenarios.

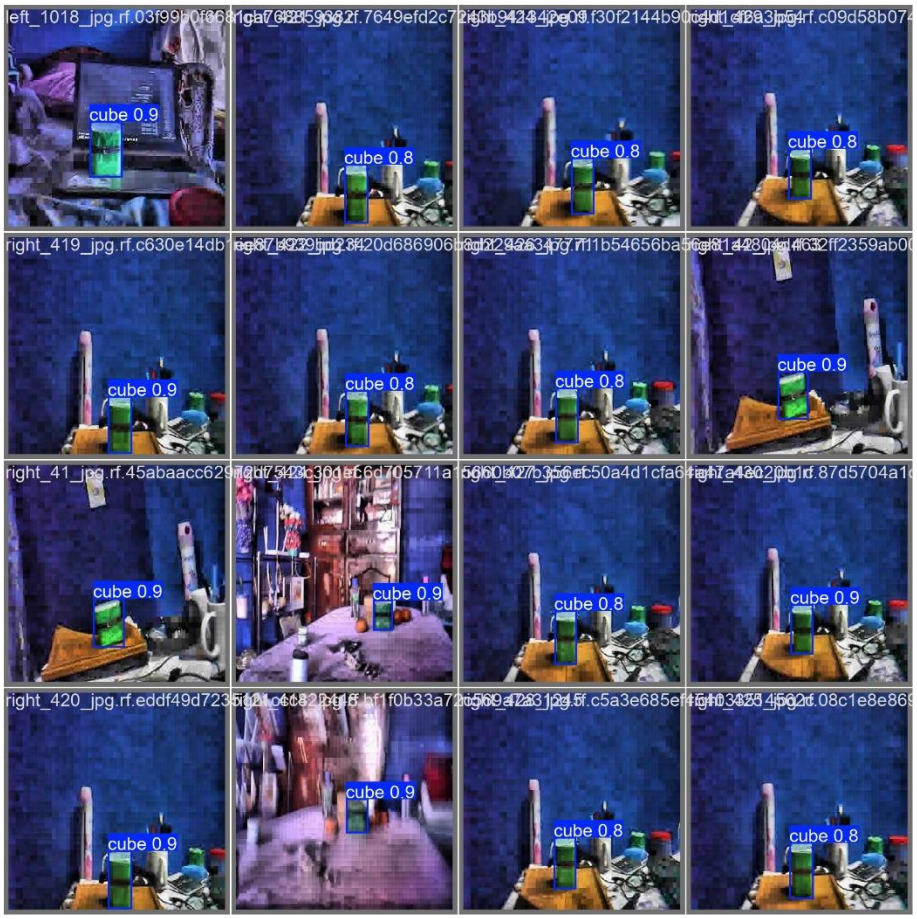


Figure 6-1: Prediction of Cube Object

The model was trained for 110 epochs for optimal object detection with satisfactory confidences for each detection and accurate distance estimation.

The trained model was successfully able to detect and recognize green cube which allows the robot to detect, recognize and reach the uniquely identified objects.

Figure 6-2: Prediction of Cubical Object

The model was evaluated based on different performance metrics:

F1 Curve

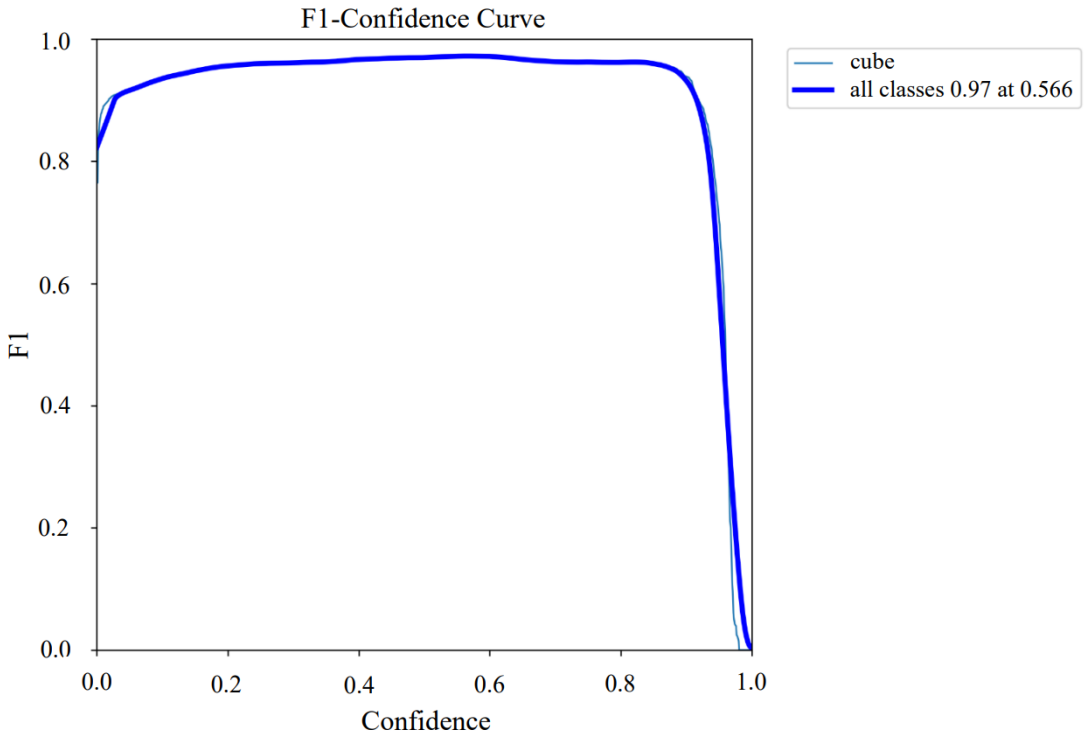


Figure 6-3: F1 Curve for Cube Dataset

The F1-confidence curve for cube is shown in Figure 6-3. For the different confidence values, different pairs of precision-recall values are obtained. The f1 score for each precision- recall is plotted against the confidence level. The unit value of Area Under

Curve (AUC) shows that the model is able to predict the class of objects in the image accurately.

PR Curve

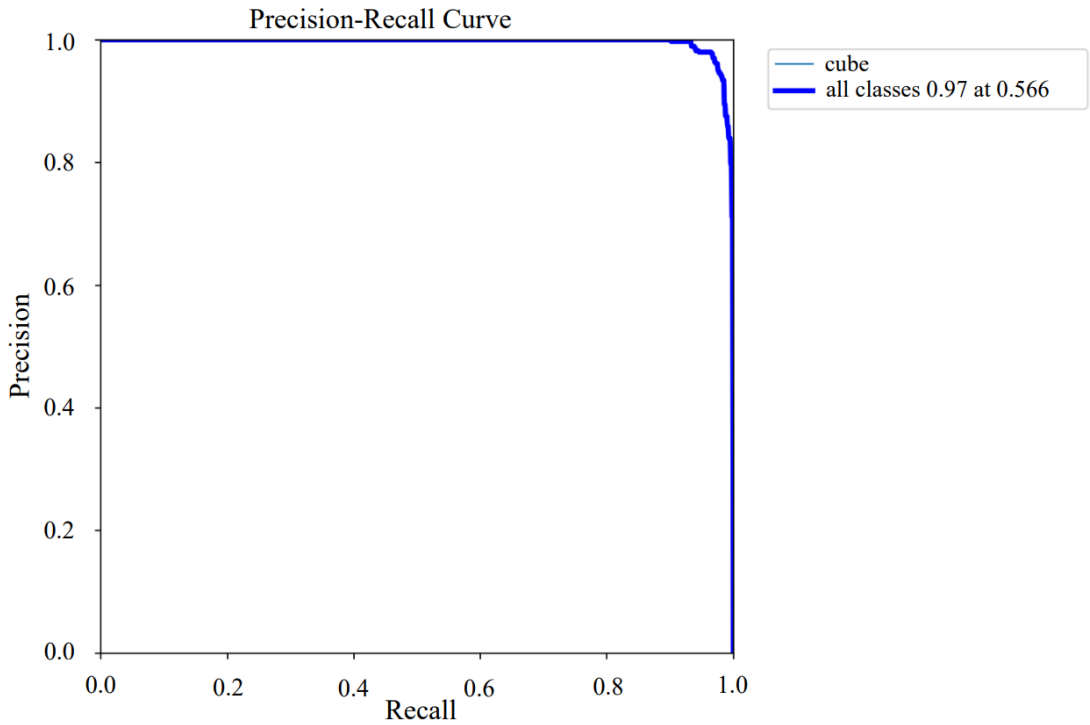


Figure 6-4: PR Curve for Cube Dataset

The precision-recall curve for the cube is shown in the Figure 6-4. For different probability thresholds, a different pair of precision and recall value is obtained. These data points are then plotted to obtain the PR curve. The area under the PR curve is nearly 1 which shows that the model is able to classify the detected objects in the image accurately.

Other Plots

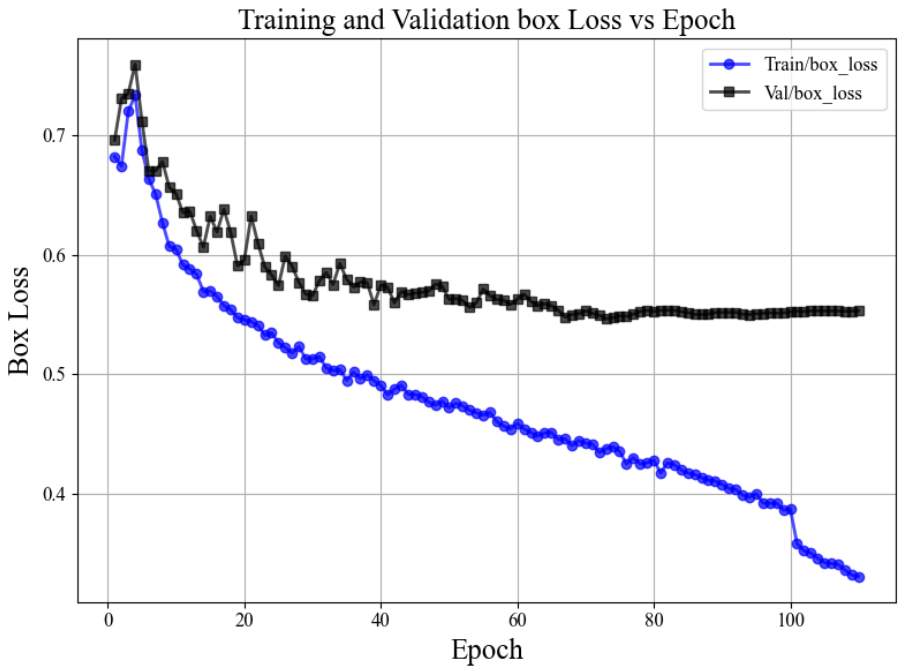


Figure 6-5: Box Loss over 110 Epochs

Figure 6-5 shows the plot between training box loss and validation box loss over number of epochs. The training box_loss represents the loss incurred by the model while predicting the bounding box for the Cube in a training set. Whereas validation box_loss represents the loss incurred while predicting the bounding box for the cube in the validation set. High box loss in training and validation set for the models trained on 0 to 5 epoch shows that they are struggling to locate the bounding box for the cube. With the increase in the number of epochs from 5 to 40 the training box loss decreases steadily. This is also caused by low precision bounding box annotations in dataset.

With the increase in number of epochs, the loss stabilizes for validation dataset. The loss is almost consistent and fixed at a value at about 0.55 when trained for 110 epochs. But for training dataset, the loss keeps on decreasing. The loss gap is about 0.25 for 110 epochs between training and validation datasets, which indicates some overfitting. This can be tackled using early stopping when gap is not that great.

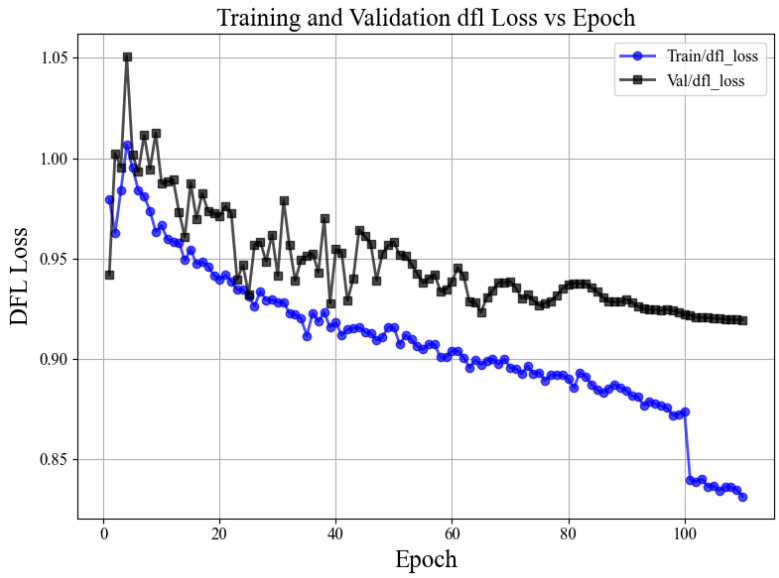


Figure 6-6: DFL Loss Over 110 Epochs

Figure 6-6 shows the plot between training Distribution Focal Loss and validation Distribution Focal Loss over the number of epochs. The Distribution Focal Loss is used to refine bounding box prediction. It adjusts the cross-entropy loss equation by adding a factor that reduces the loss contribution from easy to classify examples (distinct classes) and increases the contribution from hard to classify examples (similar classes). For 0 to 5 numbers of epochs the models struggle to predict bounding box offsets. As the number of epochs increases the DFL loss for both training and validation set decreases smoothly which shows that the models can precisely predict bounding box offset.

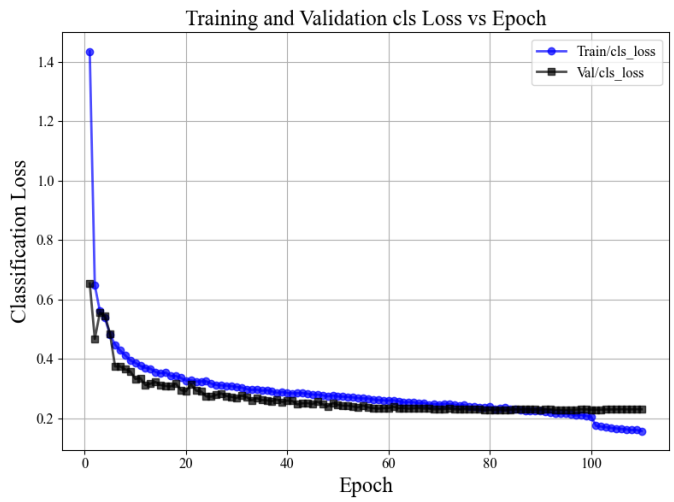


Figure 6-7: Class Loss over 110 Epochs

Figure 6-7 shows the plot between training class loss and validation class loss over the number of epochs. The training class loss represents the loss incurred by the model while classifying the Cube in a training set. Whereas validation class loss represents the loss incurred while classifying the cube in the validation set. High value of the classification loss in training and validation set for 0 to 10 number of epochs shows that the respective models are not able to classify the cube correctly. With the increase in the number of epochs the training and validation class loss decreases steadily. The constant and close values of training classification loss and validation classification loss for the models trained on 100 to 110 epochs shows they are optimized and can perform precise classification.

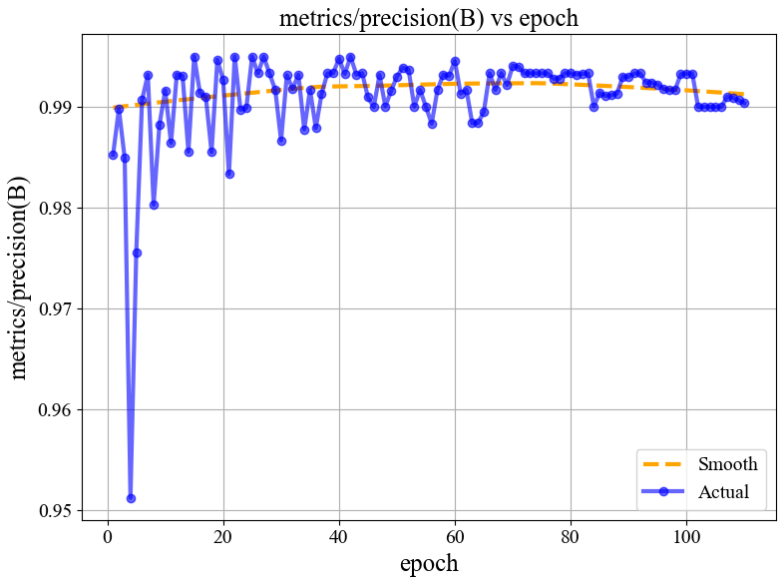


Figure 6-8: Precision over 110 Epochs

Precision of a model can be defined as the ratio of number of correctly classified positive examples to total number of predicted positive examples. In the Figure 6-8, the precision value fluctuates highly from 0.92 to 0.99 for the models trained on 0 to 20 epochs. The fluctuating value of precision shows that the model classifying the other object as cube is high. As the number of epochs is increased up to 110 the precision stabilizes to 0.99 which refers that 99 percent of object predicted as cube are truly cube.

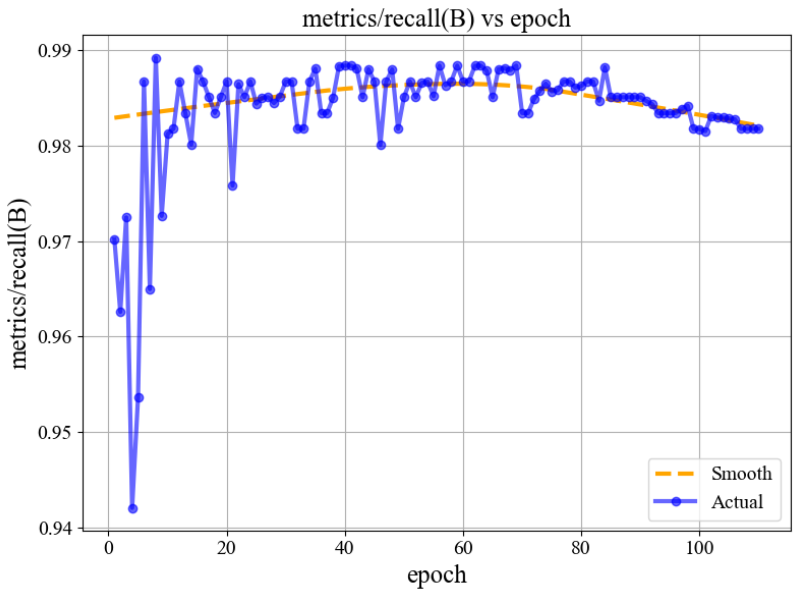


Figure 6-9: Recall over 110 Epochs

Recall of a model can be defined as the ratio of number of correctly classified positive examples to total number of actual positive examples. In the Figure 6-9, the lower value of recall for the models trained on 0 to 20 epochs infers that model predicting cube as other class is high. As the number of epochs are increased up to 110, convergence of recall to 0.98 refers that 98 percent of cube in dataset is predicted as cube.

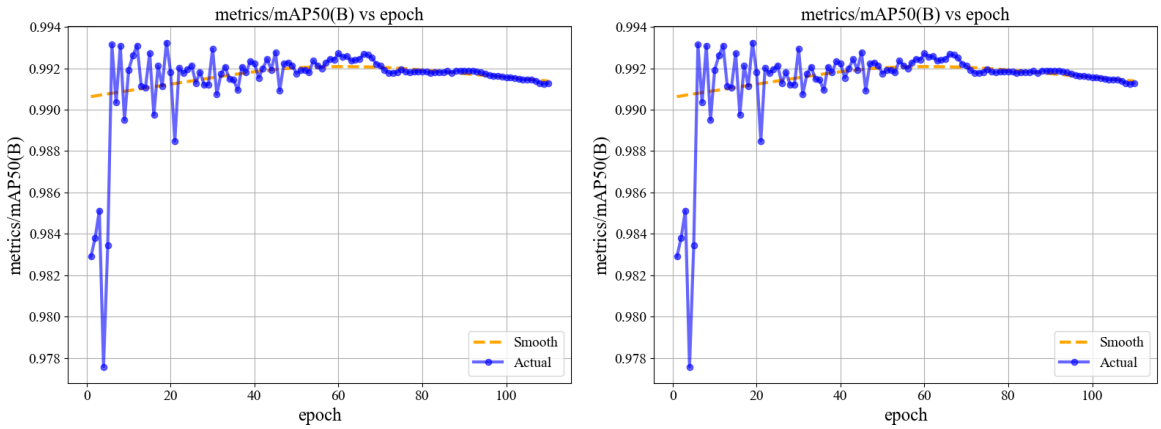


Figure 6-10: mAP50-95 for 110 Epochs

In the Figure 6-10, the increasing mAP value at different Intersections over Union highlights that the model can generalize training data and makes an accurate prediction with the increase in number of epochs.

From all the metrics discussed above it is clear that a model trained on 110 epochs is optimized and suitable for the object detection.

6.3 Parameters Calibration

When the focal length and $tan\theta$ values are calibrated using the two distance values i.e. 30cm and 60cm, the values were obtained as follows:


```
{
  "tantheta": 0.2577295068171453,
  "focal-length": -15.325558951755319
}
```


6.4 Lagging and Heating in VGA resolution

The ESP32 Cam module employs a ESP32-S microcontroller, which is a dual core, 32-bit processor which is responsible for image capture, data processing and wireless communication. The OV2640 camera sensor communicates with ESP32-s through the DVP (Digital Video Port) and SSCB (Serial Camera Control Bus). The OV2640 supports the resolution specified in Table 9-1. The VGA refers to a 680x480 pixel resolution that is suitable for real time object detection. On using VGA resolution, it was found that the ESP32 Cam starts to heat excessively as the processor has to process 307,200 pixels and communicate it wirelessly with the server. To reduce processing overhead and to protect the processor from heating, resolution mode is changed to HVGA which refers to 480x320. The heating issue of the ESP32 is solved as the processor has to process only 153600 pixels only.

The distance estimation of the object using the stereo camera setup directly depends on the number of pixel shifts. When the resolution is decreased from the SVGA to HVGA, the total number of pixels to represent the image also decreases i.e. the number of pixel shifts for the object in stereo setup also decreases, which leads to inaccuracy in distance estimation. It was found that a compromise between accuracy and processor overheating has to be made.

6.5 Servo Motor angle multiplier effect

The servo motor can rotate in a range from 0 to 180. For moving the humanoid robot to a desired location servo motors placed in different joints are to be rotated in a specific angle. On implementing the walking cycle, it was found that there can be ±2-degree fluctuation in rotational angle of each servo motor. To move a single leg of robot 6 servos should be rotated at a specified angle. If all 6 servos produce even a 2-degree error then an error of 6*2 = 12 degree is at the end effector of leg. An error of 12 degree can easily dislocate the robot.

6.6 Issue with LM7805 connected in parallel

On connecting LM7805 in parallel, a heating issue arises which may damage the LM7805 along with the battery. When two or more LM7805 are connected in parallel their output voltage is a common voltage. As each regulator's internal control mechanism operates independently, it adjusts its internal resistance accordingly. Even a slight difference in characteristics of the pass element causes current imbalance. Also the LM7805 has a tolerance of 2% so some of the regulators may be in 5.1V and others may be in 4.9V. The one at high voltage tries to supply the high current compared to the regulators at low voltage. To supply the additional current LM7805 employs a circuit with series pass transistor (Q1) along with Resistance as seen in Figure 6-11.

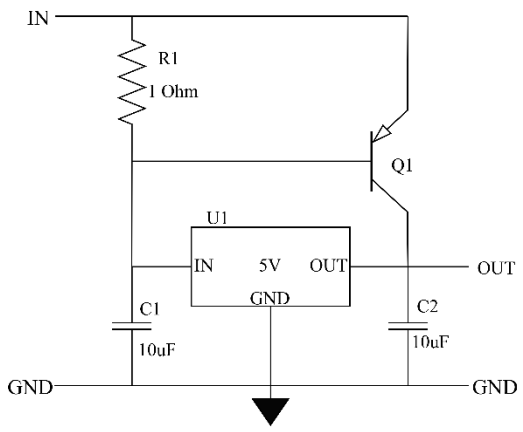


Figure 6-11: Internal Circuitry of LM7805 for Additional Current Supply

When the current requirement is below 700mA, the voltage across the resistor is low, which keeps Q1 off. However, for a current requirement of more than 700mA, the voltage across R1 increases enough to turn on the transistor Q1, then the required

additional current is provided by it. The poser dissipation of the LM7805 as heat given by (6-1). As the LM7085 starts to overheat thermal protection circuit kicks in to prevent the damage as a result current delivered to the load decreases. Reduction of current causes the output of the regulator to drop below the required voltage. The remaining LM7805 regulators try to compensate by supplying more current and they start to overheat and enter thermal shutdown.

As the regulators start to enter and leave thermal shutdown, the output voltage oscillates and may lead to the power failure. To resolve output voltage oscillation, issue a filter capacitor is used. As a filter capacitor stores the electrical energy and as soon as the voltage drops, it discharges to provide a constant voltage. The alternating thermal shutdown and overheating can only be resolved by using either a larger heat sink or a ballast resistor, but it makes the circuit complex and bulky. It was found that to make a circuit compact a compromise has to be made.

$$P = (V_{in} - V_{out}) \times I_{load} \quad \textbf{(6-1)}$$

6.7 SMPS

Heating of the LM7805 caused power failure due to which the overall circuitry was incapable of delivering the required torque to the servos. To deal with this problem an SMPS that can produce an output of 5V at a current up to 40A. The SMPS was found to be able to deliver the required current to the servos. Although, it delivered the required current, it was found that whenever there is a high current requirement the output voltage of the SMPS sharply reduces below 4.95V i.e. 1.5 times the output voltage of the AMS1117 that shutdown the ESP32 and the connection of the ESP32 with the server was broken. To resolve this issue, separate batteries with four LM7805 connected in parallel were used to power up the ESP32 CAM and ESP32. It was found that ESP32 was not able to send the control signal to servos properly due to which jittery motions were seen. This may be due to the fact that the ESP32 CAM draws current upt0 800mA when WiFi is used for data transmission and significant heating of LM7805 in parallel doesn't let it to deliver the required current. So, ESP32 and the ESP32 CAM module were powered separately through a battery connected to a 5V Buck converter. The Buck converter was able to provide the required current. Using

this technique, overall circuit was powered up without significant heating and power failure.

6.8 Distance Estimation

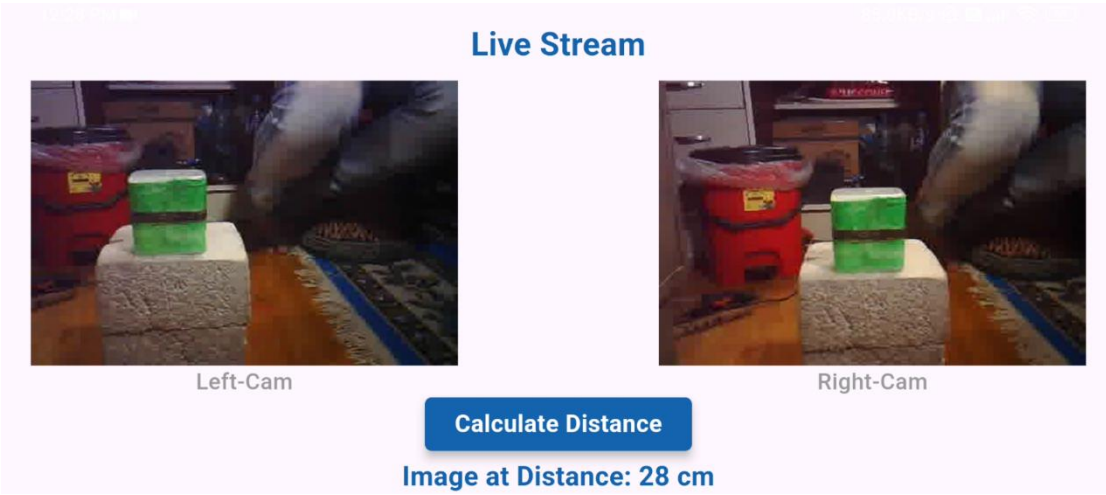


Figure 6-12: Distance Estimation of Object

As seen in Figure 6-12, the distance from the set of ESP32-Cam and the cube was calculated to be 28 cm when the cube was placed around 29 cm from the ESP32-CAM. The result was perfectly accurate for this distance.

Similarly, the distance was estimated for a range of distances from 10 cm to 70 cm from the ESP32-CAMs. The results were obtained as seen in Figure 6-13. The results were most accurate in the range of 15 cm to 60 cm. However, the estimated distances deviated from true distances when the object was placed too near (less than 10 cm) and farther than 65 cm. When the object is too near to the camera, some parts of the object's image are out of frame, causing inaccurate measurement of the object's center.

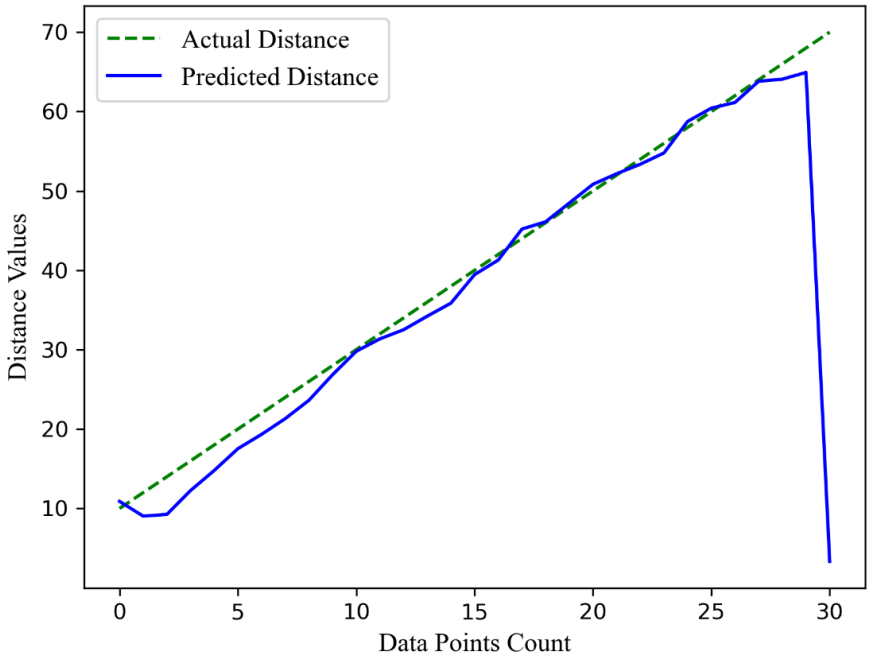


Figure 6-13: Distance Estimation Accuracy Plot

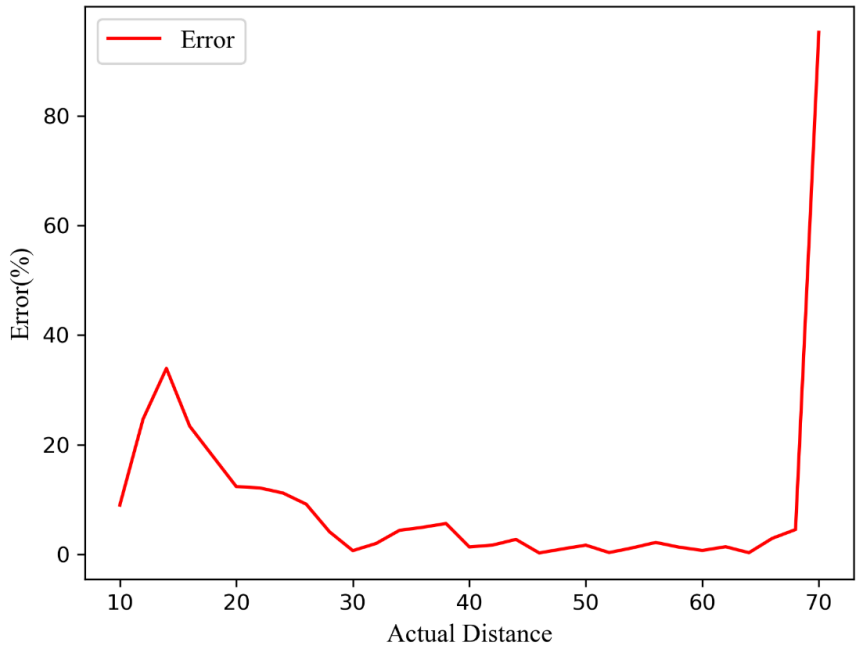


Figure 6-14: Estimation Error for Distances

The bounding box could not cover the object precisely. The average accuracy of the distance estimation without considering the last point (which showed the terrible results due to low number of pixel shift when distance is increased) is 93.37%. Considering the last point, the accuracy was obtained to be 90.52%. This shows that the distance estimation is performing quite well on the mentioned distances.

The latter one was caused due to lower resolution of ESP32-CAM, due to which the pixel shifts of bounding boxes in frames captured by the cameras when the object was far away started to be virtually unnoticeable. The confidence score of object detection also dropped when the object was placed farther. So, objects detected with low confidence score were rejected during the image processing. Hence no accurate distance estimation was done for them. When the confidence score required to was reduced to 70% from 80% in the image processing step, the objects were detected to a farther distance.

The model's confidence was decreasing on increasing the distance which can be easily seen from the confidence score recorded for 2 different distances:

For distance 30cm

Confidence1: 0.9251140356063843

Confidence2: 0.9113008975982666

For distance 60cm

Confidence 1: 0.8243626356124878

Confidence 2: 0.8143503069877625

The Confidence1 and Confidence2 refers to the confidence of the model to classify the object as a cube for images from ESP32-CAM 1 and 2 respectively. This drastic decrease in the model's confidence when distance is just increased by 30 cm caused the model to not get enough confidence for larger distances, hence failing to calculate the distance.

When the resolution of the camera was increased from HVGA to VGA, this added more pixels to the frames and hence the bounding boxes covered the cube in frames more precisely. This slightly increased the distance estimation accuracy for objects at farther distance.

6.9 Robot Movement in Simulation

6.9.1 Forward Movement

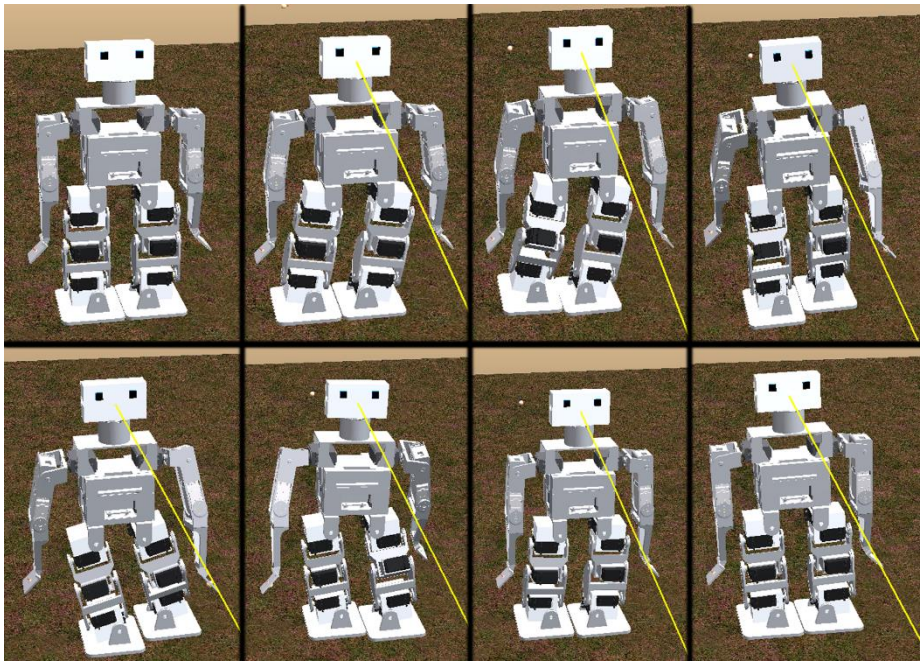


Figure 6-15: Forward Movement in Simulation

The robot has human-like stable movement when the forward movement module was simulated. The trajectory path of the left and right foot can be seen in the Figure 6-15 itself.

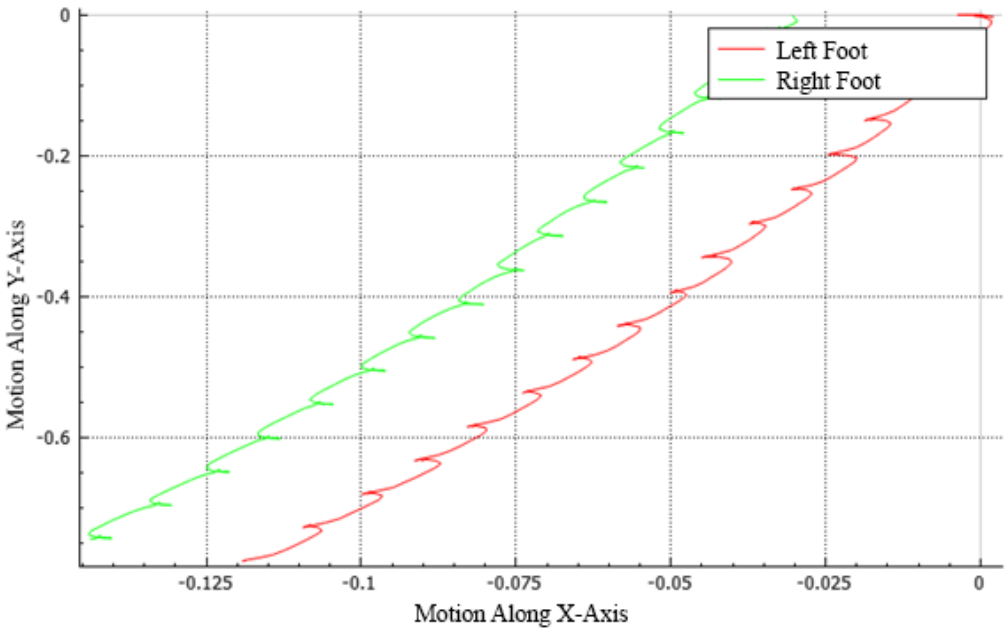


Figure 6-16: Trajectory of Left and Right Feet in Forward Movement

The graph seen in the Figure 6-16 shows the trajectory of x and y coordinates of the left and right leg of the robot model when it is walking along a path. The curve on the left side shows the trajectory of the right foot and the curve on the right side shows the trajectory of the left foot. It can be seen that the path traced by both feet are symmetric to each other, showing stability in movement. The wiggly trajectory is seen because the foot has to be moved sideways in order to balance the weight of the robot. The obtained forward movement is ideal for application.

6.9.2 Sideways Movement

The sideways movement was successfully observed for the robot. The robot shifts its weight on the leg which is not raised. The raised leg's joints are bent sideways to cover a certain distance on sideways and the upper body is back to its initial position as seen in Figure 6-17.

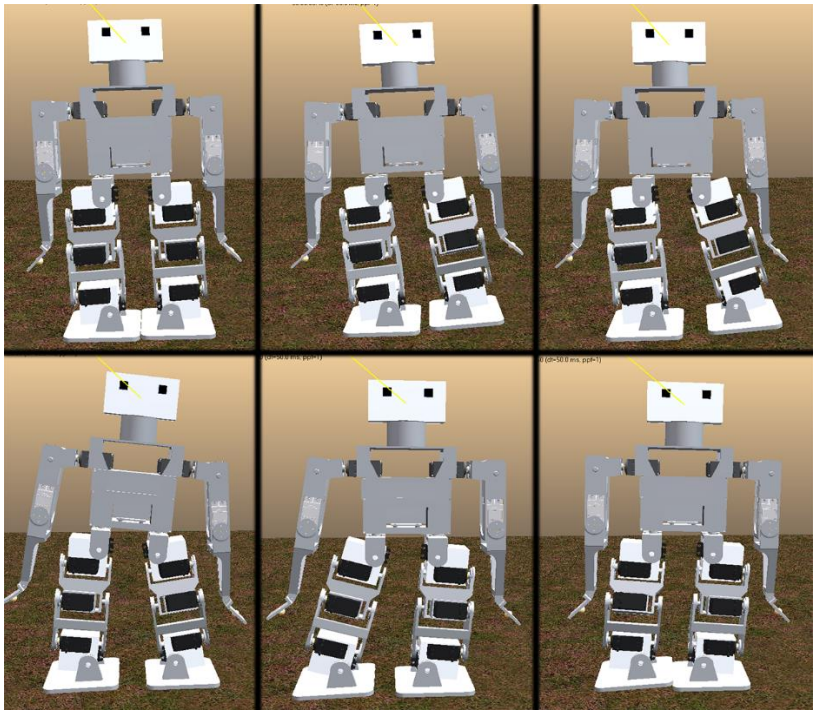


Figure 6-17: Sideways Movement in Simulation

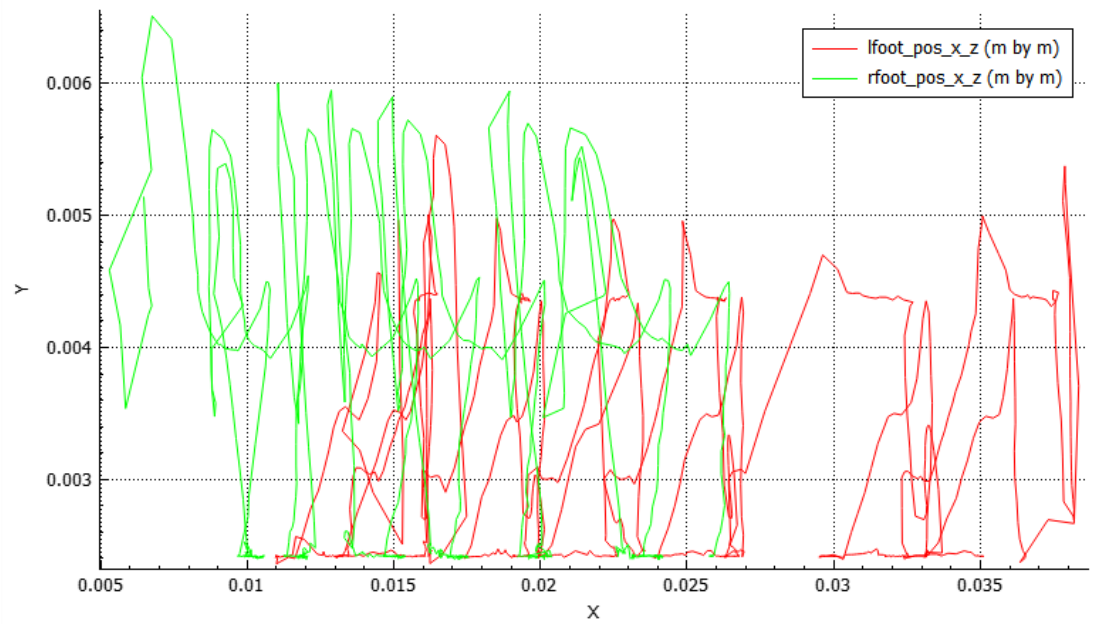


Figure 6-18: Trajectory of Left and Right Feet in Left Sideways Movement

As seen from the graph of the trajectory of the left and right foot in X and Z directions when moving in the left direction in Figure 6-18, the curves are not perfectly smooth. It is because of friction between the feet and the ground causing a sort of vibrating effect when one leg is raised completely. It is also seen that the right foot curve reaches a bigger height than the left foot. The body weight is first shifted towards the right foot when moving in the left direction. However, it does so only for a small amount of time so that the robot body does not apply too much weight on one leg otherwise it would lose balance and fall over. When moving in the left direction, more force is applied in the left direction. So, the robot gets pushed, causing the right foot to spend greater time in air than the left foot, and hence the curve of the right foot has greater height.

The “grazing” curve at the bottom of the graph is due to sliding of the left foot on ground when moving towards the left direction. When the robot tries to lift its right foot, the body has to be tilted in the left direction. Due to the force applied to move the upper body on the left, an opposite reaction is seen on the left foot, causing it to slide in the left direction.

6.9.3 Pick and Drop Movement

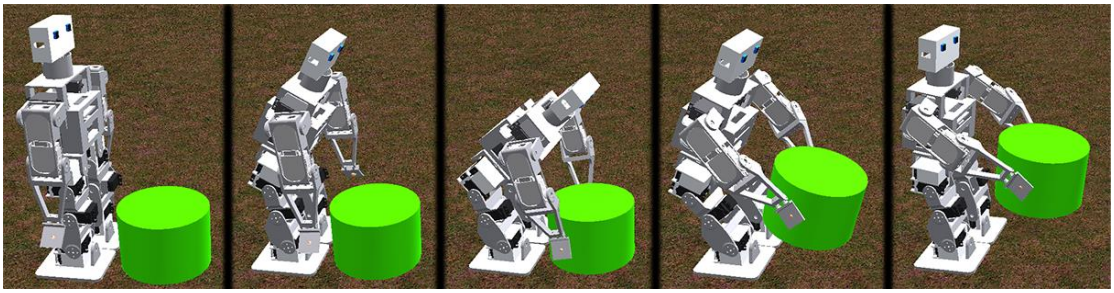


Figure 6-19: Pick Low Object Motion

The robot successfully picked up a cubical object at ground as seen from Figure 6-19. The excessive weight at the robot's waist caused no trouble to pick up the object in simulation. It bent and reached out its arms all way to the object at ground and picked up by balancing the mass using leg motors.

However, due to excess weight of the upper body of the robot, excessive bending caused the robot to fall on its front as seen from physical testing outside simulation. Once again, balancing the center of mass was necessary. Due to this constraint, only picking up object at certain height seemed feasible. So, an object was placed on a certain height, and picked up as seen from Figure 6-20. Using the inverse procedure, drop motion was also deduced.

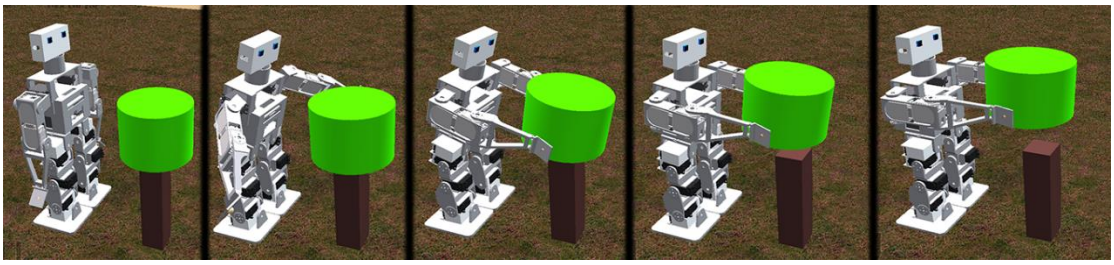


Figure 6-20: Pick Object at Elevated Position

The weight of the object also proved to be an important factor to be considered while implementing pick and drop motion. It was found that picking up heavier objects caused significant disruption in center of gravity of the robot. So, to mitigate this effect, light object was used.

6.9.4 Rotate Motion

Rotating the motion involved rotating the yaw joint of the legs as seen from Figure 6-21. Once clockwise direction rotating motion was implemented, anti-clockwise rotation was also achieved by mirroring the clockwise rotation. The amount of rotation of the whole body directly depended on the rotating angle of yaw joint. The degree of rotation of robot body was found to be almost the same as the degree of rotation of yaw joint of leg.

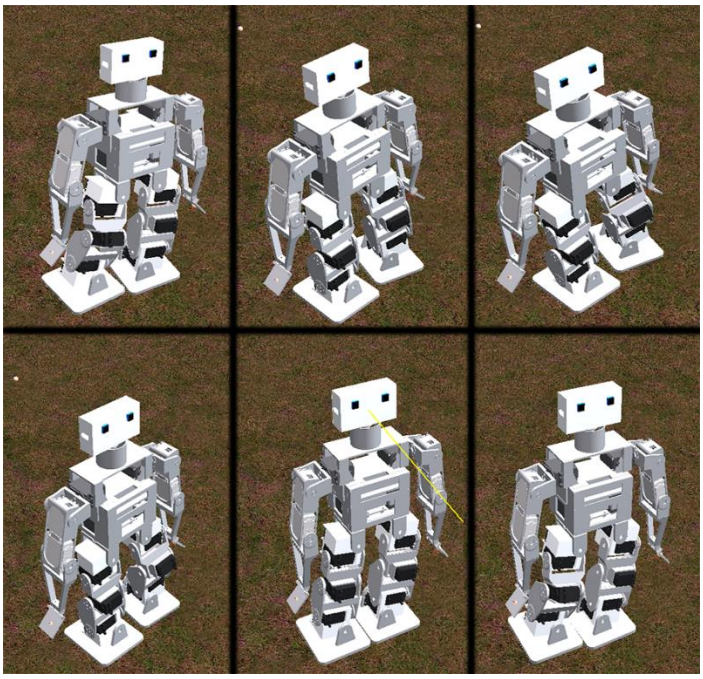


Figure 6-21: Anti-Clockwise Rotation in Simulation

6.9.5 Fall Recovery

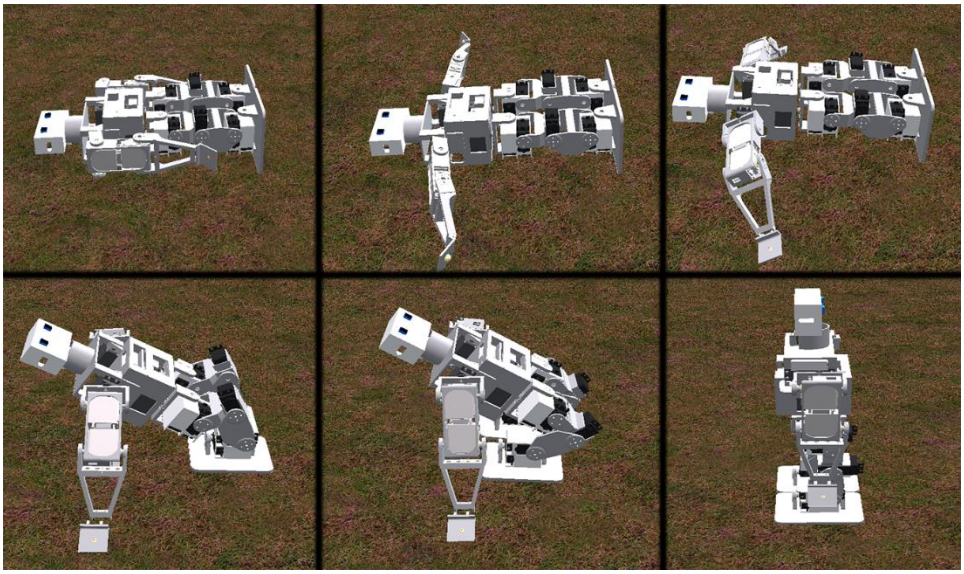


Figure 6-22: Fall Recovery Facing Up

When the robot was laid on its back facing upwards, it executed fall recovery by first spreading its hands wide apart, then pushing the body upwards. The hands aided the feet in getting up as seen in Figure 6-22.

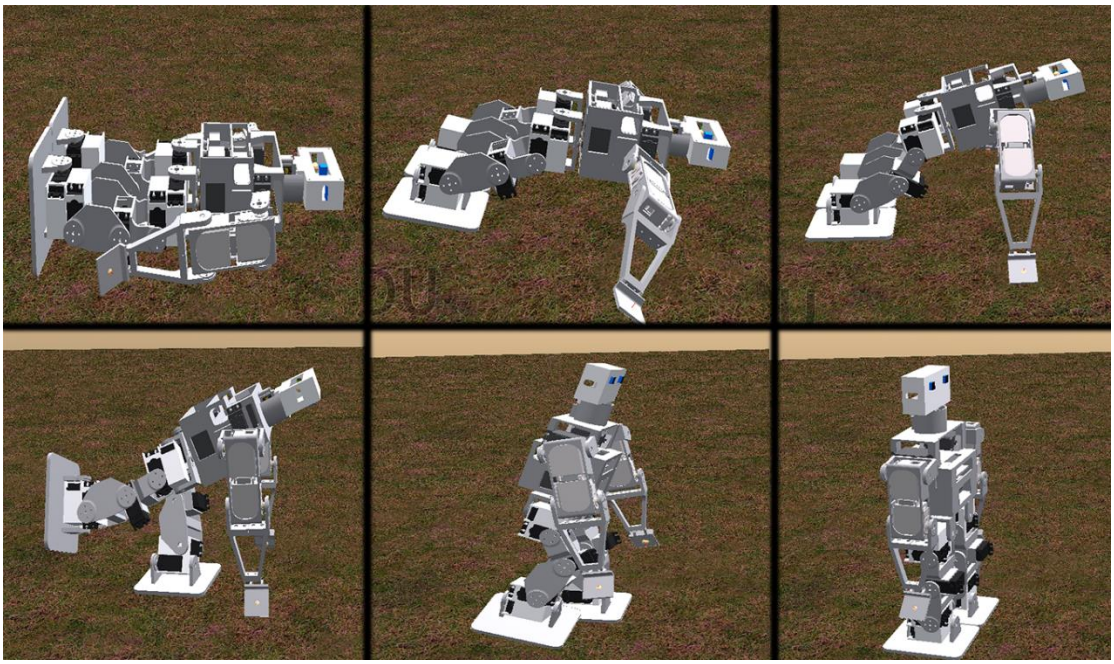


Figure 6-23: Fall Recovery Facing Down

When the robot was facing down when laid, it again used the help of hand joints to get up. But this time, to balance out the center of mass, it also used the right leg to spread backwards. Then it gradually stood up to stable position.

This simulation was based on the Pybullet 2.78 physics engine. However, the body produces unstable motion when changing to another physics engine, which is due to the disparity in friction and contact model that each physics engine utilizes.

6.10 Robot Assembly

The proposed and designed humanoid robot was successfully 3D printed and assembled Figure 9-7. All the robot's parts were perfectly fit through various iterations and design changes to achieve expected dynamics and range of motion after changing the design using SG90 motor with metal gear Mg996R motor and also iterating over the design to solve gear stripping issue with SG90 servo and torque limitation with metal gear servo motors. The initially designed robot is seen in Figure 9-5, also posed space limitations for all the components. Beside this, the battery used can deliver enough current required by all the motors and other electronic components but, due to limitation of voltage regulation circuit and rapid degradation of LM7805 voltage regulator, external power supply (SMPS) used is able to deliver enough power to the motors solving the limitation of voltage regulation circuit and an additional buck converter to power sensitive electronics like: Esp32 and ESP32 Cams drastically improved the performance of each component, essentially solving all major issues.

As a result of iterative designs, the final robot designed and fabricated uses 18 metal gear servo motors (MG996R) for all the major joints and 1 plastic gear micro servo (SG90) for the neck Figure 9-7. All the required control circuits and microcontroller are housed in the torso and two Esp32 Cams are housed by the head with proper mass distribution and dynamics. The final robot designed and assembled is successfully able to perform needed motions when required and sense its environment using the Cameras for stereo vision. It can perform various maneuvers autonomously to navigate to the object and manipulate it.

6.11 Motions in Physical Robot

6.11.1 Motor Initialization

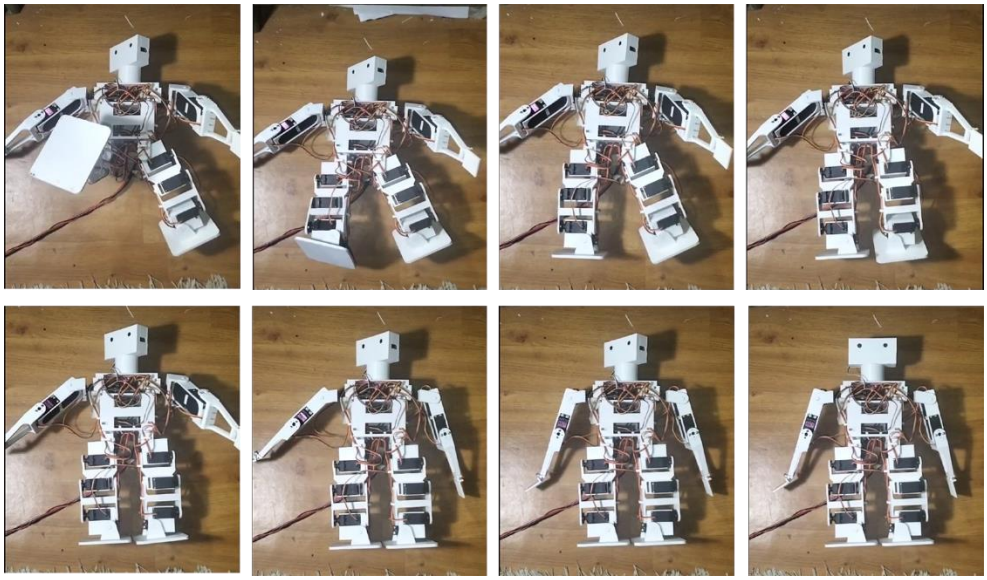


Figure 6-24: Motor Initialization in Robot

The steps of motor initialization are shown in Figure 6-24. During initialization, all the motor moves one after another to a predefined position to obtain a stable initial pose for the robot. The steps were taken one at a time every second to create a stable initial standing position for the robot as shown. This step ensures that the next motion or the sequence of motions to be executed are as expected and stable.

6.12 Walking Sequence

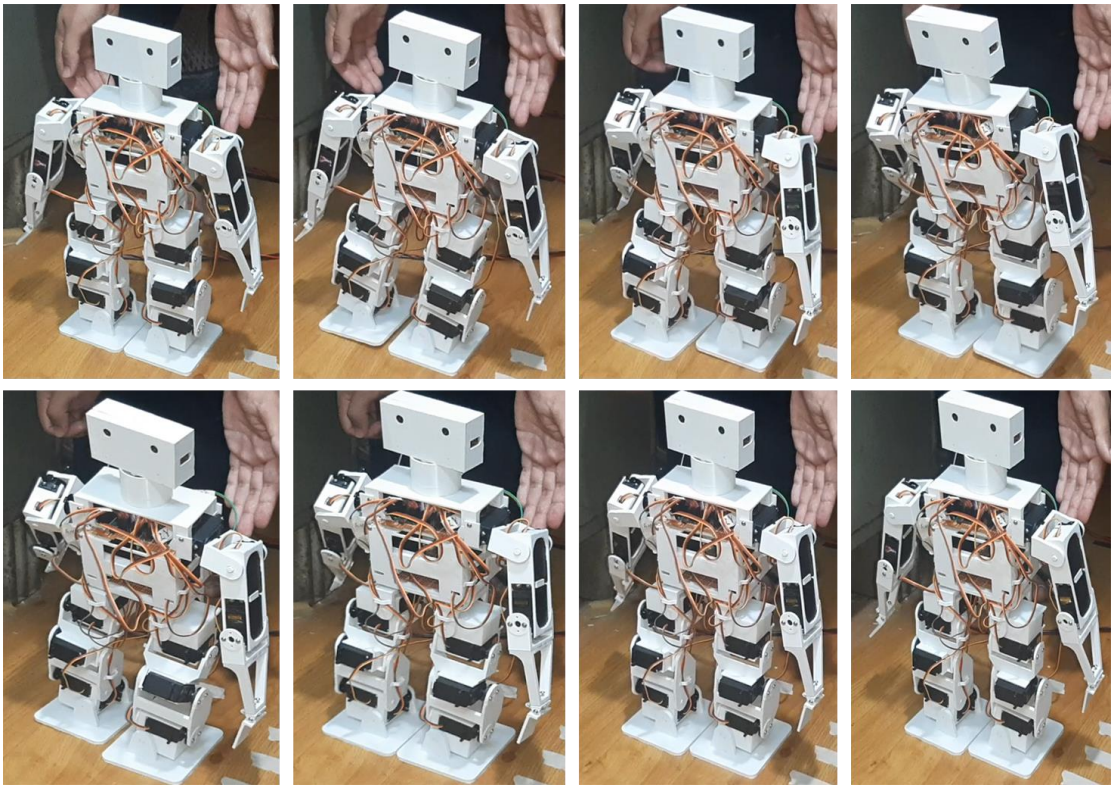


Figure 6-25: Walking Sequence of Robot

The motion of the legs in the physical robot when static was found to be very close to that of simulation but due to differences in the physics and dynamics of body parts and joints in simulation software/environment and real world, drastic changes to motion data had to be made to achieve stable walk motion with stable transitions between frames. After adjusting the motion data, the robot moved with stable posture which can be seen in Figure 6-25 as well. While walking, the robot first shifts its whole weight to one leg and moves another to cover a certain distance in forward direction and the same was repeated for another leg to move another step as well completing one complete walking cycle. A stabilizing frame introduced between each cycle ensures smooth and stable transition between cycles. Multiple walking cycles were repeated to cover a longer distance leading to proper implementation of walk motion in the physical robot. This motion is used to close the distance between the object and robot.

6.13 Left/Right Turn

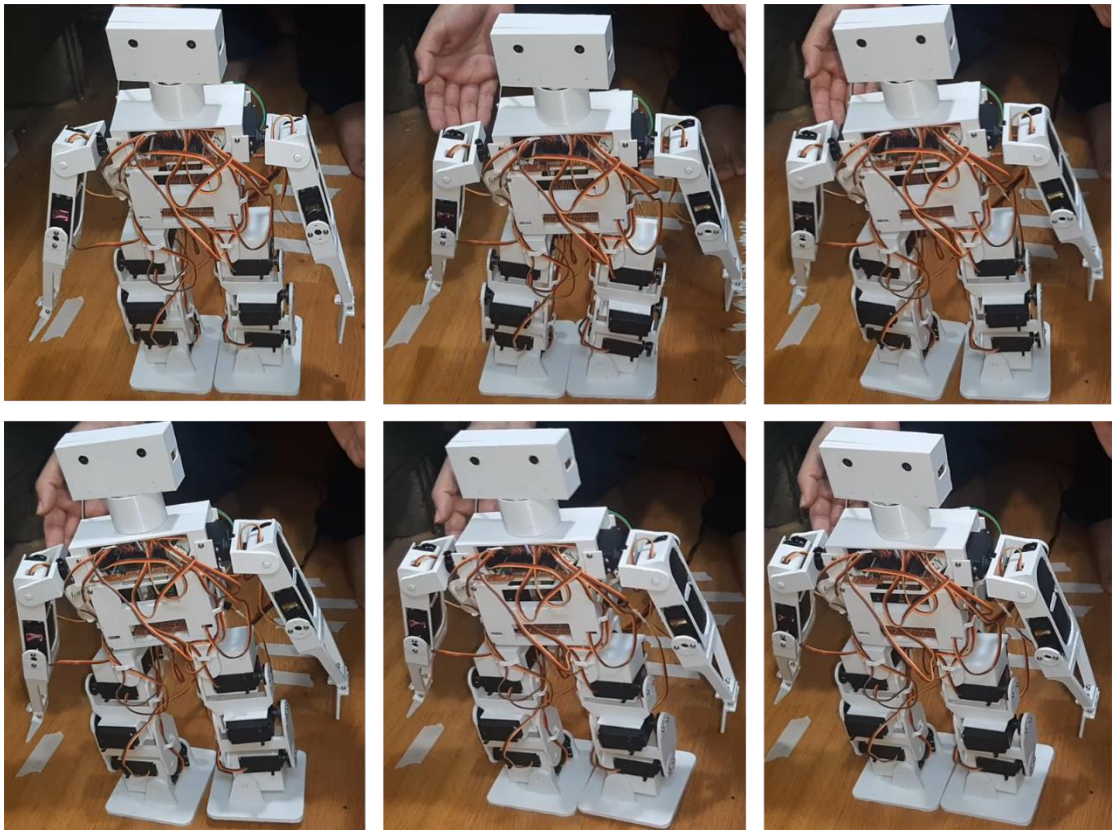


Figure 6-26: Rotating Motion of Robot

The turning of the robot was also successfully implemented in the physical robot with some changes in the frame data obtained from simulation. While turning, the robot properly shifts its weight on one leg and rotates another leg by a certain amount in one direction using Hip Yaw joint and then balances out by putting down all legs causing the robot to turn in one direction. Same but mirrored data was applied for turning in another direction to create turning motion in clockwise and counter clockwise direction. Multiple turning motion cycles were repeated for a larger turning angle. This motion allows the robot to turn in the direction of the object's presence.

6.14 Side Movement

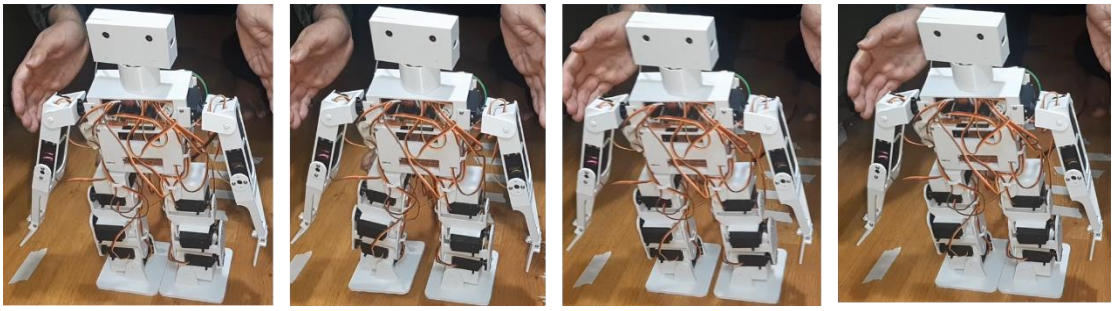


Figure 6-27: Sideways Movement of Robot

The sideways movement (Left and Right) was also properly and successfully implemented in the robot with some change in the motion data obtained from simulation. Using the sideways movement data from simulation, adjustments were needed and the robot moved in both left and right direction with stable posture. While walking sideways, the robot first shifts with whole weight on one leg using Foot Roll and moves another leg in inward direction using Hip Roll in any one direction (Left or Right) after which it readjusts its posture back to initial position completing one step in sideways direction. Multiple steps were repeated to cover a larger distance in any direction. This motion is only executed if the object is not center aligned allowing the robot to align itself with the object in the center of its view for effective object manipulation.

6.15 Pick/Drop Movement

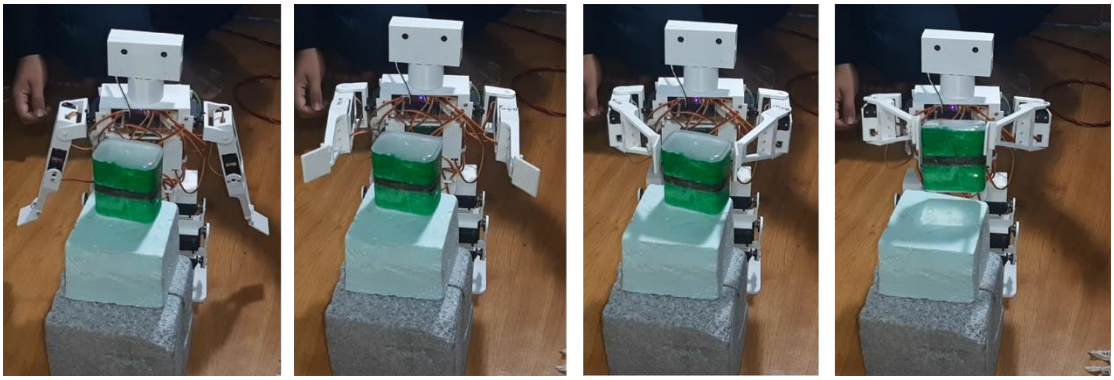


Figure 6-28: Pick Up Motion

The final motion that was successfully implemented in the physical robot is Pick/Drop motion. This motion also worked using motion data obtained from simulation but with some dynamics issues which worked with some modification in motion data. In the

pick/drop motion implemented in simulation, the robot bends a little to pick the object but in the physical robot when the same motion was implemented, the robot successfully executed the motion but after picking the object the robot became unstable. To avoid this, the bending of the robot body was eliminated to achieve stable pick and drop of the object at every stage. With this change, the robot picks the object while standing and drops in the same location as shown in the Figure. This motion allows the robot to pick and drop the cube (trained object) after the robot reaches the destination.

6.16 Navigation Towards the Object

When the object was not in visual range of the robot, the head of the robot turned back and forth to scan the environment for object detection. When it finally detected, the object, it executed rotating motion to face towards the object. An object placed at about 40 degrees from the line of sight of robot made the robot execute rotating motion four times, with average rotating angle of 12 degrees.

The robot walked and covered almost consistent distance on each walk cycle. No drastic difference was observed in each experiment of motion cycles. However, since the walk cycle is sensitive to the nature of the floor, the coverage is affected when the part of floor is uneven. But, as described in section 4.3.16, the minor inconsistencies are recovered by closed loop control system.

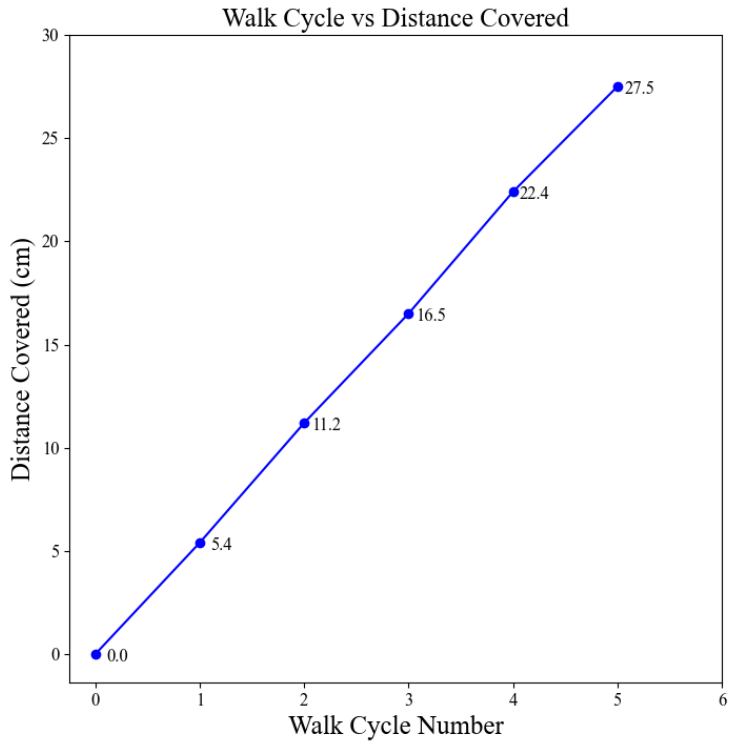


Figure 6-29: Distance Covered on Each Walk Cycle

Figure 6-29 shows that the robot covers almost 30 cm after execution of 5 walk cycles. The average distance coverage is calculated to be about 5.5 cm per walk cycle. The fluctuation itself rarely goes above 1.5 cm. The net error in distance is also about 2-3 cm which is not that significant.

The alignment control system was effective in placing the object on the center of vision of the robot. When the object was at left or at right of the view, the robot executed sideways motion to make it at the center of its view as seen in Figure 6-30.

When the object was aligned to the center, then it again started its forward movement cycle. This process continued until it reached the threshold minimum distance from the object to execute pick up motion.

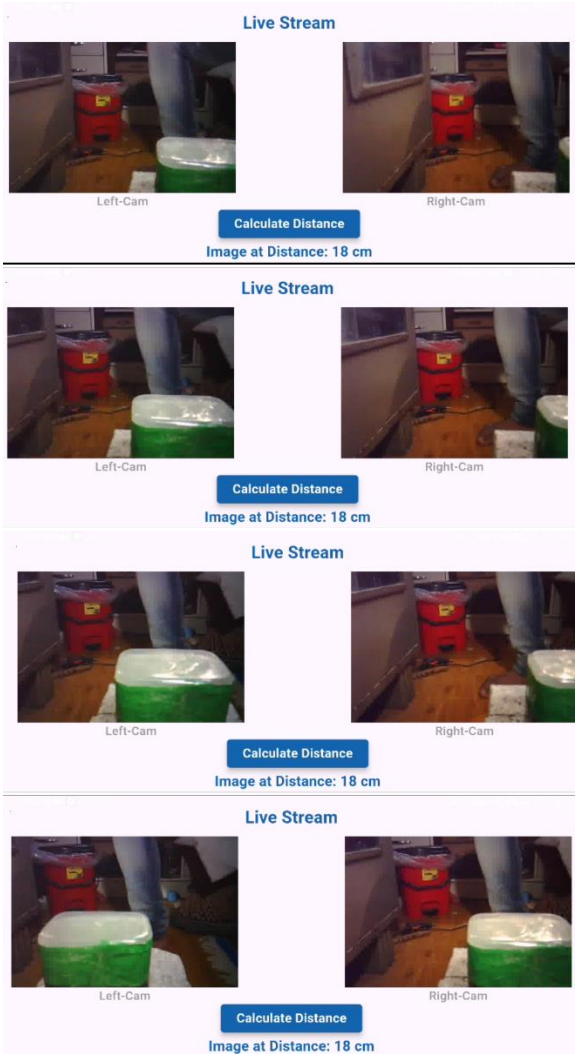


Figure 6-30: Alignment towards Object

6.17 Pick/Drop Evaluation

When the robot reached the location of the object (cube) by navigating visually, the robot successfully picked the object with the help of pre-defined pick/drop motion. The robot was able to achieve this by setting a condition in its control system to pick the object when the distance calculated from the stereo camera setup becomes less than 15 cm. When the condition was satisfied, the robot moved one step more to close the distance between the object and the robot to account for error in distance estimation.

Since, the robot does not account for the distance between the object and the robot below 15 cm, the robot overshoots and undershoots the picking of the cubical object each time. Also, due to imperfect design of the hand, the robot has difficulty in picking the object. This is due to lack of gripping strength when picking. When an object was

placed in a corner of a room or aligned with a wall beside it, the robot collided with the wall and could not reach the object to pick it since it cannot detect obstacles. Also, when the cubical object was placed with its edge facing the robot, the robot had difficulty picking the object due to reduced surface area. The dropping of the motion was also jerky and the object was dropped at the same location. The picking and dropping of the cubical object have a lot of inconsistencies due to the number of variations and deviations in calculations throughout multiple runs but, the robot was successfully able to manipulate the object consistently in each run.

6.18 Simulation vs Real World Comparison

	Physical	Simulation
Motor Torque	4.8 kg cm	5 kg cm
Fall recovery	Not implemented	Implemented
Distance estimation using camera sensor	Below 15m accuracy was Low	Highly accurate
Rotation covered per turn	12 degrees	13 degrees
Pick Up motion	Cannot bend to pick object	Can bend to pick the object
Walking Stability	Transition Frames added for stability	No transition frames used
Smoothness	4ms delay after each forward motion to estimate distance	No delay required to estimate distance
Visual Sensor	Stereo Camera setup	Vision Sensor
Real time streaming	May lag	No lag

7. FUTURE ENHANCEMENTS

The robot has dynamic capability only in the process of making decision of which motions to take based on the position of the object which it is programmed to grab. But all of its motions are based on frame data which directs the robot to take static motion for forward-walk, side-walk, rotation movement and cannot adapt based on the environment/surface it is walking on. So, with addition of sensors like FSR sensors for dynamic ZMP calculations and use of Reinforcement learning for adapting to the dynamic environment conditions and terrains can increase the usability of the robot in industrial environment. The ESP32 cam performs well but using a more resolution camera module can lead to better object detection and distance estimation to a further distance. Also, the robot has no sense of the orientation of the object that it is grabbing. So, the YOLO models with OBB (Oriented Bounding Boxes) capabilities can be adopted to figure out the orientation of the object. The motions that are required to align the object in the specific orientation can be the same motions as developed in this project.

8. CONCLUSION

A humanoid robot of small scale is fabricated using the prototyping hardware and the static motions mimicking the common motions of human are implemented. The ESP32-CAMs as stereo camera setup performed quite well although being the low-level prototyping board. The formulation and approach used for distance estimation was also accurate but its efficiency and accuracy degraded with the distance due to decreasing pixel offset between object detected from two cameras. The use of servo motors however caused a lot of problems in the process motion planning and overall motion of robot. The small wobbliness in the static position of the servo motors caused the motions to be inconsistent during the overall control system. The MG996R motors were not able to provide the rated torque. The torque observed was less than half of the rated torque which caused the motion data from the simulation to not work when directly implemented in the physical robot. The simulation data however provided the overview of how the joints should be manipulated for the desired motion which acted as the base for designing the physical motions. Thus, the use of simulation tools can be very useful for designing motions in these types of configurations having many joints.

9. APPENDICES

Appendix A: 3D Render of Humanoid Robot

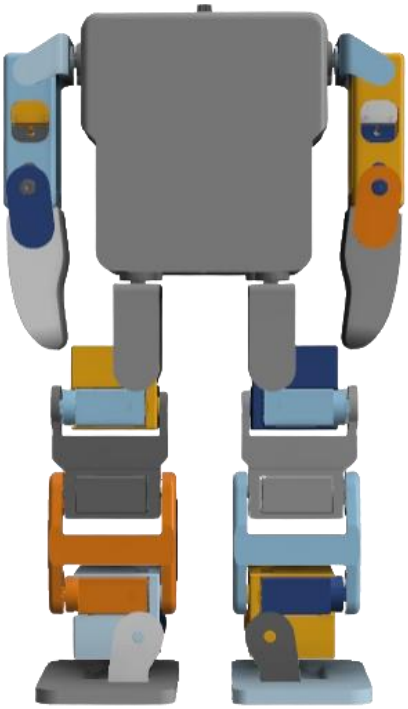


Figure 9-1: 3D Render of Humanoid Robot (Front View)

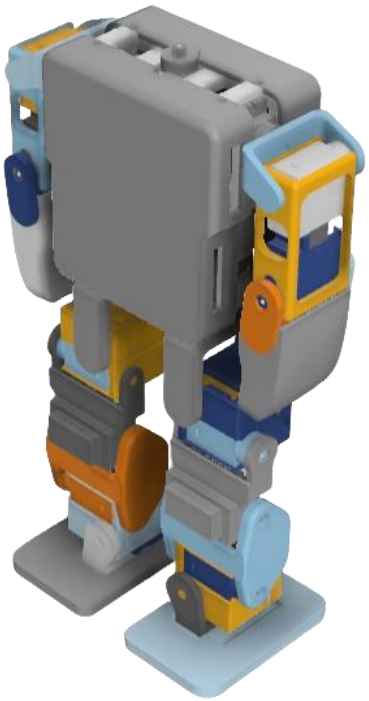


Figure 9-2: 3D Rendered Isometric View of Humanoid Robot

Appendix B: Robot Model using MG996R Servo Motors

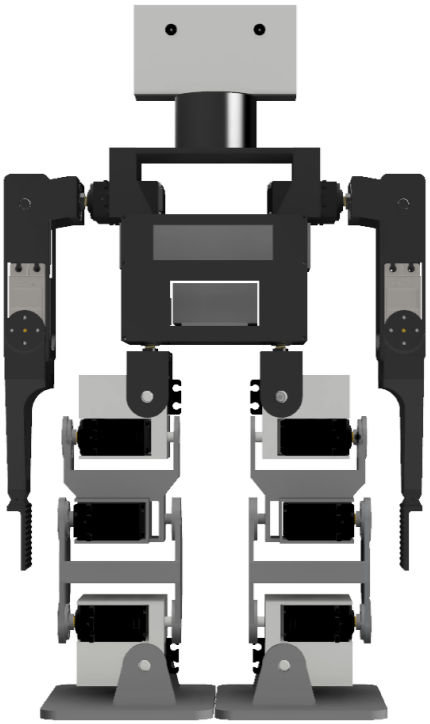


Figure 9-3: Front View of Robot Model

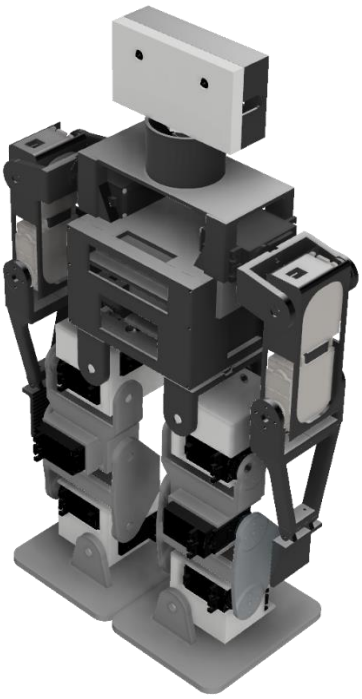


Figure 9-4: Side View of Robot Model

Appendix C: Robot Assembly using SG90 Motors

Figure 9-5: Assembled Robot Body

Appendix D: Robot Assembly Built from MG996R Motors

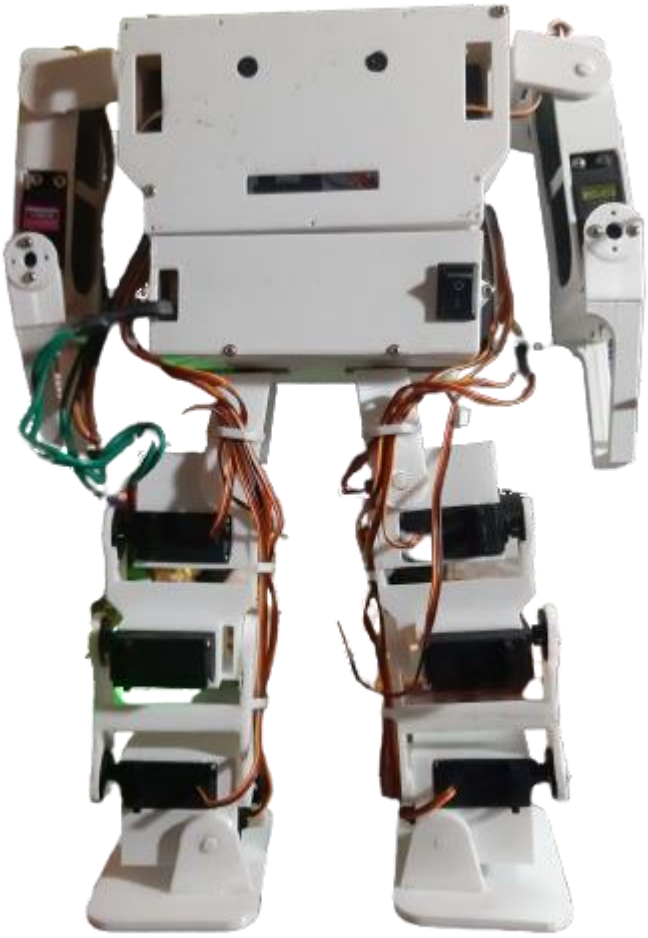


Figure 9-6: Robot Assembled using MG996R Motors

Appendix E: Externally Powered Robot Final Design

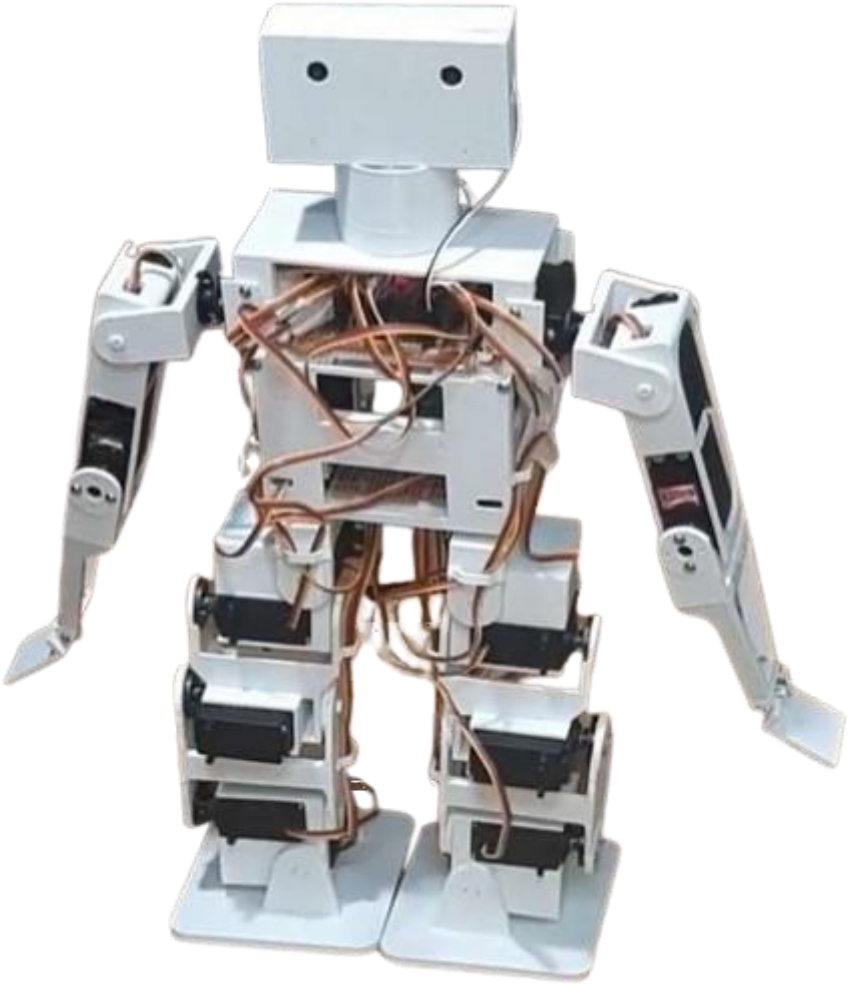


Figure 9-7: Externally Powered Design

Appendix F: Circuitry

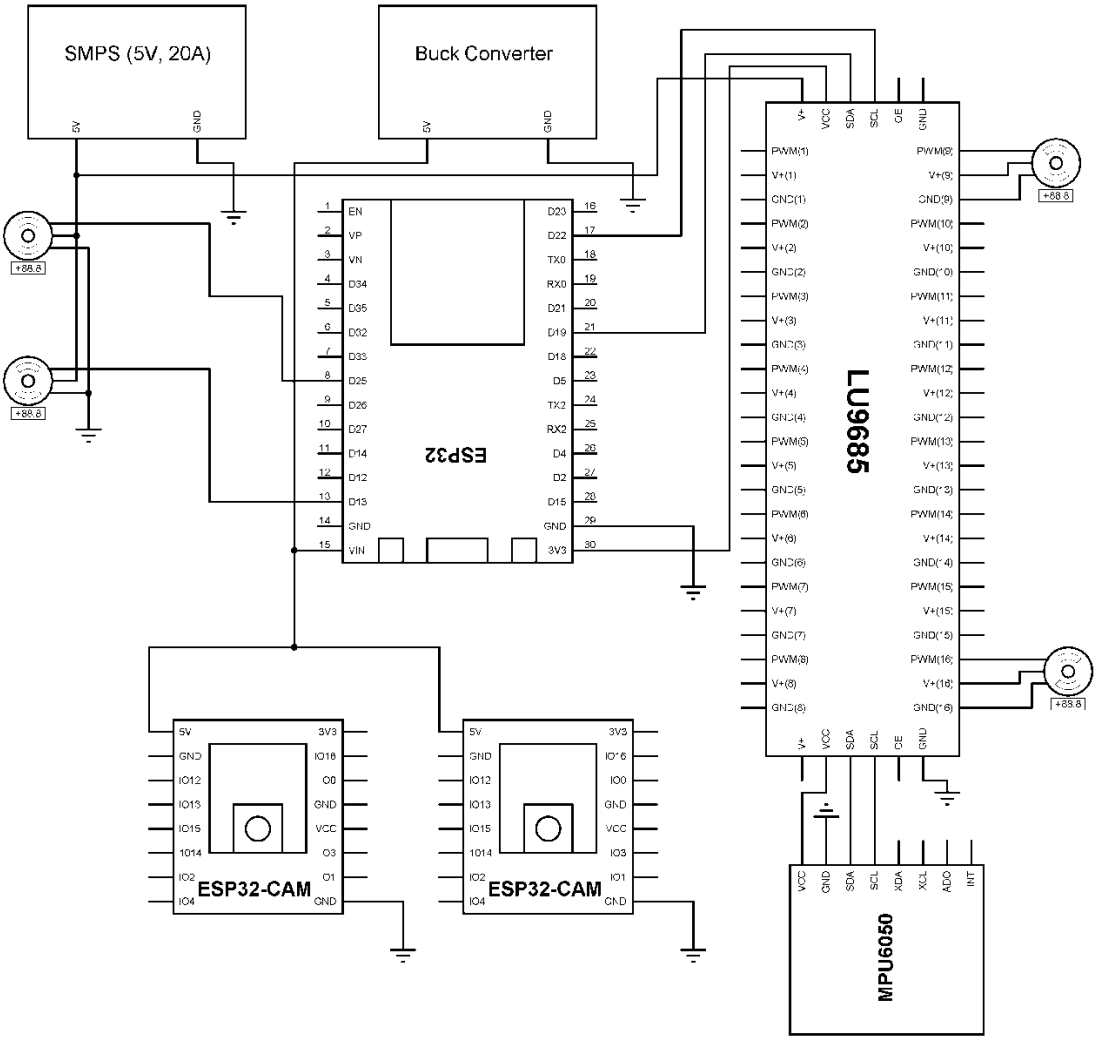


Figure 9-8: Overall Circuit Diagram

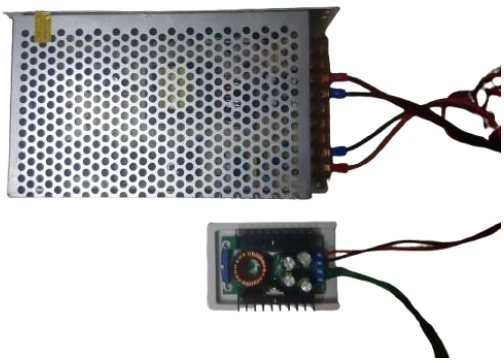


Figure 9-9: SMPS and Buck Converter Power Supply Unity

Figure 9-10: 3S LiPo Battery with Balanced Chargers

Figure 9-11: Battery with B3 Charger

Figure 9-12: LM7805 Voltage Regulation Circuit

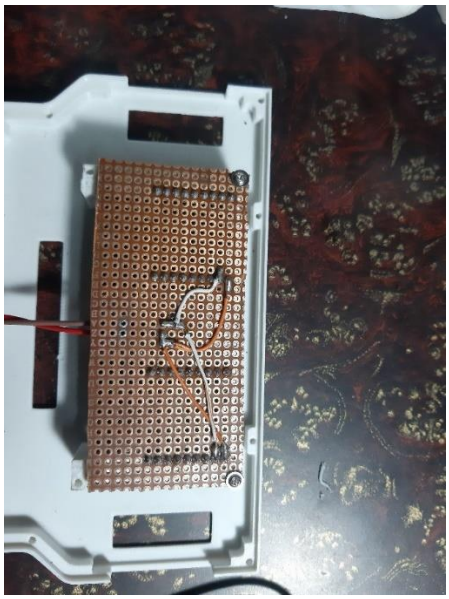


Figure 9-13: Circuitry of ESP32-CAMs

Appendix G: Torque Measurement

Figure 9-14: Weight of Test Object

Figure 9-15: Torque by Motor on Test Object

Figure 9-16: Torque Measurement

Appendix H: Resolutions of ESP32-CAM

Table 9-1: Resolutions of ESP32-CAM

Macro	Dimensions	Pixels	Typical Use Case
FRAMESIZE_96X96	96x96	9,216	Low-resolution testing, icon detection
FRAMESIZE_QQVGA	160x120	19,200	Basic object detection, low-power vision
FRAMESIZE_128X128	128x128	16,384	Face detection (basic models)
FRAMESIZE_QCIF	176x144	25,344	Small object recognition
FRAMESIZE_HQVGA	240x176	42,240	Simple video streams with low latency
FRAMESIZE_240X240	240x240	57,600	Square frame for ML applications
FRAMESIZE_QVGA	320x240	76,800	Real-time streaming & object detection
FRAMESIZE_320X320	320x320	102,400	ML vision tasks with TensorFlow Lite
FRAMESIZE_CIF	400x296	118,400	Surveillance & monitoring
FRAMESIZE_HVGA	480x320	153,600	Moderate streaming performance
FRAMESIZE_VGA	640x480	307,200	High-quality object detection
FRAMESIZE_SVGA	800x600	480,000	High-quality streaming
FRAMESIZE_XGA	1024x768	786,432	High-resolution images (PSRAM needed)
FRAMESIZE_HD	1280x720	921,600	HD video streaming (PSRAM critical)
FRAMESIZE_SXGA	1280x1024	1,310,720	High-detail applications (max load)
FRAMESIZE_UXGA	1600x1200	1,920,000	Max resolution for OV2640 sensor

Appendix I: Physical Specifications of Humanoid Robot

Table 9-2: Mass Specification

Parts	Mass (gram)
3D printed frame	800 gm
MG996R - Motors (18)	(50 x 18) =900 gm
SG90 Servo Motor (1)	10 gm
ESP32 Dev Module	10 gm
ESP32-Cam (2)	(10 x 2) = 20 gm
MPU6050	5 gm
LU9685	10 gm
Screws	50 gm
Total	**1805 gm**

Table 9-3: Height and Width Specifications

Measurement Parameters	Measurement (mm)
Height	37 cm
Width (Front)	24 cm
Width (Side)	9.7 cm

Appendix J: Mobile Application User Interface

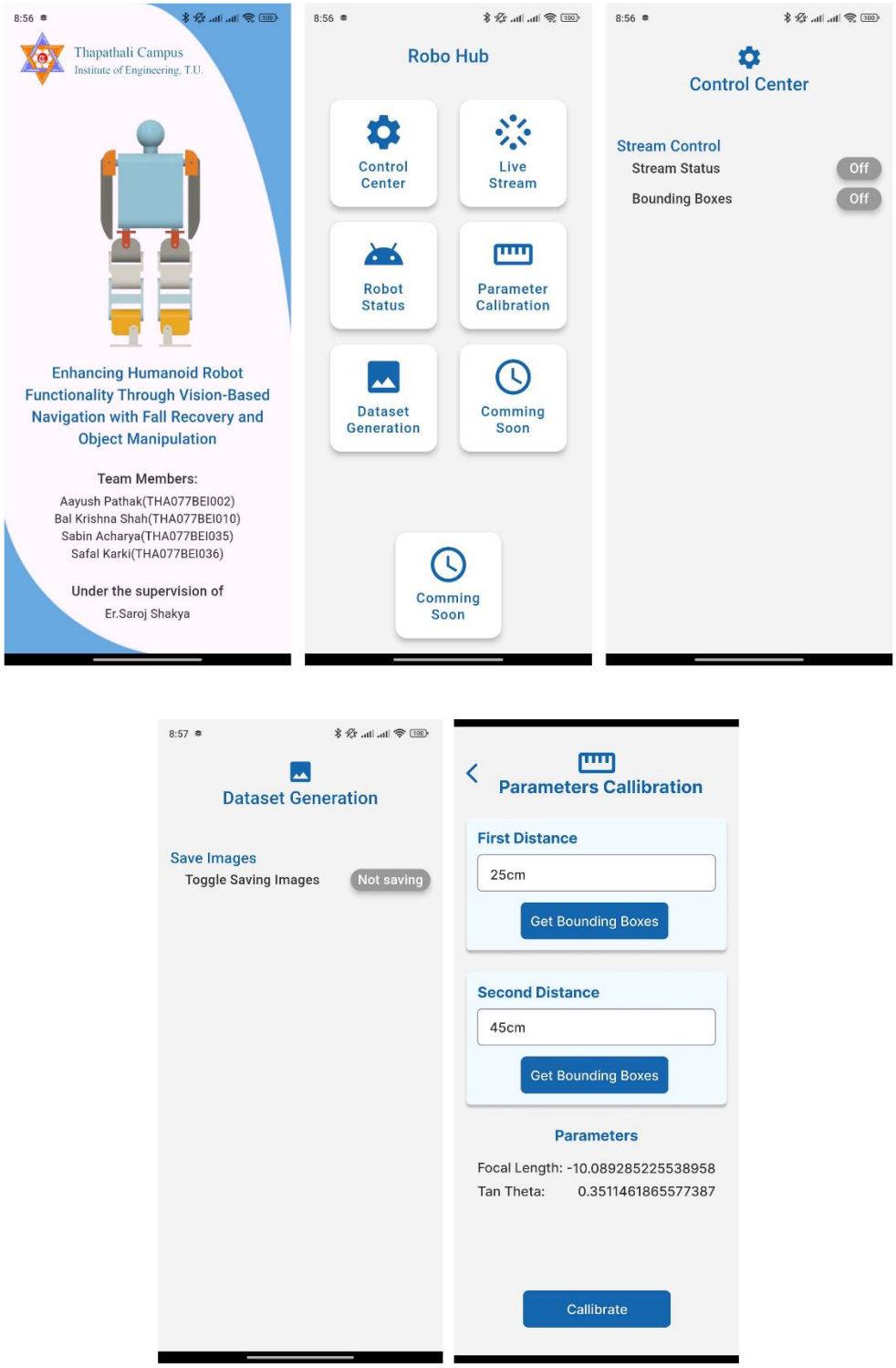


Figure 9-17: User Interface of Mobile Application

Appendix K: Source Code for Smooth Inter-Frame Interpolation

```
#define stepTime 50
Robot currentRobotState = Robot(0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0);
void moveToFrame(int frameNumber)
{
    Robot nextRobotState = Robot(walkingFrames[frameNumber]);
    float time = walkingTime[frameNumber];

    float numIter = time * 1000 / stepTime;

    // Calculating step value for each iteration.
    float leftShoulderPitchStep =
(nextRobotState.leftShoulderPitch() -
currentRobotState.leftShoulderPitch()) / numIter;
    float leftShoulderRollStep =
(nextRobotState.leftShoulderRoll() -
currentRobotState.leftShoulderRoll()) / numIter;
    float leftArmStep = (nextRobotState.leftArm() -
currentRobotState.leftArm()) / numIter;

    float rightShoulderPitchStep =
(nextRobotState.rightShoulderPitch() -
currentRobotState.rightShoulderPitch()) / numIter;
    float rightShoulderRollStep =
(nextRobotState.rightShoulderRoll() -
currentRobotState.rightShoulderRoll()) / numIter;
    float rightArmStep = (nextRobotState.rightArm() -
currentRobotState.rightArm()) / numIter;

    float leftHipYawStep = (nextRobotState.leftHipYaw() -
currentRobotState.leftHipYaw()) / numIter;
    float leftHipPitchStep = (nextRobotState.leftHipPitch() -
currentRobotState.leftHipPitch()) / numIter;
    float leftHipRollStep = (nextRobotState.leftHipRoll() -
currentRobotState.leftHipRoll()) / numIter;
    float leftKneeStep = (nextRobotState.leftKnee() -
currentRobotState.leftKnee()) / numIter;
    float leftAnkleStep = (nextRobotState.leftAnkle() -
currentRobotState.leftAnkle()) / numIter;
    float leftFootStep = (nextRobotState.leftFoot() -
currentRobotState.leftFoot()) / numIter;

    float rightHipYawStep = (nextRobotState.rightHipYaw() -
currentRobotState.rightHipYaw()) / numIter;
    float rightHipPitchStep = (nextRobotState.rightHipPitch() -
currentRobotState.rightHipPitch()) / numIter;
    float rightHipRollStep = (nextRobotState.rightHipRoll() -
currentRobotState.rightHipRoll()) / numIter;
```

```
    float rightKneeStep = (nextRobotState.rightKnee() -
currentRobotState.rightKnee()) / numIter;
    float rightAnkleStep = (nextRobotState.rightAnkle() -
currentRobotState.rightAnkle()) / numIter;
    float rightFootStep = (nextRobotState.rightFoot() -
currentRobotState.rightFoot()) / numIter;

    float headStep = (nextRobotState.head() -
currentRobotState.head()) / numIter;
    float torsoStep = (nextRobotState.torso() -
currentRobotState.torso()) / numIter;

    for (int j = 1; j <= numIter; j++)
    {
        // For Left Arm
        moveStep(leftShoulderPitchMotor,
static_cast<int>(currentRobotState.leftShoulderPitch() +
leftShoulderPitchStep * j));
        moveStep(leftShoulderRollMotor,
static_cast<int>(currentRobotState.leftShoulderRoll() +
leftShoulderRollStep * j));
        moveStep(leftArmMotor,
static_cast<int>(currentRobotState.leftArm() + leftArmStep * j));

        // For Right Arm
        moveStep(rightShoulderPitchMotor,
static_cast<int>(currentRobotState.rightShoulderPitch() +
rightShoulderPitchStep * j));
        moveStep(rightShoulderRollMotor,
static_cast<int>(currentRobotState.rightShoulderRoll() +
rightShoulderRollStep * j));
        moveStep(rightArmMotor,
static_cast<int>(currentRobotState.rightArm() + rightArmStep *
j));

        // For Left Leg
        moveStep(leftHipYawMotor,
static_cast<int>(currentRobotState.leftHipYaw() + leftHipYawStep
* j));
        moveStep(leftHipPitchMotor,
static_cast<int>(currentRobotState.leftHipPitch() +
leftHipPitchStep * j));
        moveStep(leftHipRollMotor,
static_cast<int>(currentRobotState.leftHipRoll() +
leftHipRollStep * j));
        moveStep(leftKneeMotor,
static_cast<int>(currentRobotState.leftKnee() + leftKneeStep *
j));
        moveStep(leftAnkleMotor,
static_cast<int>(currentRobotState.leftAnkle() + leftAnkleStep *
j));
```

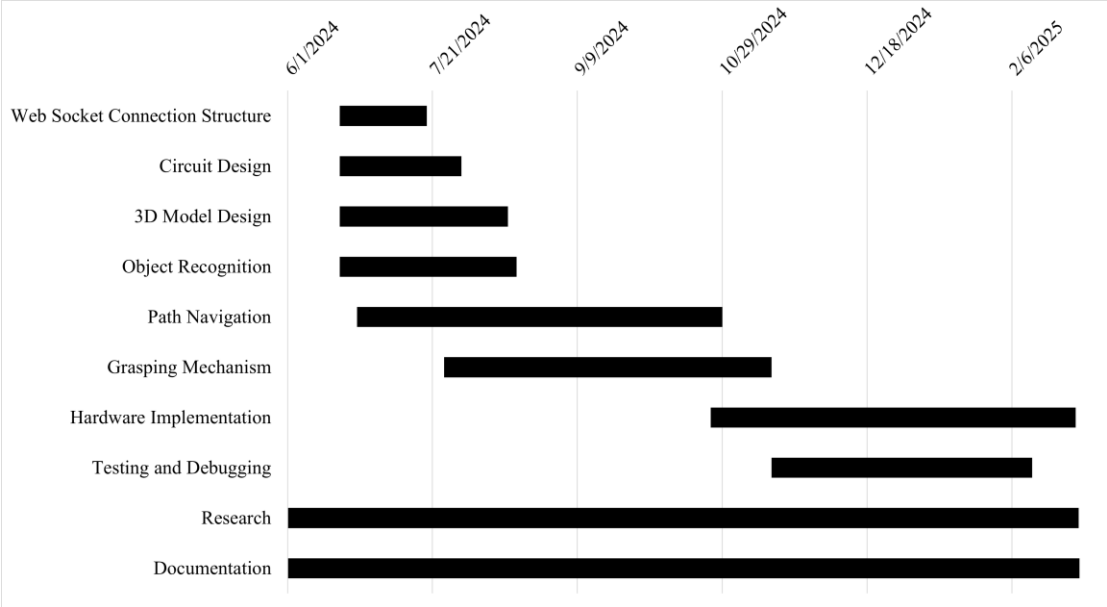
```
        moveStep(leftFootMotor,
static_cast<int>(currentRobotState.leftFoot() + leftFootStep *
j));

        // For Right Leg
        moveStep(rightHipYawMotor,
static_cast<int>(currentRobotState.rightHipYaw() +
rightHipYawStep * j));
        moveStep(rightHipPitchMotor,
static_cast<int>(currentRobotState.rightHipPitch() +
rightHipPitchStep * j));
        moveStep(rightHipRollMotor,
static_cast<int>(currentRobotState.rightHipRoll() +
rightHipRollStep * j));
        moveStep(rightKneeMotor,
static_cast<int>(currentRobotState.rightKnee() + rightKneeStep *
j));
        moveStep(rightAnkleMotor,
static_cast<int>(currentRobotState.rightAnkle() + rightAnkleStep
* j));
        moveStep(rightFootMotor,
static_cast<int>(currentRobotState.rightFoot() + rightFootStep *
j));

        // For Head and Torso
        moveStep(headMotor,
static_cast<int>(currentRobotState.head() + headStep * j));
        moveStep(torsoMotor,
static_cast<int>(currentRobotState.torso() + torsoStep * j));

        delay(stepTime);
    }

    currentRobotState = nextRobotState;
}
```

Appendix L: Motion Planning Interface

Figure 9-18: App Interface for Motion Planning

Appendix M: Project Schedule

Table 9-4: Gantt Chart



Appendix N: Project Budget

Table 9-5: Project Budget

S.N.	Components	Quantity	Price per quantity (Rs.)	Total Price (Rs.)
1	ESP32	1	900	900
2	ESP32-Cam	2	1300	2600
3	16-Channel LU9685 Servo Driver	1	550	550
4	Gyroscope	1	350	350
6	MG996R Servo	20	600	12000
7	LM7805	7	50	350
8	Battery	1	2000	2000
9	SMPS	1	1700	1700
10	9A Buck Converter	1	1100	1100
11	Two Pin Connector	4	10	40
10	3D Printing	-	-	20000
11	Miscellaneous	-	-	5000
	Total			**41590**

References

[1] C. Jewell, "Bringing AI to life," World Intellectual Property Organization, September 2018. [Online]. Available: https://www.wipo.int/wipo_magazine/en/2018/05/article_0003.html. [Accessed June 2024].

[2] Z. Li, B. Xu, D. Wu, K. Zhao, S. Chen, M. Lu and J. Cong, "A YOLO-GGCNN based grasping framework for mobile robots in unknown environments," *Expert Systems with Applications,* vol. 225, 2023.

[3] R. Kumar, S. Lal, S. Kumar and P. Chand, "Object Detection and Recognition for a Pick and," in *Asia-Pacific World Congress on Computer Science and Engineering*, Nadi, Fiji, 2014.

[4] E. DANDIL and K. K. ÇEVİK, "Computer Vision Based Distance Measurement System using Stereo Camera View," in *3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey, 2019.

[5] R. H. Abiyev, M. Arslan, I. Günsel and A. Ça÷man, "Robot Pathfinding Using Vision Based Obstacle," in *3rd IEEE International Conference on Cybernetics (CYBCONF)*, Exeter, UK, 2017.

[6] R. R. Thavai and S. k. N. Kadam, "Inverse Kinematics Solution for Biped Robot," *IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE),* vol. 12, no. 1, pp. 57-62, 2015.

[7] M. A. Ortega-Palacios, A. D. Palomino-Merino and F. Reyes-Cortes, "INVERSE KINEMATICS MODEL FOR A 18 DEGREES OF FREEDOM ROBOT," *Journal of Automation, Mobile Robotics and Intelligent Systems,* vol. 17, no. 1, pp. 22-29, 2023.

[8] D. Huang, W. Fan, Y. Liu and T. Liu, "Design of a Humanoid Bipedal Robot Based on Kinematics and," in *2020 IEEE/ASME International Conference on*, Boston, USA, 2020.

[9] M. Sakhineti and S. Jayabalan, "Design and Fabrication of SHRALA: Social Humanoid Robot," in *Third International Conference on Computing and Network Communications (CoCoNet'19)*, Trivandrum, 2020.

[10] C. S. Pati and R. Kala, "Vision-Based Robot Following Using PID Control," *Technologies,* vol. 5, no. 2, p. 34, 2017.

[11] P. WASILEWSKI and R. GRADZKI, " Standalone brushless motor module optimized," *BULLETIN OF THE POLISH ACADEMY OF TECHNICAL SCIENCES,* vol. 70, no. 2, p. e141008, 2022.

[12] S. Abdolshah, M. Abdolshah and S. Tang, "Trajectory Planning and Walking Pattern Generation of," *International Journal of Robotics and Automation (IJRA),* vol. 4, no. 2, pp. 135-142, 2015.

[13] A. Kumar, F. Zipeng, P. Deepak and J. Malik, "RMA: Rapid Motor Adaptation for Legged Robots," arXiv, 2021.

[14] L. Meng, Y. Dai, S. Wang, K. Zhuang, F. Sun, D. He and A. Wang, "Design and implementation of flyback converter design and," *IET Control Theory & Applications,* vol. 17, pp. 2038-2046, 2023.

[15] J. Terven, D.-M. Córdova-Esparza and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer," *Machine Learning and Knowledge Extraction,* vol. 5, no. 4, pp. 1680-1716, 2023.

[16] E. Hern´andez and R. Vel´azquez, "Design and Development of ZERO," in *IEEE Colombian Conference on Automatic Control*, Bogota, 2011.

[17] Q. Liao, B. Zhang, X. Huang, X. Huang, Z. Li and K. Sreenath, "Berkeley Humanoid: A Research Platform for," arXiv, 2024.