Summer 8-16-2024

# Omobot: a Low-cost Mobile Robot for Autonomous Search and Fall Detection

Shihab Uddin Ahamad
*University of Maine*, shihab.ahamad@maine.edu

# OMOBOT: A LOW-COST MOBILE ROBOT FOR AUTONOMOUS SEARCH AND FALL DETECTION

By

Shihab Uddin Ahamad

Bachelor's of Science in Electrical, Electronic and Communication Engineering,

Military Institute of Science and Technology, Dhaka, Bangladesh,

January 2020

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master's of Science

(in Computer Engineering)

The Graduate School

The University of Maine

August 2024

## UNIVERSITY OF MAINE GRADUATE SCHOOL LAND ACKNOWLEDGMENT

The University of Maine recognizes that it is located on Marsh Island in the homeland of Penobscot people, where issues of water and territorial rights, and encroachment upon sacred sites, are ongoing. Penobscot homeland is connected to the other Wabanaki Tribal Nations—the Passamaquoddy, Maliseet, and Micmac—through kinship, alliances, and diplomacy. The University also recognizes that the Penobscot Nation and the other Wabanaki Tribal Nations are distinct, sovereign, legal and political entities with their own powers of self-governance and self-determination.

# OMOBOT: A LOW-COST MOBILE ROBOT FOR AUTONOMOUS SEARCH AND FALL DETECTION

By Shihab Uddin Ahamad

Thesis Advisor: Dr. Vikas Dhiman

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master's of Science
(in Computer Engineering)
August 2024

Omobot, a cost-effective autonomous mobile robot designed for indoor environments, focuses on fall detection and response for elderly care. The thesis presents the design, development, and evaluation of Omobot's potential for aiding elderly people by combining mechanical and electronics design, software development, and improving fall detection accuracy.

The robot's design includes 3D printed structure and Mecanum wheels that allow it to move easily in indoor environment. It is powered by NVIDIA's Jetson Nano, which helps it to process data from sensors in real-time for navigation and fall detection. The software framework built on Robotic Operating System (ROS) helps in sensor integration, real-time data processing, and provide an open-source platform. System identification modelling is used to simplify control system complexity.

The system includes a fall detection system that uses images captured from the robot's camera perspective and improves detection accuracy through homography transformation that simulates human sight levels. It uses the YOLOv8-Pose, which is a single-stage detector model, to detect the presence of humans, track movements, and identify falls. Once a fall is detected, the system sends emails to designated responders.

Omobot surpasses traditional fall detection systems in accuracy, usability, and reliability, demonstrating significant potential for real-world application. The robot's ability of autonomously navigation contributes to safer living environments for the elderly. The open-source design of Omobot ensures it can be continually updated and improved by the research community for future assistive technologies. The low-cost build can aid in removing the economic barriers that typically limit the deployment of advanced assistive technologies without compromising reliability.

## DEDICATION

To my wonderful parents, thank you for always believing in me. Your support encouraged me to travel far from home to get my degree. I'm grateful for your guidance, which taught me how important it is to try new things and know my own strengths and weaknesses. To my lovely wife, thank you for being my support during the hard times, and teaching me how to be brave, and overcome challenges.

## ACKNOWLEDGEMENTS

This research would not have been possible without the support of many individuals. My sincere gratefulness to all the people who contributed to my journey.

My sincerest gratitude to Dr. Vikas Dhiman for his constant support, patience, and guidance. His mentorship has been crucial in helping me develop essential research skills and improve my abilities. I am very grateful to my co-advisor, Dr. Vijay Devabhaktuni, for his dedication and guidance for the application of my work for helping the society. Dr. Vijay's unwavering dedication has been a great source of inspiration and has significantly contributed to my academic and personal growth. I am thankful to my committee member, Dr. Prbuddha Chakrborty, for guiding me tackling the security issues with my research.

I am grateful to Masoud Ataei for helping me with a significant contribution of my work, Fall Detection. Thank you to all the members of Computer Vision and Autonomous Robotics lab (CVAR), specially Elvis Gyaase, Jeffrey Eiyike, and Karun Varghese for always supporting me with your valuable ideas and helping me in conducting experiments. Thank you Abhilas Durgam and Zafaryab Haider for always be there for me and giving me ideas for improving the research work.

# CONTENTS

UNIVERSITY OF MAINE GRADUATE SCHOOL LAND ACKNOWLEDGEMENT ..... ii

DEDICATION ..... iii

ACKNOWLEDGEMENTS ..... iv

LIST OF TABLES ..... ix

LIST OF FIGURES ..... x

LIST OF ABBREVIATIONS ..... xii

1. INTRODUCTION ..... 1

    1.1 Fall Detection ..... 1

    1.2 Related Work ..... 1

    1.3 Organization ..... 5

2. ROBOT MODELLING AND DESIGN ..... 7

    2.1 Introduction ..... 7

    2.2 Hardware Design ..... 8

        2.2.1 Robot's Base Chassis ..... 9

        2.2.2 Jetson Nano ..... 13

        2.2.3 Arduino Nano ..... 14

        2.2.4 DC-gear Motor with Encoder and Mecanum Wheel ..... 16

        2.2.5 LiDAR ..... 17

        2.2.6 Camera ..... 19

2.2.7 Battery ........ 20
2.2.8 Power and Battery Management System ........ 21
2.2.9 Wireless Charging System ........ 23
2.2.10 Final Assembly ........ 24
2.3 Physical Experiment and Results ........ 25
2.3.1 Payload Capacity ........ 25
2.3.2 Battery Runtime ........ 25
2.3.3 Summary of Findings ........ 26
3. SOFTWARE AND CONTROL SYSTEM DESIGN ........ 27
3.1 Frame Attachment ........ 27
3.2 Inverse Kinematics ........ 28
3.3 Forward Kinematics ........ 29
3.4 Motor Driver Control ........ 30
3.5 Motor Controller Model ........ 31
3.6 Motor System Identification ........ 32
3.7 Effectiveness of motor system identification ........ 33
3.8 Software System ........ 34
3.8.1 Installation and Setup ........ 35
3.8.2 ROS Topic Flow ........ 36
3.8.2.1 Localization: ........ 37
3.8.2.2 Perception: ........ 37
3.8.2.3 Navigation: ........ 37

3.8.2.4 Input/Output, Control, and Fall Detection: ..................... 38

3.9 Operational Experiment and Results ..................... 39

3.9.1 Odometry Accuracy Experiment ..................... 39

3.9.2 Obstacle Avoidance Efficiency ..................... 41

3.9.3 Summary ..................... 43

4. FALL DETECTION SYSTEM ..................... 45

4.1 System Overview ..................... 45

4.2 Pre-processing ..................... 45

4.2.1 Using homography for Converting robot viewpoint images to human viewpoint ..................... 46

4.2.2 Homography from equation of a plane ..................... 46

4.3 Person detection and human pose estimation ..................... 48

4.3.1 YoloV8 ..................... 49

4.3.2 Human Bounding Box Prediction and Keypoints Detection ..................... 51

4.4 Fall detection from human pose ..................... 52

4.4.1 Rules-based Fall Detection ..................... 52

4.4.2 MLP-based Fall Detection ..................... 53

4.5 Experiment and Results ..................... 55

4.5.1 Evaluating the fall detection pipeline ..................... 55

4.5.2 Robustness to homography ..................... 56

4.5.3 Qualitative results ..................... 59

4.5.4 Summary ..................... 60

5. CONCLUSION AND FUTURE WORK ........ 61

5.1 Conclusion ........ 61

5.2 Future Work ........ 61

5.2.1 Enhanced Sensor Integration ........ 62

5.2.2 Adaptive Homography Transformations ........ 62

5.2.3 Fall Detection Model Enhancements ........ 62

5.2.4 Multi-Sensor Data Fusion ........ 63

5.2.5 User Interaction and Interface Enhancements ........ 63

5.2.6 Real-World Testing and Customization ........ 63

BIBLIOGRAPHY ........ 64

BIOGRAPHY OF THE AUTHOR ........ 70

# LIST OF TABLES

2.1 Cost Breakdown . . . . . . 8

2.2 Specifications of the Frame Base . . . . . . 12

2.3 Printing parameters for base frame . . . . . . 12

2.4 Size, frame, and coordinate of the Jetson Nano . . . . . . 14

2.5 Jetson nano's connection with other devices and sensors . . . . . . 14

2.6 Frame, speed, and coordinate of the wheels assembly . . . . . . 18

2.7 Size, frame, technical specification, and coordinate of the LiDAR LD19 . . . . . . 19

2.8 Size, frame, technical specification, and coordinate of the IMX219 stereo camera configuration. . . . . . . 20

2.9 Size, frame, and coordinate of the Battery . . . . . . 20

3.1 Motor Driver Connection . . . . . . 30

3.2 Hyper Parameters for the Motor Model . . . . . . 33

4.1 Fall detection module performance on the created dataset . . . . . . 56

4.2 YOLOv8n accuracy reduces when images are transformed to robot POV . . . . . . 57

# LIST OF FIGURES

2.1 System Design ... 7

2.2 Trimetric View of the Robot ... 9

2.3 Schematic of the printed circuit board (PCB) ... 10

2.4 CAD model of robot's base frame ... 11

2.5 CAD model of Jetson Nano ... 13

2.6 Schematic of Arduino Nano Connection ... 15

2.7 Mecanum Wheel ... 17

2.8 CAD model of the motor and wheel assembly ... 17

2.9 CAD model of the LiDAR and LiDAR mount ... 18

2.10 CAD model of the camera and camera mounts ... 19

2.11 CAD model of the battery and battery mounts ... 21

2.12 CAD model of the power supply, charging, and battery management system (BMS) ... 22

2.13 Printed Circuit Board (PCB) ... 22

2.14 CAD model of the wireless charger receiver and mounts ... 23

2.15 Final-assembly of the robot ... 24

3.1 Cordinate frame modelling ... 27

3.2 Robot's trajectory and path deviation with and without motor system identification. ... 34

3.3 Communication among sensors and devices with ROS nodes running on Jetson Nano. 35

3.4 ROS Topics Graph 36

3.5 Mapping using LiDAR, Wheel Odometry, and `ros-slam_toolbox`. 40

3.6 Robot's workflow 42

3.7 Occupancy grid map Visualization. 42

4.1 YoloV8 Architecture. 50

4.2 MLP Architecture 54

4.3 MLP accuracy on Fall Detection Dataset 55

4.4 Applying Homography on a sample image in the data set. **Left**: image taken from the robot's POV. **Right**: transformed image to a human POV. 57

4.5 F1 Score of Pose Estimation and Comparison of fall detection accuracy. 58

4.6 **Top**: Fall event detection in the presence of multiple people. **Bottom**: Fall detected by the robot, and sample email generated by the system. 59

## LIST OF ABBREVIATIONS

| | |
|---|---|
| **2D** | Two Dimension |
| **3D** | Three Dimension |
| **COM** | Center of Mass |
| **UART** | Universal Asynchronous Receiver Transmitter |
| **IMU** | Inertial Measurement Units |
| **CNN** | Convolutional Neural Network |
| **DC** | Direct Current |
| **PETG** | Polyethylene Terephthalate Glycol |
| **AMCL** | Adaptive Monte Carlo Localization |
| **LiDAR** | Light Detection and Ranging |
| **PLA** | Poly Lactic Acid |
| **CAD** | Computer Aided Design |
| **ROS** | Robot Operating System |
| **YOLOv8** | You Only Live Once version 8 |
| **SDK** | Software Development Kit |
| **IP** | Internet Protocol |
| **SSH** | Secure Shell Protocol |
| **SPI** | Serial Peripheral Interface |

| | |
|---|---|
| **CSI** | Camera Serial Interface |
| **PWM** | Pulse Width Modulation |
| **SLAM** | Simultaneous Localization and Mapping |
| **LiPo** | Lithium Polymer |
| **MCU** | Micro Controller Unit |
| **ML** | Machine Learning |
| **RMS** | Root Mean Squared |
| **POV** | Point Of View |
| **CSP** | Cross-Stage Partial Bottleneck |
| **SPPF** | Spatial Pyramid Pooling Fast |
| **SiLU** | Sigmoid Linear Unit |
| **ReLU** | Rectified Linear Unit |
| **FPN** | Feature Pyramid Networks |
| **PAN** | Path Aggregation Network |
| **MSE** | Mean Squared Error |
| **IoU** | Intersection over Union |
| **MLP** | Multi-Layer Perceptron |

# CHAPTER 1
# INTRODUCTION

## 1.1 Fall Detection

The percentage of people aged over 65 is increasing in the OECD countries, with growth from 11.36% in 1990 to 17.96% in 2022(OECD, 2024) [1,2]. This demographic shift highlights the growing importance of fall prevention and management in addressing the challenges associated with an aging population, especially in healthcare and social support systems. Since the 1990s, there has been increasing recognition of the challenges posed by fall prevention and management among the elderly. Research by Tinetti [3] and Doughty [4] has analyze the risks and management of falls in older adults.

Fall detection systems are essential for older people. These systems help to quickly notice if someone falls and help them promptly. These systems keep an eye on older people all the time and give them help right when they need it. This makes older people safer and more independent, and it means they don't need as much help from caregivers or healthcare.

## 1.2 Related Work

K. Doughty et. al [4] discusses the evolution of telecare systems from manual emergency response to automatic sensing and health monitoring, predicting future advancements with broadband communication for enhanced elderly care. User-activated alarm systems by S Chaudhuri et al. [5] require the person to press a button to request help from community responders. One limitation of this approach is that it requires the fallen person to be conscious and decide to call for help. The hesitancy of the elderly to ask for help under such circumstances has pushed researchers to develop passive fall detection systems that do not require any action from the patients themselves.

Literature on passive fall detection systems has been reviewed multiple times in the last two decades While preventing falls is the first line of defense, the second line of defense managing falls by reducing the response and rescue time [6]. P Rajendran et al. [6] emphasizes fall prevention and detection technologies, such as automated wearable detectors and environmental modifications, that use advanced sensing and data analysis to reduce fall risks for the elderly. The survey by X Wang et al. [7] looks into various approaches and challenges in developing effective fall detection systems, emphasizing the use of multiple sensors to enhance detection accuracy and reduce false alarms by integrating data from different sensors potentially that leads to higher system robustness and reliability. Z Zhang et al. [8] categorizes existing fall detection approaches into sensor-based and vision-based, with an emphasis on non-intrusive, camera-based systems that enhance elderly care by a real-time monitoring without physical contact. In summary, these passive fall detectors can be classified based on the location of sensors as *wearable* or *ambient*. A *wearable* fall detector is worn by the person to be tracked. It typically uses IMU (Inertial Measurement Units) and health sensors to detect falls. An *ambient* fall detection technology is usually installed in the person's home. These systems typically use pressure sensors, vibration sensors, or cameras to detect a fall and alert the caregivers.

While wearable sensors have become less conspicuous, they can easily lead to false alarms. On the other hand, ambient sensors, while more accurate, are more costly. Admittedly, these sensors can be combined to complement each other. One limitation of all camera-based systems is that some consider them to invade the privacy of the individual being monitored. These technologies have complementary strengths and weaknesses, and a hybrid system can be customized based on the user's needs and preferences. These limitations, combined with advances in deep learning and autonomous robots, have led to a new type of ambient fall detectors: mobile robots as fall detectors. L Ciabattoni et al. [9] explored fall detection systems in smart homes, integrating smartphone-based acceleration data and Kinect sensors for precise location tracking to monitor elderly movements. Z A Mundher et al. [10] employs a Kinect sensor on a mobile robot that detects falls and initiates emergency communication through SMS and calls. These are ambient-mobile sensors, contrasting with the previous generation of ambient-static sensors.

Many mobile devices have been proposed for fall detection as a classification problem. This requires classifying images as either "fall" or "no fall". W H Chin et al. [11] integrated approach combining a lightweight deep learning model with an assistive mobile robot to enhance fall detection and support for the elderly focusing on a vision-based fall detection using a low-cost 2D camera and a mobile robot equipped to perform autonomous navigation and assistance tasks. C Menacho et al. [12] focuses on developing an efficient Convolutional Neural Network (CNN) model for fall detection, optimized for performance on a mobile robot equipped with limited computational resources, like the Nvidia Jetson TX2. However, the reliance on Optical Flow in the system's performance could degrade in complex dynamic backgrounds or under variable lighting conditions, which are common in real-world environments.

Extensive research has been carried out in the field of intelligent autonomous mobile robots to develop advanced features such as obstacle avoidance, object detection, path planning, and map creation. In one such effort, Andruino-R2, a mobile robot, was developed and implemented by F M Lopez-Rodriguez et al. [13] with line-following navigation combining Arduino and Java-based ROS. A A S Gunawan et al. [14] developed an integrated system named BeeButler that combines a flutter-based multi-platform mobile app and a robot to aid hotel guests in ordering amenities. It solves the previous problem of external devices by using onboard Jetson Nano and Arduino Mega 2560 to control the robot. Still, it uses a line follower and RFID tag for localization, which is unsuitable for navigating an unknown environment without setting additional lines and RFID tags. A two-wheel home-assistive robot featuring a combination of omnidirectional wheels with differential driving was developed by P. He et al. [15]. The author used an STM32 microcontroller and NVIDIA Jetson Nano to control the robot and demonstrated successful 2D and 3D SLAM experiments, autonomous navigation, and obstacle avoidance. X. Tan et al. [16] developed a ROS-based omnidirectional robot with mapping, localization, and navigation capabilities using a multisensor fusion on Raspberry Pi 4B, emphasizing accuracy and efficiency.

G. Chen et al. [17] utilized a vision-based detection method that utilizes the NanoDet-Lite, a derivative of the NanoDet model optimized for small devices like the Raspberry Pi 4 Model B. This model achieves high accuracy on low-cost hardware without a dedicated hardware accelerator. There have been significant improvements in the accuracy of object detection algorithms such as YOLO by A. Wong et al. [18] by formulating fall detection as an object detection problem instead of an image classification problem, multiple persons can be identified in the same image and classify each as fallen or not. This is especially useful when a dummy or statue is present in the same image as the elderly person that the robot is supposed to watch. However, in terms of cost, these robots are high, and a low-cost implementation was not prioritized which can be a barrier for potential scalability in the future. To address the issue, a custom low-cost mobile robot, Omobot has been designed and prototyped that is intended expressly for the purpose of periodically surveying an area and checking for persons who have fallen.

While most mobile robots depend on camera-based fall detection, some users prefer them over ambient camera-based systems. Because mobile robots perform only periodic monitoring as opposed to ambient systems' continuous monitoring. A mobile robot system can be configured to check on a person at regular intervals, such as every 30 minutes, and be triggered by wearable sensors or loud noises. The mobile robot can then search for the person in the house. If the person is found and identified as being in a fallen position, then an alert is generated. The advantage of this approach over ambient cameras is that a mobile robot checking on an elderly person is periodic, and the person being checked on is reminded of the robot's presence, providing a sense of physical interaction. As a supplementary advantage, the requirement to keep the floor clear for the robot's passage can also help prevent falls.

## 1.3 Organization

The thesis paper is organized into four main chapters, each dedicated to a different aspect of the research and development of an efficient robotic system designed to assist in elderly care by detecting falls autonomously.

**Chapter 1** starts introducing the challenges of fall prevention and management among older people, a critical issue given the increasing proportion of aged populations worldwide. It reviews existing technologies, such as user-activated alarms and passive detection systems, and positions mobile robots as an innovative solution. The chapter outlines the structure of the entire thesis and establishes the context for Omobot's development.

The hardware design, software, and assembly of Omobot are provided in **Chapter 2**. This includes comprehensive coverage of the hardware setup and software design. Experimental results validating design choices through tests on load capacity, battery life, and navigation accuracy are also presented at the end of the chapter. In **Chapter 3**, control system designs are elaborated in detail with results that show the effectiveness of the control system.

**Chapter 4** dives into the fall detection system implemented within Omobot. It describes the smart pre-processing techniques for enhancing the accuracy of the Fall detection model. Detailed experiments and results assess the effectiveness of the fall detection mechanisms under varied conditions. Also, a demonstration of Omobot in a real-world scenario is shown at the end of the chapter.

The concluding chapter, **Chapter 5**, summarizes the key findings and suggests directions for future research to further enhance the capabilities and applications of mobile robots in assistive care technologies.

This thesis work has been published in a peer-reviewed conference, 2024 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM) in July 2024, and much of the writing mirrors the content of the published paper. A significant contribution to the fall detection section 4 was made by my co-author, Masoud Ataei who played a crucial role in both the conceptualization and implementation of this part of the study. Additionally, the research was guided and mentored by Dr. Vikas Dhiman and Dr. Vijay Devabhaktuni. Their support was invaluable throughout the entire research process.

# CHAPTER 2

# ROBOT MODELLING AND DESIGN

## 2.1 Introduction

The comprehensive design of the robot, incorporating its mechanical, electronic, and software components, has been thoroughly discussed in this chapter. For a demonstration video, fall detection dataset, and instructions for using the robot, please consult the GitHub repository.[1] . The overall hierarchy of system design is presented in Figure 2.1. One of the

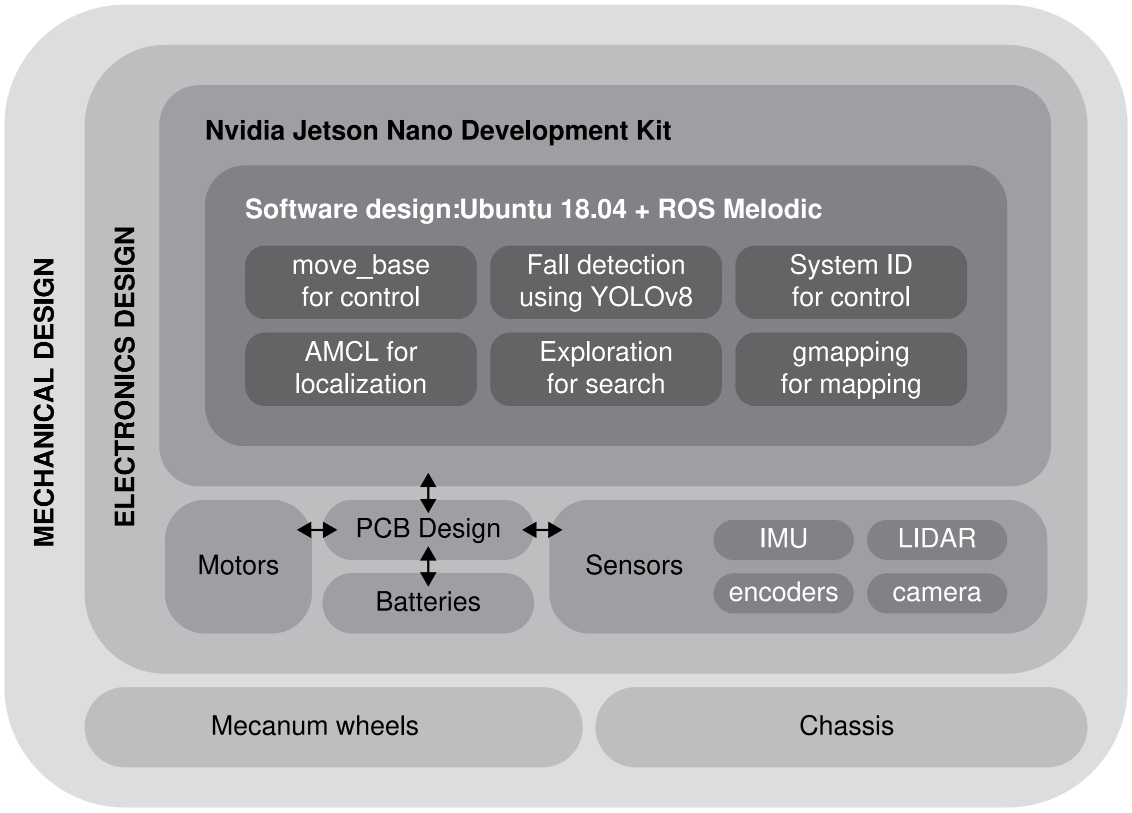


Figure 2.1: System Design

objectives is to develop a resilient, dependable, and economical robot with components priced at less than $1000. The final build-up cost of the robot is $693, and the cost breakdown of

[1]https://github.com/shihab28/omobot_js

Table 2.1: Cost Breakdown

| Components | Cost (USD) |
|---|---|
| Jetson Nano (Dev-4GB) with accessories, and Camera | 300 |
| DC gear motor (12V) with encoder and mecanum wheels | 90 |
| Arduino (Nano RP2040) | 42 |
| LiDAR (LD19) | 99 |
| IMU (MPU-6050) | 3 |
| LiPo Battery (5200mAh) | 29 |
| Power supply and charging module (12V) | 39 |
| PCB Board (Dual Layer-Through Hole) | 15 |
| Wireless charging system (12V-2A DC) | 36 |
| 3D printed parts | 30 |
| Misc (wire, bolts, nuts, connectors, etc.) | 10 |
| Total | 693 |

all the parts is displayed in Table I, which illustrates how the complete robot is built for a budget of under $700, ensuring affordability without compromising quality. The items were purchased and priced based on market price between November 2022 and December 2023. The cost breakdown is shown in the Table 2.1.

## 2.2 Hardware Design

The hardware design is composed of the detailed mechanical and electronic components design and assembly. Initially, the skeleton of the robot was designed using the Solidworks model. The device's design, as shown in Figure 2.2, shows the trimetric view of the robot's design. Afterward, the four DC gear motors with encoders and the four Mecanum wheels [19] are mounted to the bottom aluminum chassis following the correct orientation and parameter defined in the design (Distance from the origin of the robot's base-footprint to center of the wheel's origin along X-axis; l=118.5mm, distance from the origin of the robot's base-footprint to center of the wheel's origin along Y-axis; w=82.5mm). Aside from the motor mounts and bottom chassis, all other bases, mounts, and joints are individually designed and 3D printed using Polyethylene Terephthalate Glycol (PETG) and Polylactic Acid (PLA) filament. A

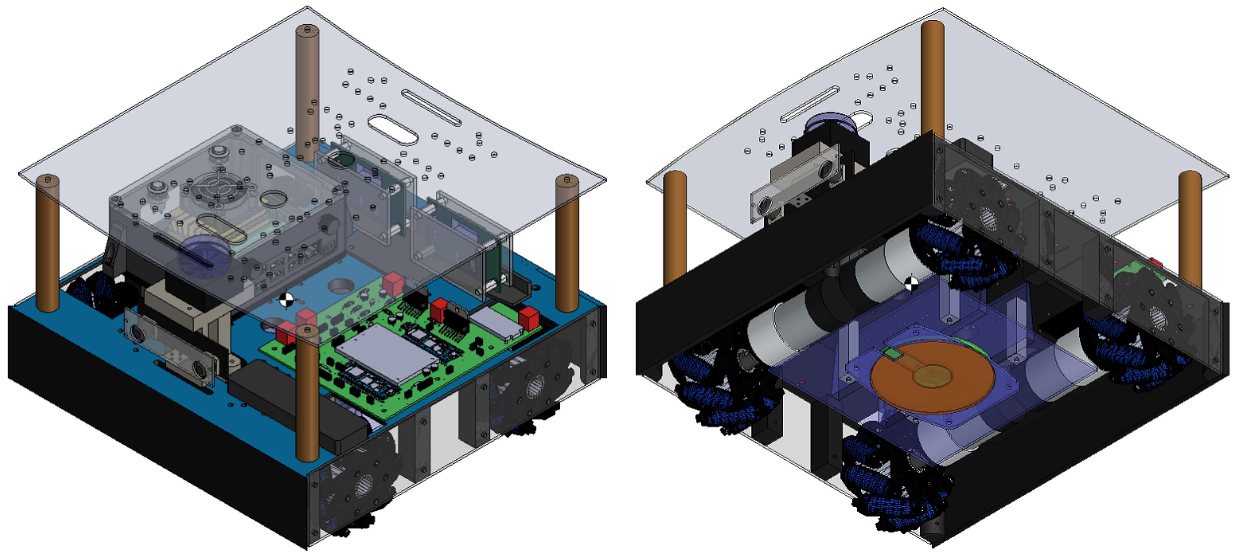


Figure 2.2: Trimetric View of the Robot

12-volt Li-Po battery is mounted below the chassis to lower the center of mass. The wireless charging coil is covered in a 3D printed cover and mounted at the center below the battery mount. Jetson Nano is located on the top of the base. The Jetson Nano is the robot's brain, processing data and executing tasks. Beside the Jetson Nano, the printed circuit board (PCB) is located. Figure 2.3 shows the schematic diagram used for the PCB. The two Arduino Nano RP2040 microcontroller is placed on the PCB along with the two H-bridge motor controllers LM298N; the DC-DC switching 12V-3A buck-boost converter to supply constant 12V supply to the motor; the DC-DC 5V-4A buck power converter to supply power to the Jetson Nano and Arduino; the charging module and battery management system (BMS).

### 2.2.1 Robot's Base Chassis

The base chassis or frame provides structural support and holds together all essential components of the robot. The base is designed using Solidworks as shown in Figure 2.4a. While designing the base, the following design requirements are followed,

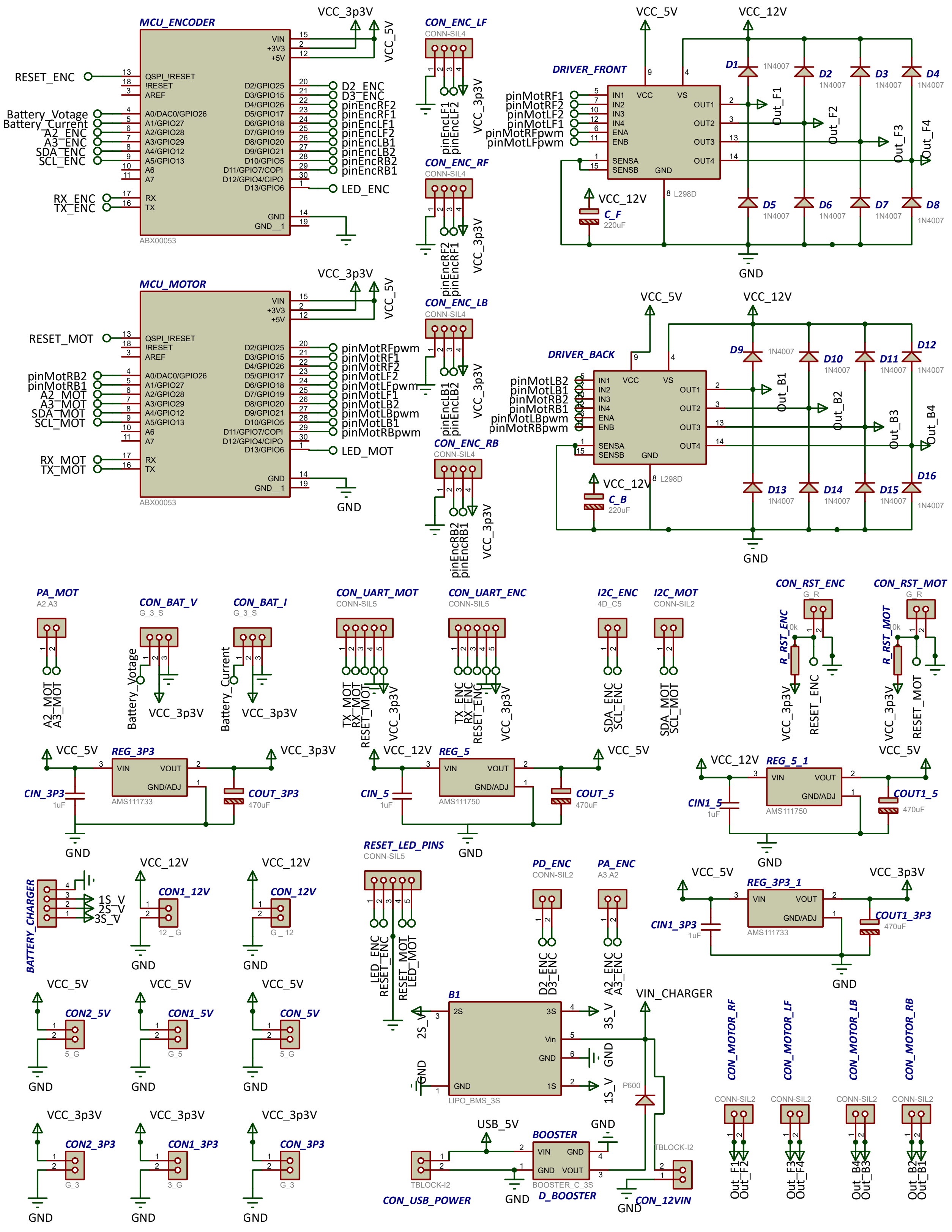
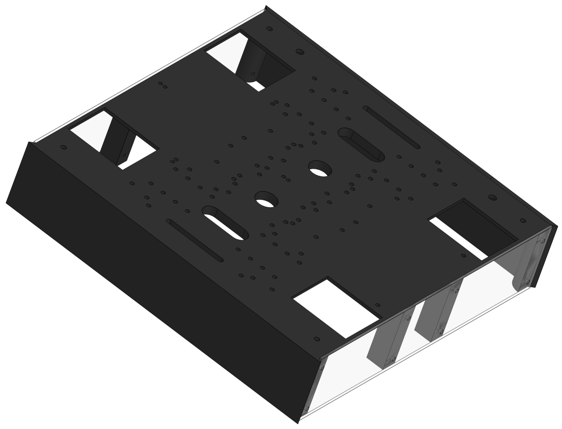


Figure 2.3: Schematic of the printed circuit board (PCB)

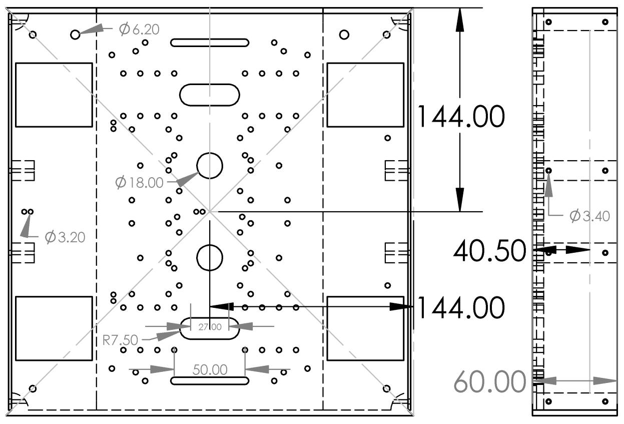
- The base should withstand physical stress and external forces during movement, collisions, or heavy payload carrying. PETG filament ensures the durability and robustness which is required from the robot [20]. A symmetrical design is adopted to keep the center of mass relatively lower and at the center of the robot frame to avoid rollover during motion.

- The base should provide a secure integration platform for the components such as motors, wheels, jetson nano, Light Detection and Ranging (LiDAR), PCB board, wireless charger, and power systems. According to research necessity, there should be sufficient options to integrate extra components securely in a later stage. As shown in the F 2.4a, multiple anchor points are kept in the design for extending attachments in the future.

- The base should have easy access to internal components, simplifying maintenance, repair, and replacement of parts if necessary. Designing a flat robot rather than adopting a vertical design ensured easier access to the components.



(a) 3D model



(b) 2D drawing from top and side view.

Figure 2.4: CAD model of robot's base frame

After many design modifications, the final design is selected as Figure 2.4b. The base's size, frame, and coordinate information are summarized in Table 2.2.

Table 2.2: Specifications of the Frame Base

| **Property** | **Value** |
|---|---|
| Frame | base |
| Length (mm) | 288 |
| Width (mm) | 288 |
| Height (mm) | 60 |
| Weight (gram) | 546 |
| Coordinates [x, y, z] (mm) | [0.0, 0.0, 40.50] |

The square-shaped base base-chassis has a length and width of 288 mm. For integrating components, sufficient symmetrical holes with multiple sizes are kept. The symmetry in the designs ensures that the weight distribution is even and that the center of mass (COM) lies in the geometric center of the base. To access the components mounted below the base, the detachable side cover is attached using eight M3 bolts instead of making the sides a single body merged with the main frame. It ensures that the Battery and wireless can be maintained without flipping the robot and avoids small objects going inside, hampering the wheel movement.

The chassis is printed with PETG filament, and the primary printing parameter is given in Table 2.3. Printing the base with PETG filament ensures robustness and provides enhanced durability [20] and impact resistance compared to other options like PLA. Also, it exhibits

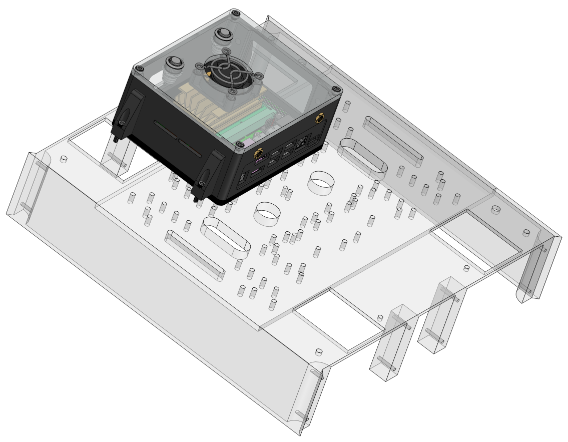
Table 2.3: Printing parameters for base frame

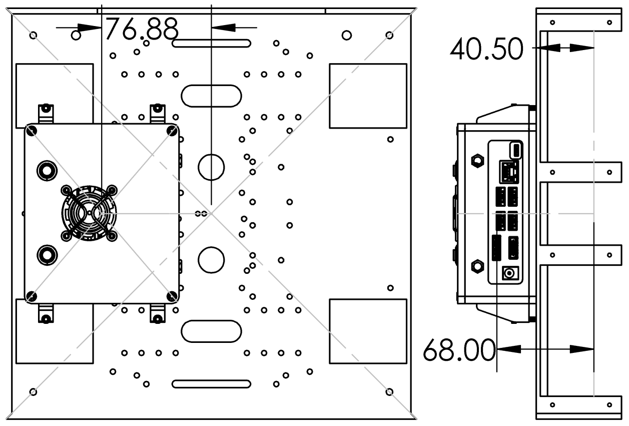
| **Property** | **Value** |
|---|---|
| Printer | Creality CR-10S5 |
| Filament Size (mm) | 1.75 |
| Nozzle Diameter (mm) | 0.4 |
| Bed Temperature (°C) | 75 |
| Extruder Temperature (°C) | 245 |
| Infill Density (%) | 35 |
| Layer Height (mm) | 0.16 |

higher temperature resistance, ensuring more excellent stability for specific robot applications under elevated temperatures.

### 2.2.2 Jetson Nano

NVIDIA's Jetson Nano 4GB Developer Edition A1 Kit [21] has been used as the robot's central brain. It is a compact yet powerful AI computer tailored for robotics and AI development. Featuring a quad-core ARM Cortex-A57 CPU and 128-core NVIDIA Maxwell GPU, it offers real-time AI processing and supports popular AI frameworks like TensorFlow and PyTorch. With Robot Operating System (ROS) compatibility and GPIO pins for sensor integration, it's an accessible platform for robotics experimentation and development. These features make it ideal for autonomous navigation, obstacle avoidance, and object recognition in an omnidirectional mobile robot [22]. With ROS Melodic support, the LiDAR LD19, IMU MPU6050, and Sony IMX219 Stereo cameras have been seamlessly integrated into the system. The Figure 2.5a shows the 3D modeling and assembly. A Samsung FIT


(a) 3D trimetric view of the model



(b) 2D drawing from top and side view

Figure 2.5: CAD model of Jetson Nano

Plus USB 3.1 64GB Flash Drive is used for storage, and the ARM64-based Ubuntu-1804 is installed using JetPack-4.5.1 software development kit (SDK). JetPack SDK 4.5.1 is an NVIDIA development kit offering enhanced features, optimizations, and AI development and deployment tools on Jetson platforms. The size, static frame, and coordinate information are summarized in Table 2.4. For wireless communication, the Waveshare AC8265 Wireless NIC Module has been installed, which supports 2.4GHz / 5GHz dual band WiFi and

Table 2.4: Size, frame, and coordinate of the Jetson Nano

| **Property** | **Value** |
|---|---|
| Frame | Jetson |
| Length (mm) | 112 |
| Width (mm) | 112 |
| Height (mm) | 60 |
| Weight (gram) | 340 |
| Coordinates [x, y, z] (mm) | [0.00, -76.88, 68.00] |

Bluetooth-4.2. Jetson supports several communication protocol methods to communicate between devices and sensors, including universal asynchronous receiver transmitter (UART), universal serial bus (USB), I2C, Internet Protocol (IP), secure shell protocol (SSH), serial peripheral interface (SPI), Camera Serial Interface (CSI), etc. Table 2.5 shows the device connection and protocol used for communication. A Samsung FIT Plus USB 3.1 64GB Flash Drive is used for storage, and Ubuntu-1804 based on ARM64 is installed using the JetPack-4.5.1 software development kit (SDK). JetPack SDK 4.6.1 [23] is an NVIDIA development kit that offers enhanced features, optimizations, and AI development and deployment tools on Jetson platforms.

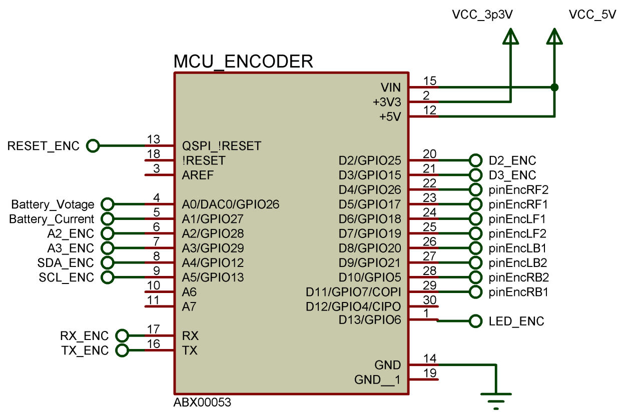
Table 2.5: Jetson nano's connection with other devices and sensors

| **Device and Sensor** | **Connection** |
|---|---|
| Arduino Nano (Encoder) | UART |
| Arduino Nano (Motor) | USB |
| LiDAR-LD19 | USB |
| Sony IMX219 Cameras | CSI |
| IMU MPU-6050 | I2C |
| Oled-Display | I2C |
| Wireless Controller | Wifi, IP |

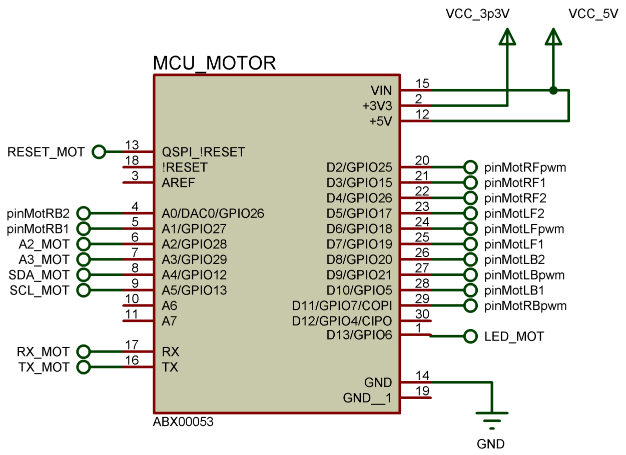
### 2.2.3 Arduino Nano

Arduino Nano RP2040 is a member of the Arduino Nano series. It features the RP2040 microcontroller, offering a clock speed of 133 MHz, 264KB of SRAM, 16 Pulse Width

Modulation (PWM) pins, 20 external hardware interrupt pins, and a USB interface. It provides a versatile platform for embedded projects requiring robust processing power and extensive I/O capabilities. Its compact form factor, measuring only 45 x 18mm, makes it suitable for space-constrained applications like robotics. Two Arduino Nano RP2040 are used for secondary microcontrollers. One of the dedicated microcontrollers is used to control motors, and the other is used to get feedback from the encoders. Eight encoder feedback is coming from the four motors to one of the Arduinos, and the 20 Interrupt pins help to read and process the feedback pulse without delay. The second Arduino is dedicated to controlling the robot by receiving motor speed from Jetson Nano to drive the motors using control signals. Figure 2.6 shows the schematic diagram with the encoder and motor driver connections. Figure 2.6a shows the encoder connections. the encoder pins are connected



(a) Encoder Controller



(b) Motor Controller

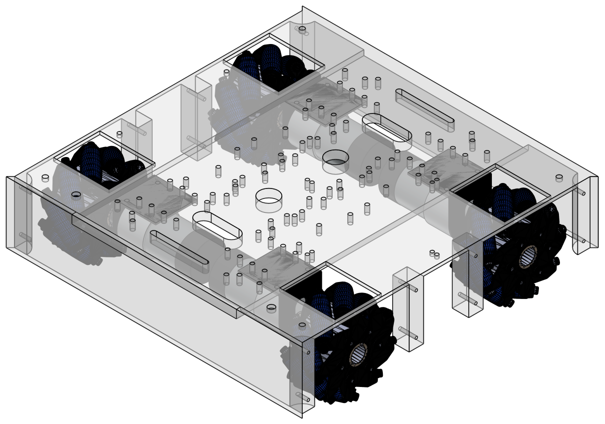
Figure 2.6: Schematic of Arduino Nano Connection

from the microcontroller's digital pin4 (D4) to digital pin11 (D11). The UART pins (RX, TX) are connected to the UART pins of the Jetson nano, respectively, to establish serial communication. The battery voltage and current drawn by the robot are monitored using the analog pin0 (A0) and analog pin1 (A1), respectively. The measured speed of the four-wheel (pulse/sec), battery voltage level (V), and current (mA) are sent to Jetson Nano as an array. Analog pin2 (A2) and pin3 (A3) are kept for future use. Similarly, Figure 2.6b shows the connection for the motor. Twelve connections are required to control the motor: three connections (two for assigning the rotational direction and one for controlling the speed using 8-bit PWM) for each motor. The ten connections go from digital pin2 (D2) to digital pin11 (D11), and the rest of the two are connected to analog pin0 (A0) and analog pin1 (A1). The Arduino receives the desired motor speed from Jetson Nano via USB communication. Analog pin2 (A2) and pin3 (A3) are kept for future use.

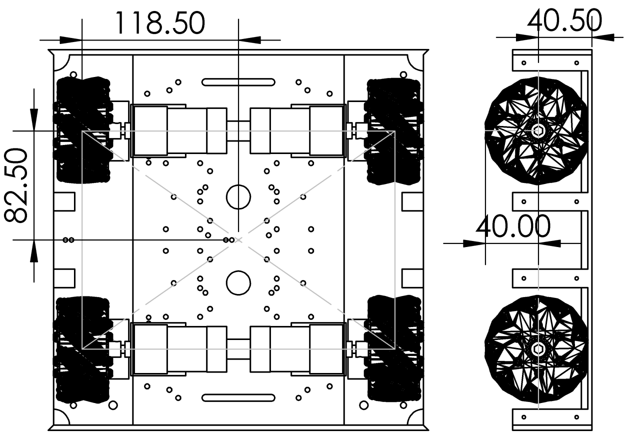
### 2.2.4 DC-gear Motor with Encoder and Mecanum Wheel

Four DC-gear motors equipped with an encoder are used to move the robot. DC-gear motors are perfect for mobile robots due to their high torque output, providing sufficient power to move them swiftly across various paths. The encoders provide real-time feedback about the motor's position and speed [24], enabling precise control and navigation. This feedback loop allows for accurate distance and direction calculations, providing better path planning and obstacle avoidance, which is crucial as the robot moves indoors. Mecanum wheels offer omnidirectional movement, enabling a robot to move in any direction without changing its orientation. The mecanum wheel, as shown in Figure 2.7, has angled rubber rollers around its edge that rotate at 45 degrees to the wheel plane and axle line. These rollers let the wheel move in any direction by creating force perpendicular to their axis [19]. The wheel design allows for omnidirectional movement on a 2D surface. As shown in Figure 2.8, the four motors and wheel are mounted in the vertex of the rectangle, keeping the origin of the base frame as its circumcentre, meaning that the distance of the four wheels' origin is constant. Then, coordinate frames are attached to each of the wheels, which helps to define the position and orientation of the wheel, aiding in accurate odometry calculations for determining the robot's movement and position. Additionally, it assists in the motion planning algorithms by providing the relationship between its wheels' movements and control

Figure 2.7: Mecanum Wheel



(a) 3D trimetric view of the model



(b) 2D drawing from bottom and side view.

Figure 2.8: CAD model of the motor and wheel assembly

signal. Table 2.6 shows the frame, speed range, encoder pulse per rotation, and coordinates of individual wheels. The diameter (r) of each wheel is 80 mm, the thickness ($t_W$) of the wheel is 32.5 mm, and the mass ($m_W$) is 68 grams. All the rotational axis of the wheels are Y-axis aligned with the Y-axis of the base frame.

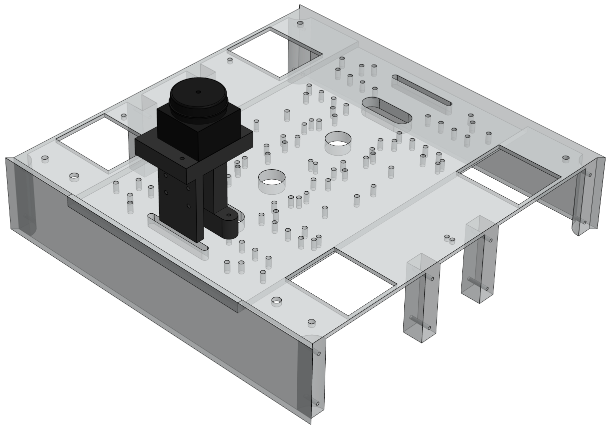
### 2.2.5 LiDAR

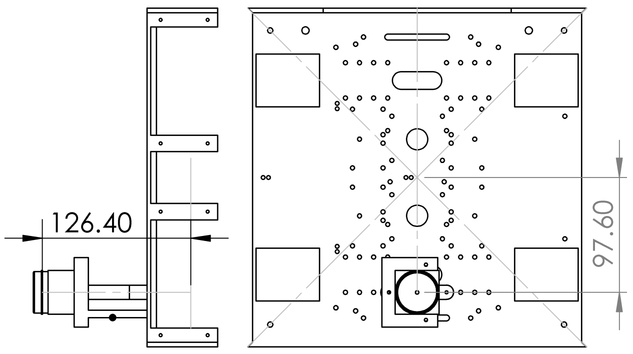
To perceive the environment, the robot uses LiDAR scans to measure distances and create a 2D point cloud, which can be converted into a grid map. LiDAR is crucial in robotics for providing accurate environmental mapping for navigation, obstacle detection, and localization, perceiving surroundings effectively [25]. This helps robots move autonomously

Table 2.6: Frame, speed, and coordinate of the wheels assembly

| **Frame** | **Maximum Speed (rad/sec)** | **Minimum Speed (rad/sec)** | **Pulse Per Rotation (pulse/rot)** | **Coordinates [x, y, z] (mm)** |
|---|---|---|---|---|
| wheel1 | 2.38 | 1.20 | 3264 | [ 82.5, 118.5, 0.0] |
| wheel2 | 2.38 | 1.20 | 3264 | [ 82.5, -118.5, 0.0] |
| wheel3 | 2.38 | 1.20 | 3264 | [-82.5, 118.5, 0.0] |
| wheel4 | 2.38 | 1.20 | 3264 | [-82.5, -118.5, 0.0] |

while navigating or exploring new places. For this robot, LiDAR LD19 has been used and mounted on the front section of the base frame, as shown in Figure 2.9a. The LiDAR LD19


(a) 3D trimetric view of the model



(b) 2D drawing from top and side view.

Figure 2.9: CAD model of the LiDAR and LiDAR mount

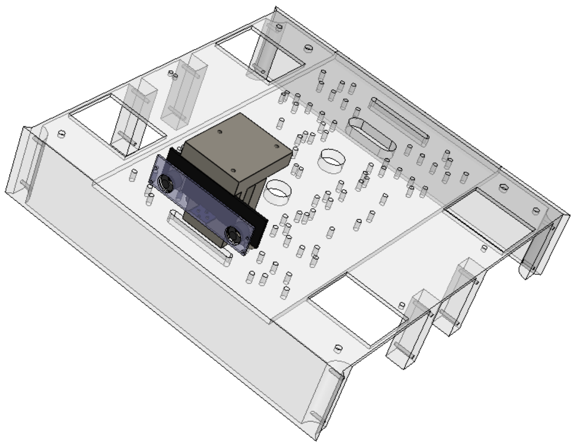
sends out an infrared laser and records the time of travel for reflection from objects, using this time to calculate distances. It combines these distances with angle measurements to create a 2D map of points representing the surroundings. This map is sent to the Jetson Nano via USB communication while the LiDAR adjusts its movements through a control system to work steadily. The ROS-Melodic-SLAM package then uses this map from the LiDAR to localize the robot and any objects it detects in the map. The frame, technical specifications, and coordinate information are summarized in Table 2.7.

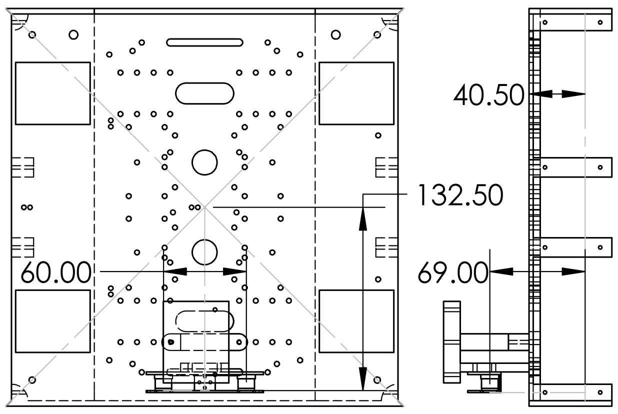
Table 2.7: Size, frame, technical specification, and coordinate of the LiDAR LD19

| **Property** | **Value** |
|---|---:|
| Frame | LiDAR |
| Dimensions (mm) | 38.6, 38.6, 33.5 |
| Measuring Range (m) | 0.02-12 |
| Sweep Frequency (Hz) | 10-13 |
| Scanning Range (°) | 360 |
| Coordinates [x, y, z] (mm) | [97.60, 0.00, 126.40] |

### 2.2.6 Camera

For visual perception, two Sony IMX219 cameras are configured to be used as a stereo camera. At first, the cameras are calibrated, which includes correcting lens distortions, adjusting camera intrinsics and extrinsic, and determining the relative position and orientation between the two cameras. For the stereo calibration, OpenCV-Python Stereo-Calibration [26] is used to find the intrinsic and extrinsic parameters of the cameras. A stereo camera can be used for depth perception by capturing a scene simultaneously from two distinct viewpoints, mimicking the human eyes' binocular vision. The disparities between the viewpoints allow the system to compare points in the images to calculate depth and make a 3D visualization of the scene using triangulation method [27]. The Figure 2.10 shows the stereo camera assembly.


(a) 3D trimetric view of the model



(b) 2D drawing from top and side view.

Figure 2.10: CAD model of the camera and camera mounts

The Sony IMX219 camera is cheaper than other stereo cameras in the market but offers good imaging capabilities. The stereo camera has two mono cameras kept 60mm apart, and one of the captured images from the cameras is used for fall detection as, currently, stereo vision is not required for the fall detection problem. The cameras are connected to the Jetson Nano using the two CSI interfaces. The frame, technical specifications, and coordinate information are summarized in Table 2.8. The next is to celebrate the cameras.

Table 2.8: Size, frame, technical specification, and coordinate of the IMX219 stereo camera configuration.

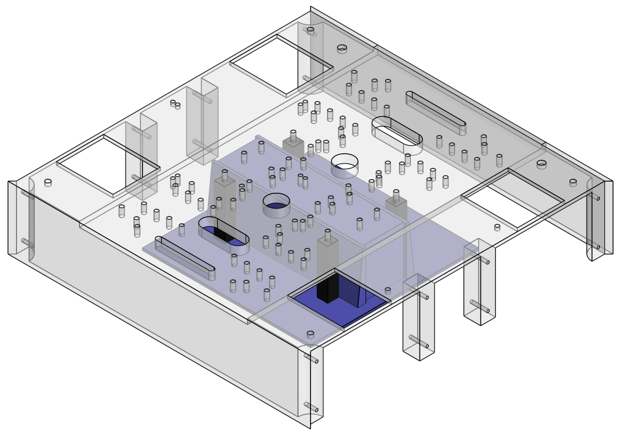
| **Property** | **Value** |
|---|---:|
| Frame | Camera |
| Baseline Length (m) | 80 |
| Focal Length (m) | 2.6 |
| Resolution (Megapixels) | 8 |
| Angle of View d/h/v (°) | 83/73/50 |
| Coordinates [x, y, z] (mm) | [132.50, 0.00, 69.00] |

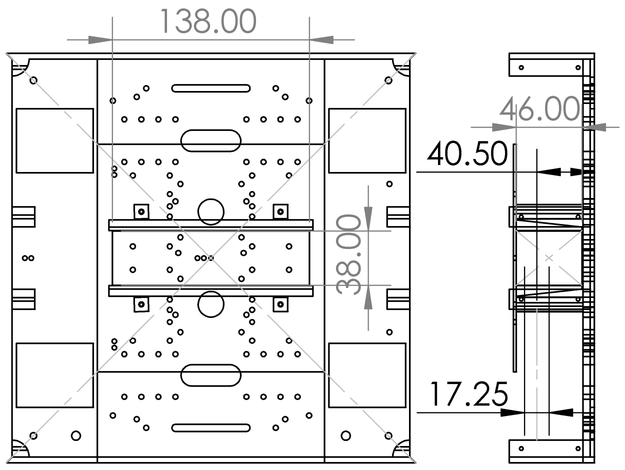
### 2.2.7 Battery

Choosing the right battery for a robot is crucial. It depends on power needs, weight limits, and how long it should operate. Lithium Polymer (LiPo) batteries seem ideal for 2.11a, mobile robots due to their low weight, high energy density, and discharge rate, providing sufficient power while keeping the robot's weight minimal [28]. A 3-cell, 12.6V, 25C, 5200mAh LiPo battery is used to power the the robot. As shown in Figure The

Table 2.9: Size, frame, and coordinate of the Battery

| **Property** | **Value** |
|---|---:|
| Frame | Battery |
| Length (mm) | 138 |
| Width (mm) | 38 |
| Height (mm) | 46 |
| Weight (gram) | 398 |
| Coordinates [x, y, z] (mm) | [0.00, 0.00, 17.25] |


(a) 3D trimetric view of the assembly



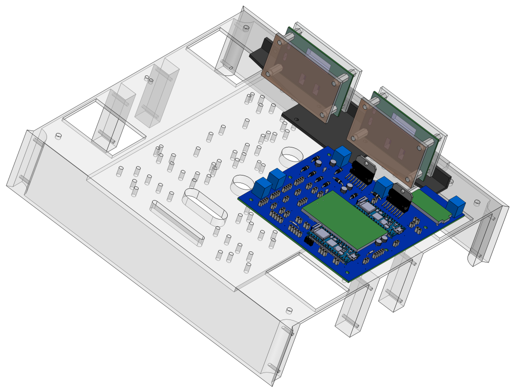
(b) 2D drawing from top and side view.

Figure 2.11: CAD model of the battery and battery mounts

Battery is mounted below the base to keep the robot's center of mass as low as possible. The battery holder is mounted to the robot base using four 47mm solid extenders that are 3D-printed using PETG and M3 bolts. The size, static frame, and coordinate information are summarized in Table 2.9. The robot is estimated to run for 80 minutes on a single charge. The battery can be installed and removed by detaching the side cover and sliding it between the mounts. During the testing process, this easy installation allows you to swap the Battery with a pre-charged battery, reducing the downtime required for charging.

### 2.2.8 Power and Battery Management System

One of the most important parts of a mobile robot is its power system, which controls its operational capacity, longevity, safety, and overall performance. The robot has three different charging options; the first uses a 12.6V DC power supply directly to the input power jack, the second uses the USB-C charging port, which converts the 5V DC power to 12.6V, and the third uses the output from the wireless charger. As shown in Figure 2.12, the USB-C charging module is integrated with the PCB, and the output from the wireless charger and DC power supply are connected to the ports on the PCB, which goes to the 3cell charging module and BMS. The BMS ensures even charging of each cell, over-discharge


(a) 3D trimetric view of the model

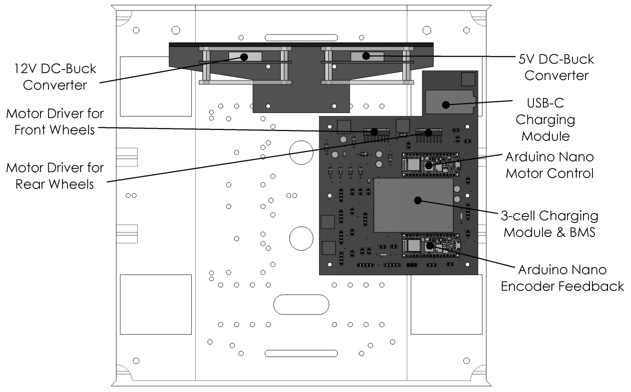
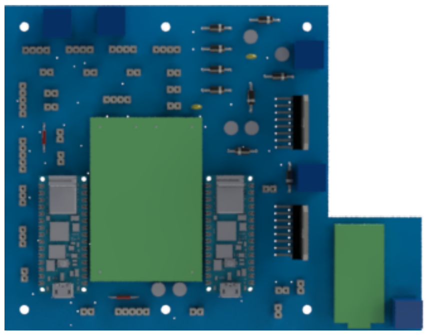

12V DC-Buck Converter
Motor Driver for Front Wheels
Motor Driver for Rear Wheels
5V DC-Buck Converter
USB-C Charging Module
Arduino Nano Motor Control
3-cell Charging Module & BMS
Arduino Nano Encoder Feedback
(b) 2D drawing from top and side view.

Figure 2.12: CAD model of the power supply, charging, and battery management system (BMS)

protection, over-current protection, over-charge protection, and over-temperature protection. The output of the Battery is connected to the DC-DC converter. The 12V DC-DC buck-boost converters ensure contact 12V irrespective of the input voltage; this ensures that the motor driver is always getting constant 12V. The high-power 5V DC-DC buck converter dedicately converts the 12V to 5V to run the Jetson Nano. While running the ROS and ML algorithms in Jetson Nano, it becomes very power-hungry. A dedicated power supply for the Jetson Nano ensures that it can operate at maximum processing speed. Figure 2.13 shows the connection of the power flow in the PCB assembly


(a) PCB designed view

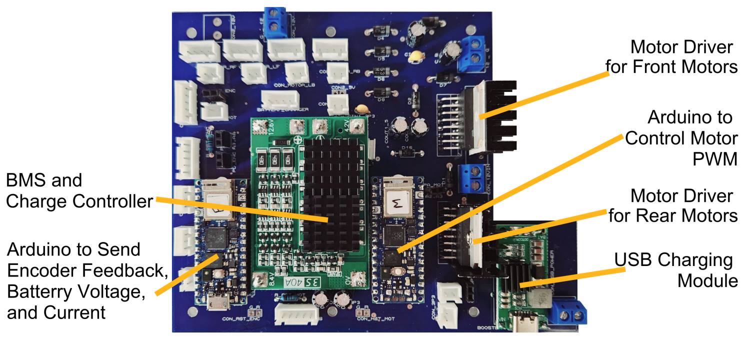

Motor Driver for Front Motors
Arduino to Control Motor PWM
BMS and Charge Controller
Motor Driver for Rear Motors
Arduino to Send Encoder Feedback, Batttery Voltage, and Current
USB Charging Module
(b) PCB after the assembly of the components

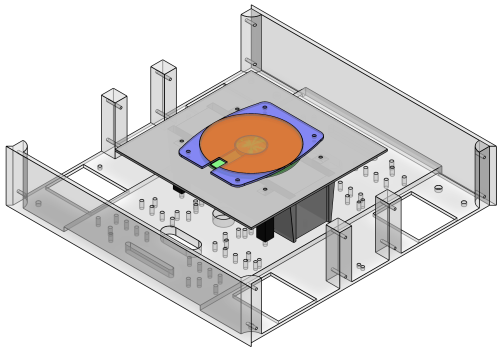
Figure 2.13: Printed Circuit Board (PCB)

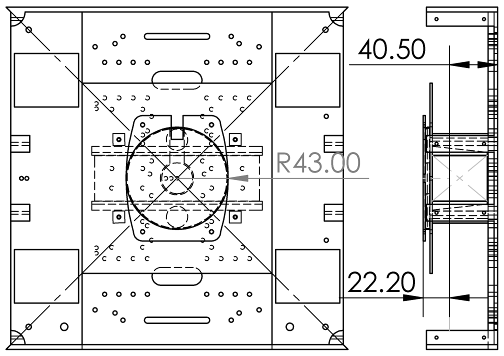
### 2.2.9 Wireless Charging System

A wireless charging system is integrated with the robot to enable uninterrupted operation of the robot. This is achieved by using a 3cell 12.6V wireless charging module. The ideal voltage of a fully charged battery is 12.6V, which is charged using a Constant Current / Constant Voltage (CC/CV) charging system. Until the charger reaches its maximum voltage($V_{B,max} = 4.2$V) per cell with a minimum voltage rating ($V_{B,min}$) of 3.7V, it will keep the charging rate steady. The battery's current capacity ($C_B$) given current-voltage ($V_B$ %) can be calculated using the (2.1). The Arduino-Nano for the encoder reads

$$C_B = (V_B - V_{B,min}) * 100/(V_{B,max} - V_{B,min}) \tag{2.1}$$

the current-voltage of the Battery using analog voltage sensor module and analog pin A0 (Battery-Voltage) of MCU_Encoder as shown in 2.6a. Then the MCU_Encoder sends the voltage to Jetson Nano; when the voltage level drops below threshold capacity $C_{B,th}$ the robot looks for the map's wireless charger docking station location, localizes the docking station, and attempts to dock itself for recharge. The charging system has to separate coils (transmitter and receiver) in 3D-printed cases. The transmitter coil is kept inside the docking station and is connected to a 12.6v power supply. As shown in Figure 2.14, the


(a) 3D trimetric view of the model



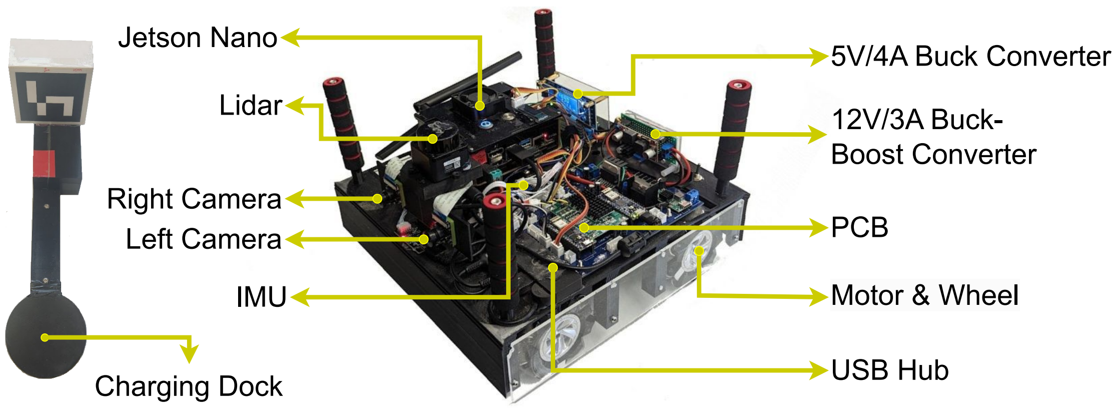
(b) 2D drawing from bottom and side view.

Figure 2.14: CAD model of the wireless charger receiver and mounts

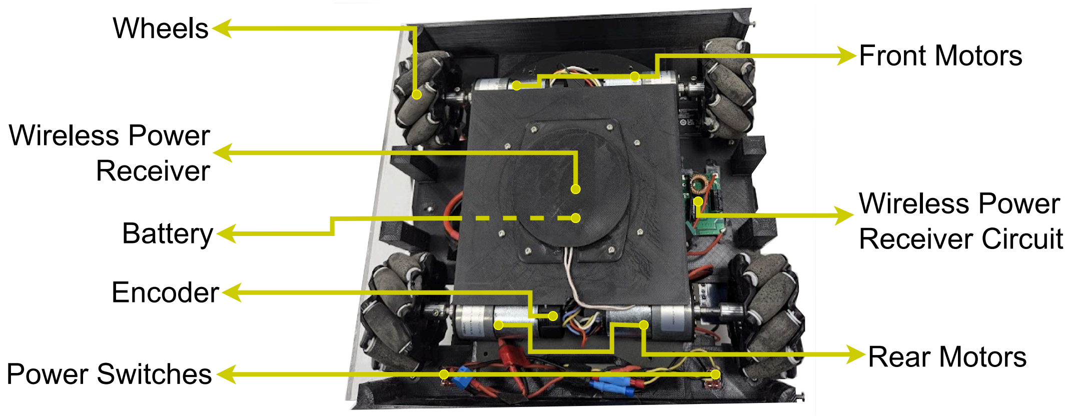
wireless charging receiver coil and circuit are mounted below the base and Battery to keep the minimum distance from the coil for maximum power transmission. The output from the receiver module is connected to the charging module and battery management system, which are responsible for charging the Battery.

### 2.2.10 Final Assembly

Figure 2.15 shows the final assembly of the robot. As shown in Figure 2.15a, the top components are assembled according to the design. Figure 2.15b shows the components mounted below the base. The components were assembled using M3 and M4 bolts of lengths 6mm, 10mm, 15mm, and 25mm.



(a) Top View of the Assembly



(b) Bottom View of the Assembly

Figure 2.15: Final-assembly of the robot

## 2.3 Physical Experiment and Results

These experiments provide valuable insights into the robot's physical capabilities and limitations, guiding future improvements and deployments in real-world applications. Two comprehensive experiments were conducted to evaluate the robot's payload capacity and its runtime under a single battery charge.

### 2.3.1 Payload Capacity

The first experiment aimed to determine the maximum payload capacity the robot could handle. The methodology involved incrementally adding weight to the robot and testing its ability to move. Initially, a baseline weight was placed on the robot, and the robot was commanded to move. If the robot successfully achieved a non-zero velocity, the weight was increased. This process involved adding *200g* increments to the robot and retesting its movement capabilities. This step-wise addition continued until the robot could no longer achieve a non-zero velocity, indicating it had reached its payload limit. Through this process, it was determined that the robot could carry a maximum payload of at least 5.4 kg, excluding the robot's own mass. This finding demonstrates the robot's robust design and its ability to handle substantial weights, making it suitable for various applications that may require transporting objects over short distances.

### 2.3.2 Battery Runtime

The second experiment focused on evaluating the robot's battery capacity and determining its runtime on a single battery charge. For this experiment, the robot was programmed to navigate to random goal points within a pre-built map of a room. This continuous movement simulation was designed to mimic real-world operating conditions, ensuring that the battery usage data collected would be representative of typical usage scenarios. The battery voltage was continuously recorded using a `rosbag` file, a standard logging format used in ROS for recording and storing data. The experiment began with the battery charged to 90% of its total capacity. The robot was allowed to operate until the battery charge level dropped to 30%. This lower threshold was chosen to prevent deep discharge, which can negatively impact battery lifespan. The recorded data showed that the robot operated for 1 hour and 33 minutes before the battery charge decreased from 90% to 30%. This runtime is significant for a mobile robot equipped with a 5200mAh battery, indicating that the robot can perform tasks for a considerable duration before requiring a recharge. The information gathered from this experiment is crucial for planning the operational schedules and charging cycles for the robot in various deployment scenarios.

### 2.3.3 Summary of Findings

- **Maximum Payload Capacity**: The robot can carry a maximum payload of at least 5.4 kg, excluding its own mass.

- **Battery Runtime**: Under continuous operation from 90% to 30% battery charge, the robot runs for approximately 1 hour and 33 minutes with a 5200mAh battery.

# CHAPTER 3

# SOFTWARE AND CONTROL SYSTEM DESIGN

## 3.1 Frame Attachment

For a 4-mecanum wheel robot, each wheel has its local coordinate frame attached to the center of the robot. A rectangular wheel configuration is used around the robot's center such that the origins of the robot and the four wheels are located in the same XY plane as shown in Figure 3.1.

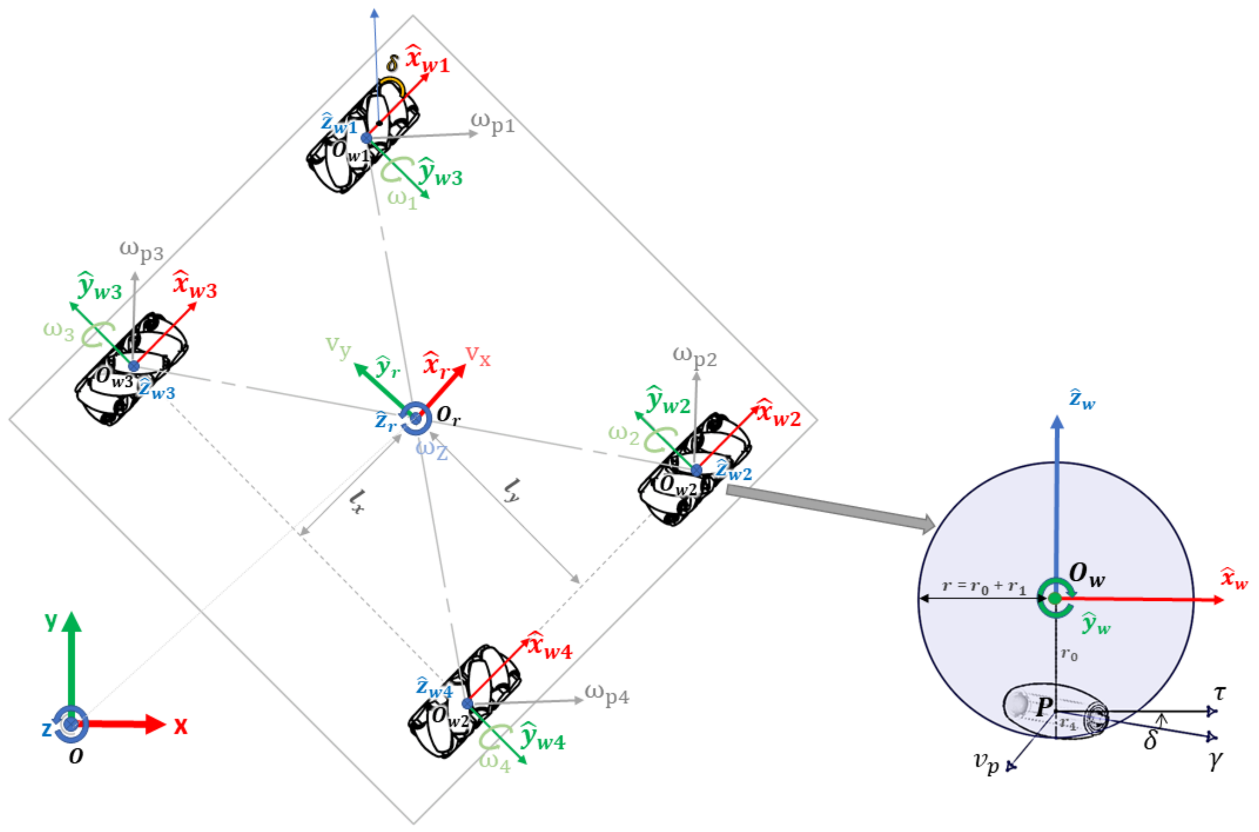


Figure 3.1: Cordinate frame modelling

The origins of the robot and wheels are defined as follows:

- $\mathbf{O_r} = [o_{r,x}, o_{r,y}, o_{r,z}]^T \in \mathbb{R}^{\mathbf{3\times 1}}$; Coordinates of the robot

- $\mathbf{O_{w1}} = [o_{w1,x}, o_{w1,y}, o_{w1,z}]^T \in \mathbb{R}^{\mathbf{3\times1}}$; Coordinates of the front-left wheel.
- $\mathbf{O_{w2}} = [o_{w2,x}, o_{w2,y}, o_{w2,z}]^T \in \mathbb{R}^{\mathbf{3\times1}}$; Coordinates of the front-right wheel.
- $\mathbf{O_{w3}} = [o_{w3,x}, o_{w3,y}, o_{w3,z}]^T \in \mathbb{R}^{\mathbf{3\times1}}$; Coordinates of the rear-left wheel.
- $\mathbf{O_{w4}} = [o_{w4,x}, o_{w4,y}, o_{w4,z}]^T \in \mathbb{R}^{\mathbf{3\times1}}$; Coordinates of the rear-right wheel.

Let the radius of the entire wheel with roller be r, which is the sum of roller radius ($r_1$) and the distance ($r_0$) of the origin of wheel $\mathbf{O_W}$ from the center of the roller $\mathbf{P}$ as shown in Figure 3.1. As the robot and wheel's origin lie in the same XY plane and the wheels are arranged in a rectangle, the absolute distance (L) between each wheel and origin is also equal. So, to simplify the modeling, the following information can be used:

- $o_{w1,z} = o_{w2,z} = o_{w3,z} = o_{w4,z} = o_{r,z}$
- $||o_{w1,x}\hat{\mathbf{x}} - o_{r,x}\hat{\mathbf{x}}|| = ||o_{w2,x}\hat{\mathbf{x}} - o_{r,x}\hat{\mathbf{x}}|| = ||o_{w3,x}\hat{\mathbf{x}} - o_{x}\hat{\mathbf{x}}|| = ||o_{w4,x}\hat{\mathbf{x}} - o_{r,x}\hat{\mathbf{x}}|| = l_x$
- $||o_{w1,y}\hat{\mathbf{y}} - o_{r,y}\hat{\mathbf{y}}|| = ||o_{w2,y}\hat{\mathbf{y}} - o_{r,y}\hat{\mathbf{y}}|| = ||o_{w3,y}\hat{\mathbf{y}} - o_{r,y}\hat{\mathbf{y}}|| = ||o_{w4,y}\hat{\mathbf{y}} - o_{r,y}\hat{\mathbf{y}}|| = l_y$
- L $= \sqrt{l_x^2 + l_y^2}$
- R $= l_x + l_y$

For manufacturing efficiency, cost-effectiveness, and simplifying analysis process, the offset angles ($\delta$) between the roller axis and lateral axis are uniformly set to the same value. Here, the value of $\delta$ is kept at 45°.

## 3.2 Inverse Kinematics

The control signal of the robot is defined with the robot's velocity matrix, $\mathbf{V} = [v_x, v_y, \omega_z]^T \in \mathbb{R}^{\mathbf{3\times1}}$, where $v_x$ is the magnitude of longitudinal velocity of the robot along the X-axis of the robot, $v_y$ is the magnitude of lateral velocity of the robot along the Y-axis of the robot, and $\omega_z$ is the magnitude of angular velocity of the robot along the Z-axis of the robot. The angular velocity of wheels along the Y-axis is given as $\mathbf{\Omega} = [\omega_1, \omega_2, \omega_3, \omega_4]^T \in \mathbb{R}^{\mathbf{4\times1}}$. Now, for a given desired velocity ($\mathbf{V}$), the corresponding angular velocity ($\mathbf{\Omega}$) of the wheel can be written as follows [29]:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -R \\ 1 & 1 & R \\ 1 & 1 & -R \\ 1 & -1 & R \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{3.1}$$

The compact form of equation (3.1) can be expressed as:

$$\mathbf{\Omega} = \mathbf{J_V V} \tag{3.2}$$

where, $\mathbf{J_V} \in \mathbb{R}^{\mathbf{4\times3}}$ is the Jacobian matrix, and $\mathbf{V}$ the command velocity matrix.

## 3.3 Forward Kinematics

From the (3.1), the forward kinematics for the given angular velocity of the motor, $\mathbf{\Omega}$, the corresponding robot's velocity matrix $\mathbf{V}$, can be found as follow:

$$\mathbf{V} = \mathbf{J_V}^{\dagger}\mathbf{\Omega} \tag{3.3}$$

Here, the $\mathbf{J_V}^{\dagger} \in \mathbb{R}^{\mathbf{3\times4}}$ represents the pseudo inverse of the Jacobian matrix $\mathbf{J_V}$ and given as:

$$\mathbf{J_V}^{\dagger} = (\mathbf{J_V}^{\mathbf{T}}\mathbf{J_V})^{-1}\mathbf{J_V}^{\mathbf{T}} \tag{3.4}$$

Now, following the convention of the forward kinematics of (3.1), the expanded form of (3.3) can be written as follows:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_0 \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ \frac{-1}{R} & \frac{1}{R} & \frac{-1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{3.5}$$

## 3.4 Motor Driver Control

Two L298N motor drivers are used for the robot. The L298N motor driver is a dual H-bridge module capable of controlling two DC motors with up to 2A per channel, featuring built-in diodes for back EMF protection [30]. Its ease of use with Arduino and better power

Table 3.1: Motor Driver Connection

| **Component** | **Connection** | **Pin** |
|---|---|---|
| Motor Driver 1 (Motor A Front right- RF) | IN1 to pinMotRF1<br>IN2 to pinMotRF2<br>ENA to pinMotRFpwm<br>OUT1 and OUT2 | Digital pin 3<br>Digital pin 4<br>Digital pin 2<br>Motor A terminals |
| Motor Driver 1 (Motor B Front Left- LF) | IN4 to pinMotLF1<br>IN3 to pinMotLF2<br>ENB to pinMotLFpwm<br>OUT1 and OUT2 | Digital pin 7<br>Digital pin 5<br>Digital pin 6<br>Motor B terminals |
| Motor Driver 2 (Motor C Rear Left- LB) | IN2 to pinMotLB1<br>IN1 to pinMotLB2<br>ENA to pinMotLBpwm<br>OUT1 and OUT2 | Digital pin 10<br>Digital pin 8<br>Digital pin 9<br>Motor C terminals |
| Motor Driver 2 (Motor D Rear right- RB) | IN4 to pinMotRB1<br>IN3 to pinMotRB2<br>ENB to pinMotRBpwm<br>OUT1 and OUT2 | Analog pin A1<br>Analog pin A0<br>Digital pin 11<br>Motor D terminals |
| Power | VCC of Driver<br>Gnd of Driver | +12V<br>GND |

handling capacity makes it ideal for motor control applications for the robot. The motors are controlled using input pins (IN1, IN2, IN3, IN4) that determine the direction, and PWM pulses in the enable pins (ENA, ENB) determine the speed of the connected motors. To control motor direction, set IN1 HIGH and IN2 LOW for one direction and IN1 LOW and IN2 HIGH for the opposite; both HIGH or LOW stop the motor, IN3 and IN4 control motor B. For speed control, PWM signals on ENA and ENB adjust the speeds of motors A and B, respectively.

## 3.5 Motor Controller Model

At the motor controller level, an 8-bit PWM signal is used to control the motors, $\mathbf{\Omega}$ by controlling the input voltage of the motors, $\mathbf{V_M} \in \mathbb{R}^{\mathbf{4\times1}}$ shown in (3.6),

$$\mathbf{V_M} = [v_{M1}, v_{M2}, v_{M3}, v_{M4}]^T \tag{3.6}$$

from the corresponding PWM value, $\mathbf{\Omega_{pwm}} \in \mathbb{Z}^{\mathbf{4\times1}}$ shown in (3.7),

$$\mathbf{\Omega_{pwm}} = [\omega_{pwm,1}, \omega_{pwm,2}, \omega_{pwm,3}, \omega_{pwm,4}]^T \tag{3.7}$$

The relation between $\mathbf{V_M}$ and $\mathbf{\Omega_{pwm}}$ is shown in (3.8).

$$\mathbf{V_M} = \mathbf{\Omega_{pwm}} \cdot \frac{v_{m_{max}} - v_{m_{min}}}{\omega_{pwm_{max}} - \omega_{pwm_{min}}} \tag{3.8}$$

where $v_{m_{max}}$, $v_{m_{min}}$, $\omega_{pwm_{max}}$, $\omega_{pwm_{min}}$ are maximum motor input voltage, minimum motor input voltage, maximum PWM value, and minimum PWM value respectively. Initially, by looking at the data of $\mathbf{\Omega}$ vs $\mathbf{\Omega_{pwm}}$ curve, it seemed that it followed a hyperbolic relation. So, to estimate $\mathbf{V_M}$ from a desired motor speed $\mathbf{\Omega}$, a hyperbolic function is used as shown in (3.9).

$$\mathbf{\Omega_{pwm}} = (\mathbf{\Omega} - \mathbf{B})\mathbf{C}^{\dagger} \tag{3.9}$$

where $\mathbf{C} = [c_1, c_2, c_3, c_4]^T \in \mathbb{R}^{\mathbf{4\times1}}$, and $\mathbf{B} = [b_1, b_2, b_3, b_4]^T \in \mathbb{R}^{\mathbf{4\times1}}$ are hyper-parameters to control the hyperbolic function. The objective is to find the optimized value of $\mathbf{B}$ and $\mathbf{C}$

for n number of samples, for which the least square mean (LSM) objective function($\mathbf{E} = [e_2, e_2, e_3, e_4]^T \in \mathbb{R}^{\mathbf{4\times1}}$) is minimum.

$$\mathbf{E} = \frac{1}{n}\sum_{i=0}^{n}((\mathbf{\Omega_i} - \mathbf{B})\mathbf{C}^\dagger - \mathbf{\Omega_{pwm,i}})^2 \tag{3.10}$$

Combining (3.2) and (3.9), the desired PWM values($\mathbf{\Omega_{pwm}}$) for control velocity($\mathbf{V}$) is expressed as (3.11)

$$\mathbf{\Omega_{pwm}} = (\mathbf{J_V}\mathbf{V} - \mathbf{B})\mathbf{C}^\dagger \tag{3.11}$$

The final $\mathbf{\Omega_{pwm}}$ is calculated from control velocity($\mathbf{V}$) sent to the motor driver, which generates the corresponding motor input voltage($\mathbf{V_M}$).

## 3.6 Motor System Identification

For identifying the motor model, the PWM signal($\mathbf{\Omega_{pwm}}$) of the four motors is sent to the Arduino-motor microcontroller as string data using serial port and corresponding motor speed($\mathbf{\Omega}$) from the Arduino-encoder is recorded. This step is repeated for PWM values ranging from -255 to 255. For $[-55 < \omega_{pwm,i} < 55]$, the input voltage is not enough to start the motor, and these values are excluded from the sample dataset. As for these values, the corresponding $\omega_i$ is zero, so it makes it impossible to calculate $\mathbf{C}$. To solve these, the dataset is defined as the union of the positive PWM values ranging from 55 to 255 and negative PWM values ranging from -255 to -55, where $\omega_{pwm,i} \in [-255, -55] \cup [55, 255]$. To simplify more, the data is sampled at an increment of 5, making a total number of samples(n) 80, and the $\mathbf{B}$ and $\mathbf{C}$ are calculated for positive and negative velocity individually. Table 3.2 shows the optimized hyperparameters for positive rotation and Negative rotation.

Table 3.2: Hyper Parameters for the Motor Model

| Parameters (B, C) | Positive Speed ($\omega_{pwm} > 0$) | Negative Speed ($\omega_{pwm} < 0$) |
|---|---|---|
| $b_1$ | 130.63 | -129.32 |
| $b_2$ | 132.61 | -133.68 |
| $b_3$ | 137.10 | -130.08 |
| $b_4$ | 129.74 | -132.54 |
| $c_1$ | -4096.39 | -4089.71 |
| $c_2$ | -4446.28 | -4585.02 |
| $c_3$ | -4431.14 | -4080.27 |
| $c_4$ | -3954.66 | -4167.72 |

## 3.7 Effectiveness of motor system identification

To evaluate the effectiveness of the motor system identification (described in Section 3.6), the robot was instructed to follow a constant angular velocity while tracing a desired circular trajectory with a radius of 0.65 m. The performance of two controllers was assessed based on their accuracy in tracking the desired trajectory:

- Controller incorporating system identification.
- Controller operating without system identification.

In the case of the *controller with system identification*, the robot controller used the estimated parameters $b_i$ and $c_i$ from Table 3.2, and the robot's position per second was recorded. In the case of *controller without system identification*, the robot controller used a linear model from $\omega_i$ to $u_{pwm,i}$ with the slope estimated from the range of wheel velocities and corresponding range of the PWM signal. The trajectories followed by the robot are shown in Figure 3.2. The **Top** figure is the trajectories of the robot, and the **Bottom** figure is the deviation from the expected trajectory with and without using the parameters from motor system identification. The root mean squared (RMS) deviation of the trajectory without system identification is 0.297 m, which is 5.6 times higher than the deviation obtained using our controller model with system identification (0.053 m). In comparison to the expected trajectory's radius of

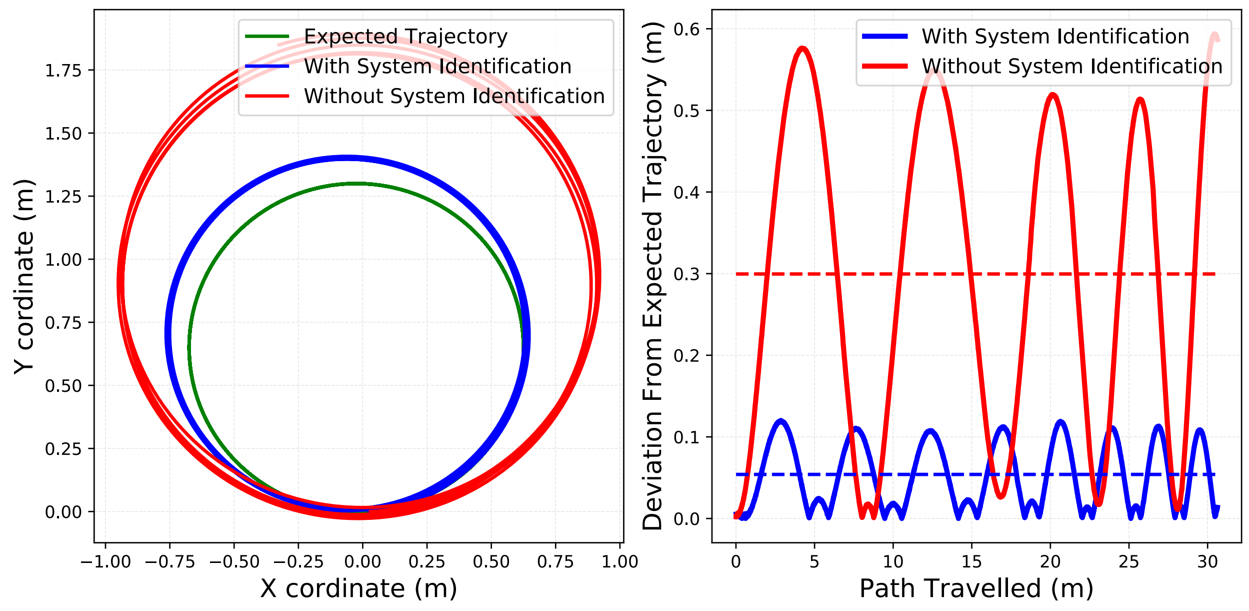


Figure 3.2: Robot's trajectory and path deviation with and without motor system identification.

0.65 m, the deviation accounts for 8.15% with our controller model and 45.69% without using system identification. This demonstrates that system identification has enabled us to accurately track desired paths while minimizing errors that a high-level feedback loop can offset. Another option that wasn't incorporated is to use a feedback controller, where the PWM signal is adjusted based on feedback from wheel encoders. One disadvantage of not utilizing a feedback controller is that if the motor's internal parameters or the robot's parameters change over time, the model may not perform as well as the estimated parameters.

## 3.8 Software System

Software design is crucial for making our robot autonomous. This section details the software environment, framework, and specific components utilized in the Omobot project. The software is installed on NVIDIA's Jetson Nano, using the latest Linux kernel supported by Jetson Nano, NVIDIA L4T 32.7.1, which is part of NVIDIA's JetPack-4.6.1. This setup includes the ARM64-based Ubuntu-18.04 Linux distribution. The primary software framework used is the Robotic Operating System (ROS) Melodic [31], which is the latest version of ROS supported by Ubuntu-18.04. Jetson Nano supports several communication protocols to interface with peripheral devices and sensors, including UART, USB, I2C, IP, and CSI. The overall ROS-based software framework interacting with sensors, motors, joystick, and internet is shown in Figure 3.3.

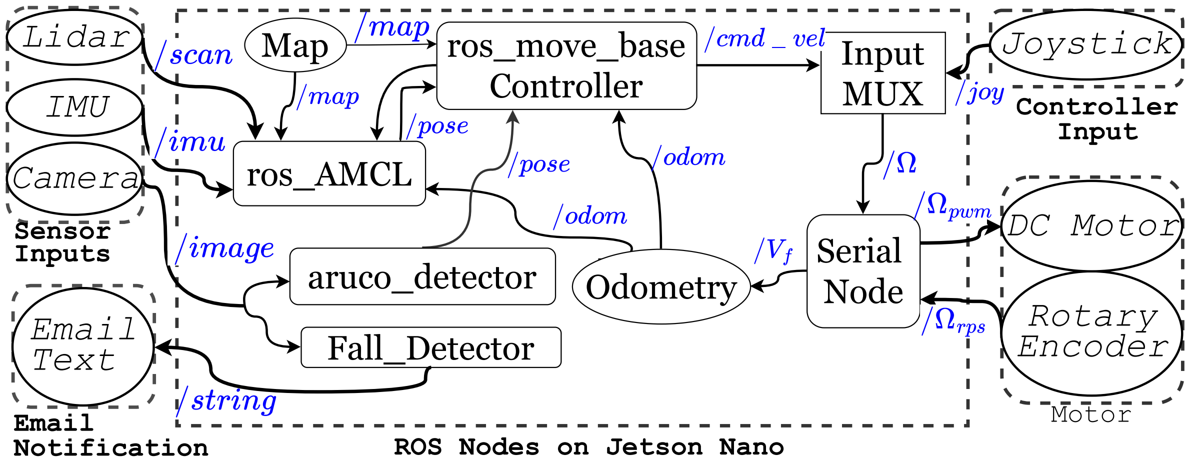


Figure 3.3: Communication among sensors and devices with ROS nodes running on Jetson Nano.

### 3.8.1 Installation and Setup

The first step is setting up the Jetson Nano with its JetPack image, followed by the installation of ROS Melodic. This process involves configuring the system to accept software from packages.ros.org and setting up the necessary GPG keys. The ROS Desktop package, which includes tools like rqt and rviz, is then installed. It is recommended that ROS environment variables are loaded automatically during new shell sessions by updating the .bashrc script.

Additionally, the rosdep tool is installed and initialized to manage system dependencies for ROS packages. A catkin workspace is then created and configured, which is necessary for running and developing ROS packages. This workspace allows the installation of other ROS packages from the source, enabling further customization and functionality of the robot.

### 3.8.2 ROS Topic Flow

The ROS topic flow shown in the diagram represents a comprehensive system for autonomous navigation, localization, and perception. Each topic plays a specific role in ensuring the robot can navigate safely and effectively within its environment.

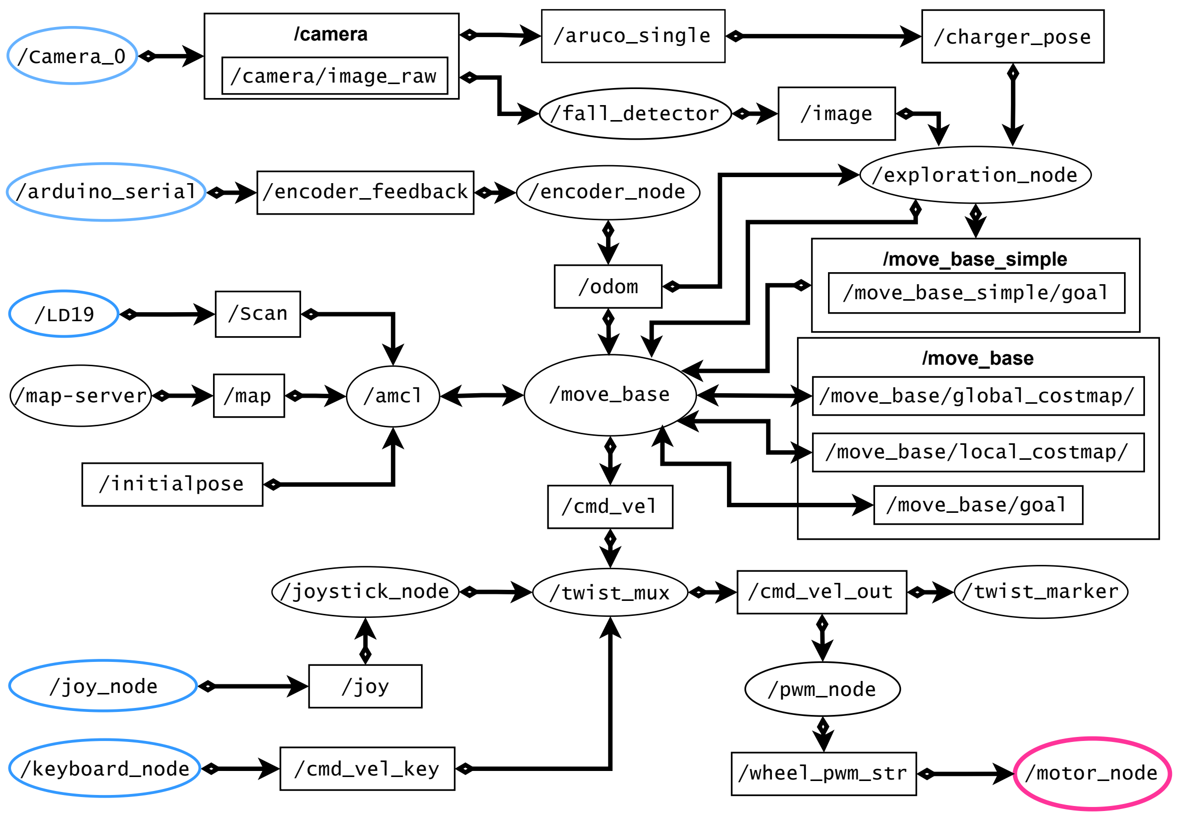


Figure 3.4: ROS Topics Graph

The use of sensors like LiDAR, cameras, and encoders, combined with advanced algorithms like Simultaneous Localization and Mapping (SLAM) and Adaptive Monte Carlo Localization (AMCL), enables the robot to build maps, localize itself, perceive its surroundings, and navigate towards goals autonomously.

#### 3.8.2.1 Localization:

- /scan: This topic is used for publishing laser scan data from the LiDAR. The scan data is used for SLAM and obstacle detection.
- /map: The map topic is used to publish the map of the environment. This map is generated during the mapping phase and is used by the localization system to understand the robot's position within the environment.
- /amcl: AMCL uses laser scans and the map to provide accurate localization data of the robot in the environment.

#### 3.8.2.2 Perception:

- /camera/image_raw: This topic publishes raw image data from the camera. It is used for detecting falls and the pose of wireless charger.
- /aruco_single: This topic is used for detecting ARUCO marker, which is utilized for localization and identification of wireless charger.

#### 3.8.2.3 Navigation:

- /odom: This topic publishes odometry information, which includes the robot's position and velocity over time. It is used to track the robot's movement inside the map.
- /move_base: The move_base node provides the navigation capabilities to the robot. It takes in goals (/move_base_simple/goal, /odom, local_costmaps, and global_costmap) data to generate the safe path to navigate the robot in the environment.
- /cmd_vel: This topic publishes velocity commands generated by the move_base navigation stack to control the robot's movement.

- /move_base_simple/goal: This topic is used to send simple poses as navigation goals to the robot.

#### 3.8.2.4 Input/Output, Control, and Fall Detection:

- /joint_states: This topic publishes the state of the robot's joints, which are used for kinematic calculations and motion planning.

- /encoder_feedback: This topic publishes feedback from the wheel encoders, which are used to calculate odometry.

- /pwm_node: This node is used for generating the PWM values for all the wheels.

- /motor_node: This node is used to send the motor's PWM values to the arduino_motor using serial communication and to monitor the motor's state.

- /fall_detector: This ROS node subscribes to the camera/image_raw topic to receive image messages. Upon receiving an image, it converts the ROS image message and uses the image for fall detection.

- /exploration_node: This node is used to control the intermediate exploration, fall detection, and charge monitoring. Depending on the situation, the exploration either explores, stops to send fall detection data, or docks for charging.

- /joy_node: This node is used to receive the data from the joystick to manually control the robot.

- /keyboard_node: This node is used to receive the data from the keyboard to manually control the robot.

## 3.9 Operational Experiment and Results

### 3.9.1 Odometry Accuracy Experiment

The experiments conducted demonstrate the robot's capabilities in mapping and odometry accuracy using low-cost components. The use of `ros-slam_toolbox` proved effective in correcting odometry errors and maintaining accurate mapping. The mapping experiments were performed in a $12.45 \times 9.95$ sq m room at the University of Maine (Barrows Hall, Rm 201). The room was first mapped using the `ros-slam_toolbox` library. The `ros-slam_toolbox` utilizes LiDAR observations along with the robot's position derived from wheel odometry to create a 2D map, continuously updating it to ensure accuracy. The loop closure technique is employed to correct errors and ensure the map accurately represents the environment. This method corrects the odometry to align the observed LiDAR scans with the expected LiDAR scans from the map. Figure 3.5 illustrates the mapping process and the trajectories of the robot and its odometry. The **Top** part of the figure shows the robot's trajectory in green arrows, indicating the origin of the robot while creating the map. The red arrows represent the trajectory of the odometry's origin. The **Bottom** part of the figure displays the deviation of the robot's and odometry's pose (translation in XY axis, orientation with respect to the Z axis) from the map's origin.

To compare the accuracy of wheel odometry with the trajectory provided by `ros-slam_toolbox`, the robot was navigated in a complete loop around the room, ensuring it returned to the starting point. The map and the trajectories are depicted in Figure 3.5. In the grid map, black represents space occupied by obstacles, white indicates free space, gray represents areas yet to be updated, and the purple dots correspond to the LiDAR scans. The LiDAR scans align closely with the walls and obstacles, demonstrating the accuracy of the `ros-slam_toolbox` trajectory. The wheel odometry trajectory is shown in red, and the corrected `ros-slam_toolbox` trajectory is shown in green. The odometry's estimated pose

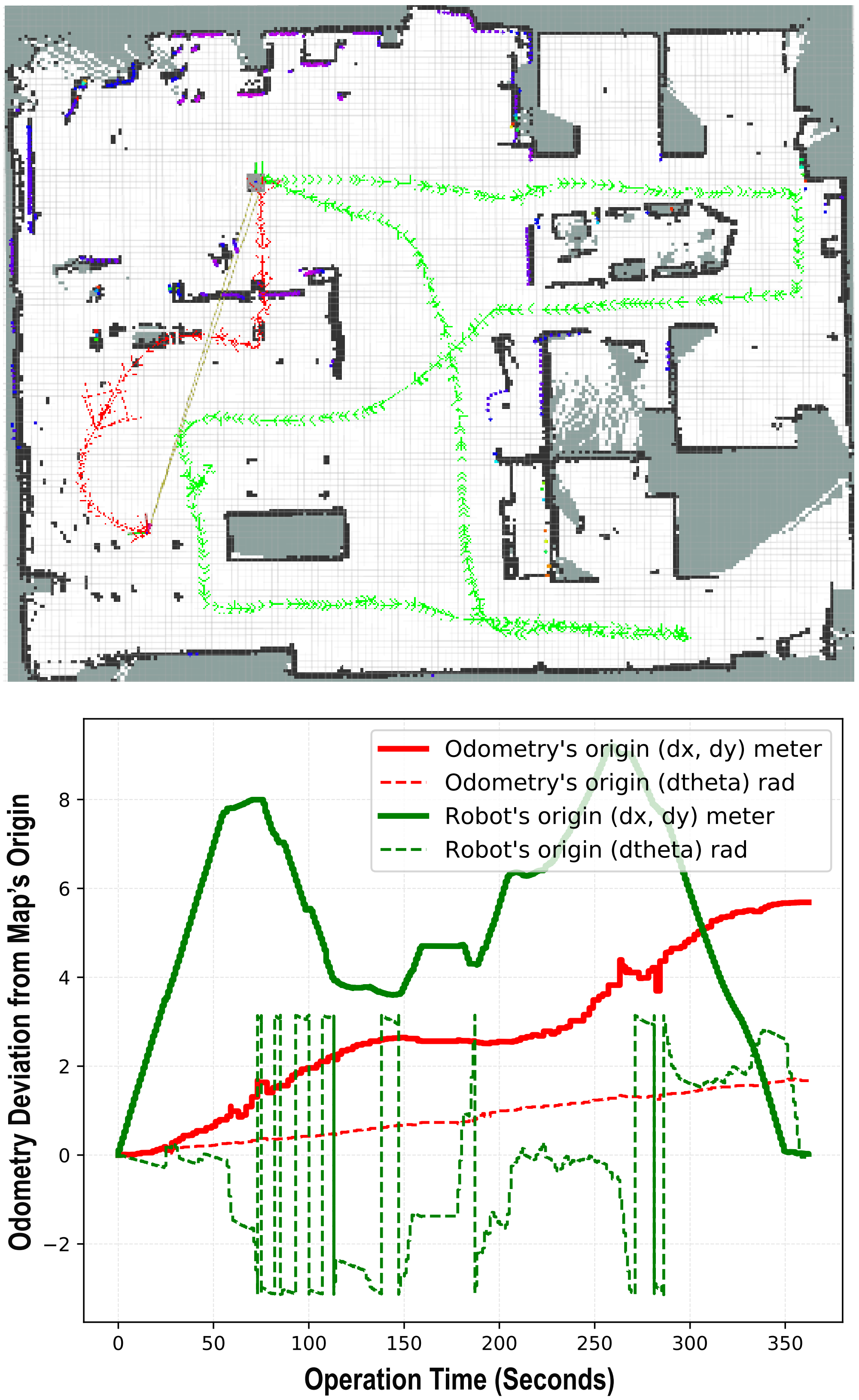


Figure 3.5: Mapping using LiDAR, Wheel Odometry, and `ros-slam_toolbox`.

at the end of the trajectory deviated from the starting pose by 5.64 meters in translation and 1.66 radians in orientation.

From this experiment, it was found that using an inexpensive LiDAR does not adversely affect performance. However, the odometry data generated using the inexpensive wheel encoder exhibited notable errors. Despite these errors, the `ros-slam_toolbox` was able to mitigate them, maintaining mapping accuracy in the room. It is well understood that LiDAR-based mapping does not perform well in environments with long featureless corridors. In such situations, reliance on wheel odometry increases, highlighting the trade-offs of using low-cost wheel encoders. The room where the map was generated had numerous features, reducing dependency on wheel odometry. However, in environments like long straight corridors with limited features, the robot's performance in mapping and localization may be compromised.

### 3.9.2 Obstacle Avoidance Efficiency

For this experiment, a detailed map of the room was constructed and manually annotated with landmark locations as described in Section 3.9.1. The primary goal was to evaluate the robot's efficiency in navigating to random landmark locations while detecting and reporting any falls observed during the exploration. The robot was tasked with navigating to various randomly selected landmarks on the map. During this process, it was also required to detect any falls and report them accurately. The flowchart detailing the map exploration, fall detection, and reporting workflow is shown in Figure 3.6. The landmark locations (A-G) and the trajectories of the robot during 30 minutes of operation are shown in green in Figure 3.7. The robot's performance was meticulously monitored to assess its obstacle avoidance efficiency and overall navigation robustness. During the 30-minute experiment, the robot encountered only one collision. This collision incident can be attributed to the inherent limitations of the 2D map created by the LiDAR scan, which may not perfectly

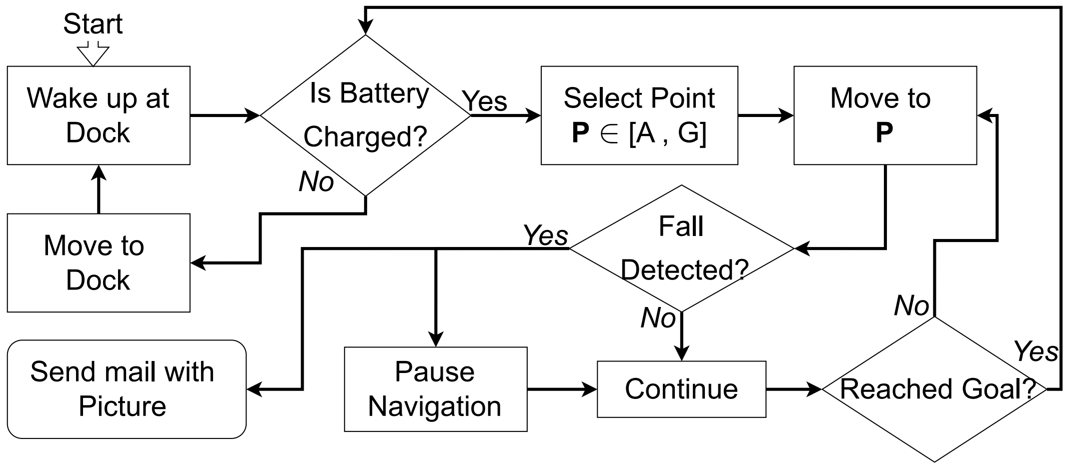


Figure 3.6: Robot's workflow

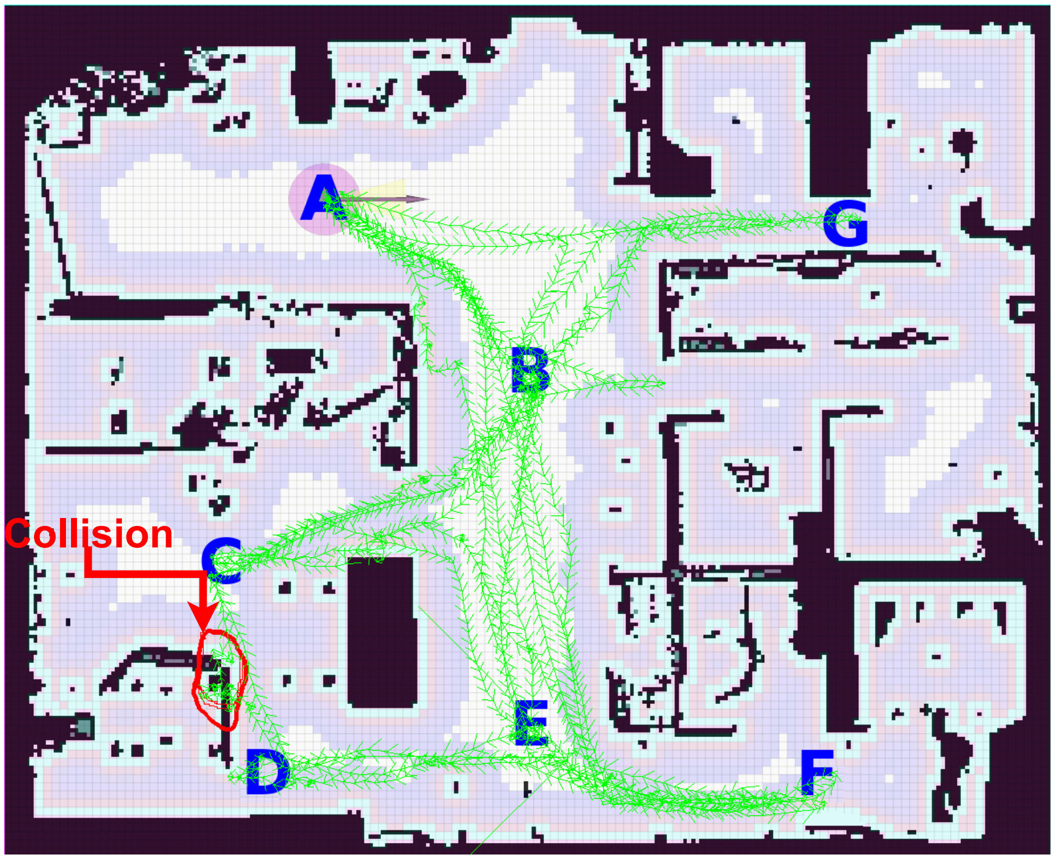


Figure 3.7: Occupancy grid map Visualization.

represent the 3D reality of obstacles in the environment. For example, the LiDAR might capture the narrow stem of an office chair as an obstacle, but the robot may collide with the wider feet of the chair, which extend beyond the stem captured by the LiDAR. The results indicate that the robot's obstacle avoidance system is highly efficient, given that it only encountered a single collision in 30 minutes. The black pixels in the occupancy grid map indicate static obstacles, the white pixels represent accessible areas, and the purple pixels denote inflated obstacles. The landmarks (A-G) are manually marked locations, and the green lines show the robot's trajectories during the robustness of the autonomous navigation experiment. The robot's ability to navigate and avoid obstacles efficiently is critical for ensuring its reliability and safety in real-world applications. The single collision observed highlights a potential area for improvement in the obstacle detection system, particularly in differentiating between the 2D and 3D aspects of obstacles.

### 3.9.3 Summary

While the robot's performance in this controlled environment was promising, certain limitations were observed. The reliance on a 2D LiDAR scan means that certain obstacles, particularly those with significant 3D features, may not be adequately represented. This can lead to occasional collisions, as seen in the experiment. Additionally, the effectiveness of the obstacle avoidance system in more complex and cluttered environments remains to be tested. Environments with fewer distinguishable features, such as long, featureless corridors, may pose more significant challenges for the robot's navigation system. In such scenarios, the robot's reliance on wheel odometry may increase, potentially affecting its overall performance.

The experiment demonstrated that the robot's obstacle avoidance system is highly efficient, with only one collision occurring during 30 minutes of operation. This indicates a robust design capable of navigating and avoiding obstacles in a typical indoor environment. However, the observed limitations suggest areas for future improvement, particularly in enhancing the robot's ability to detect and navigate around 3D obstacles.

# CHAPTER 4
# FALL DETECTION SYSTEM

## 4.1 System Overview

The fall detection system in Omobot is a crucial component aimed at ensuring the safety of elderly individuals by identifying falls and notifying caregivers promptly. This system utilizes advanced techniques in image processing for accurate fall detection. The following sections provide a comprehensive overview of the system, including mathematical formulations, algorithms, and implementation details. The fall detection system consists of three main stages:

- **Pre-processing**: Transforming the raw input images to enhance detection accuracy.
- **Person Detection and Pose Estimation**: Using YOLOv8-Pose for identifying individuals and estimating their body poses.
- **Fall Detection**: Determining if a detected person has fallen based on their pose.

## 4.2 Pre-processing

The robot uses YOLOv8-Pose [18] for pose estimation. This model is trained on the MS-COCO dataset [32]. Like most photographs found online, images in the MS-COCO dataset are taken from the human point of view (POV). Given the extremely low POV of The robot (0.15m from ground), this reduces the accuracy of The robot's ability to extract pose. The Experiment in Section 4.5.2 confirms this intuition. One way to address this problem is by data augmentation during the model's training [33]. In this work, instead transforming images from the robot point of view (POV) to an average human POV before pose estimation are adopted. This has the advantage of avoiding the additional step of retraining the model. An example of this transformation is shown in Figure 4.4.

### 4.2.1 Using homography for Converting robot viewpoint images to human viewpoint

To transform robot viewpoint images into human viewpoint Homography is used. Homography is the transformation of images from one perspective projection into another. A homography transform preserves straight lines as straight lines but parallel lines may not be preserved as parallel.

To perform this transformation, consider a 3D location $\mathbf{x} \in \mathbb{R}^{\mathbf{3}}$ in space. Let the plane $\mathcal{P}$, on which the points $\mathbf{x_1}$ lie, be described by a unit normal vector $\hat{\mathbf{n}} \in \mathbb{R}^3$ and the distance $h \in \mathbb{R}$ from the origin so that the plane is defined as $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^{\mathbf{3}} \mid \hat{\mathbf{n}}^\top \mathbf{x} = \mathbf{h}\}$. Let any point $\mathbf{x_1} \in \mathcal{P}$ on the plane projected to two cameras $\lambda_1 \mathbf{u}_1 = K_1 \mathbf{x}_1$ and $\lambda_2 \mathbf{u}_2 = K_2({}^2R_1 \mathbf{x}_1 + {}^2\mathbf{t}_1)$. where $K_1 \in \mathbb{R}^{3\times3}$ and $K_2 \in \mathbb{R}^{3\times3}$, are the intrinsic matrices of the cameras and ${}^2R_1 \in \mathrm{SO}(3)$ and ${}^2\mathbf{t_1} \in \mathbb{R}^{\mathbf{3}}$ are relative rotation and translation from camera 1 to camera 2. Then, the homography matrix ${}^2H_1(\hat{\mathbf{n}}, h)$ can be computed that maps a point $\mathbf{u_1}$ in image 1 to image 2 $\alpha \mathbf{u}_2 = {}^2H_1 \mathbf{u}_1$ as,

$$ {}^2H_1(\hat{\mathbf{n}}, h) = hK_2{}^2R_1 K_1^{-1} + K_2{}^2\mathbf{t}_1 \hat{\mathbf{n}}^\top K_1^{-1}. \tag{4.1} $$

The proof of above equation is provided in the next section. Since the homography depends on the location of the plane, therefore, multiple homographies corresponding to different distances $h$ between minimum and maximum of the LiDAR scan are sampled. Then the tomography that gives the highest confidence detection by YOLOv8 is picked.

### 4.2.2 Homography from equation of a plane

From pinhole camera projection equations for camera 1 (Robot's POV) and 2 (Human's POV) the following equation can be defined [34]:

$$ \lambda_1 \mathbf{u}_1 = K_1 \mathbf{x}_1, \qquad \lambda_2 \mathbf{u}_2 = K_2 \underbrace{({}^2R_1 \mathbf{x}_1 + {}^2\mathbf{t}_1)}_{\mathbf{x}_2} \tag{4.2} $$

where $\lambda_1, \lambda_2 \in \mathbb{R}$ are scalars that represent the depth of $\mathbf{x}_1$ from the respective camera origins and $\mathbf{x}_2 \in \mathbb{R}^3$ are the coordinates of the point $\mathbf{x}_1$ in the frame of camera 2. $\lambda_1$ can solved by computing the intersection of the ray $\mathbf{x}_1 = \lambda_1 \mathbf{K}_1^{-1}\mathbf{u}_1$, with the plane $P$.

$$\hat{n}^\top(\lambda_1 \mathbf{K}_1^{-1}\mathbf{u}_1) = h \implies \lambda_1 = \frac{h}{\hat{n}^\top \mathbf{K}_1^{-1}\mathbf{u}_1} \tag{4.3}$$

Now, substituting $\lambda_1$ in equation 4.2 and rearranging the equation to form the homography matrix $H$:

$$\begin{aligned}
\lambda_2 \mathbf{u}_2 &= \frac{h}{\hat{\mathbf{n}}^\top K_1^{-1}\mathbf{u}_1} K_2{}^2R_1 K_1^{-1}\mathbf{u}_1 + K_2{}^2\mathbf{t}_1 \\
(\hat{\mathbf{n}}^\top K_1^{-1}\mathbf{u}_1)\lambda_2\mathbf{u}_2 &= hK_2{}^2R_1K_1^{-1}\mathbf{u}_1 + K_2{}^2\mathbf{t}_1(\hat{\mathbf{n}}^\top K_1^{-1}\mathbf{u}_1) \\
\underbrace{(\hat{\mathbf{n}}^\top K_1^{-1}\mathbf{u}_1\lambda_2)}_{\alpha}\mathbf{u}_2 &= \underbrace{(hK_2{}^2R_1K_1^{-1} + K_2{}^2\mathbf{t}_1\hat{\mathbf{n}}^\top K_1^{-1})}_{^2H_1}\mathbf{u}_1.
\end{aligned} \tag{4.4}$$

In the last step, $\hat{n}^\top \mathbf{K}_1^{-1}\mathbf{u}_1\lambda_2$ can be absorbed into $\alpha$ as a scaling factor because $\alpha$ can absorb an arbitrary non-zero scalar function of $\mathbf{u}_1$.

To justify this possibility of $\alpha$, $\alpha\mathbf{u}_2 = 2H_1\mathbf{u}_1$ can be rewritten with $\alpha$ as a multiple of an arbitrary non-zero scalar function $f(\mathbf{u}_1) : \mathbb{P}^2 \to (\mathbb{R} \setminus \{0\})$ of $\mathbf{u}_1$ with $\alpha = \gamma f(\mathbf{u}_1)$ where $\gamma \in \mathbb{R}$ is a scale factor:

$$\gamma f(\mathbf{u}_1)\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{h}_1^\top \\ \mathbf{h}_2^\top \\ \mathbf{h}_3^\top \end{bmatrix}\mathbf{u}_1 = {}^2H_1\mathbf{u}_1 \tag{4.5}$$

Solving for $\gamma$ can be equate for $\gamma f(\mathbf{u}_1) = \mathbf{h}_3^\top\mathbf{u}_1$ which gives,

$$\gamma = \frac{\mathbf{h}_3^\top\mathbf{u}_1}{f(\mathbf{u}_1)} \tag{4.6}$$

Substituting $\gamma$ in equation 4.5, the original definition of the homography transform is got:

$$\frac{\mathbf{h}_3^\top \mathbf{u}_1}{f(\mathbf{u}_1)} f(\mathbf{u}_1) \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \underbrace{\mathbf{h}_3^\top \mathbf{u}_1 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}}_{\alpha'} = {}^2H_1 \mathbf{u}_1 \tag{4.7}$$

When implementing the homography on the robot and in practice use equation 4.4, we made these assumptions:

- All the points of the detected human lie on a plane.
- The plane is parallel to the x-y axis of the camera, so $\hat{n}^\top = [0, 0, 1]$.
- The person's distance is in the range of 1 to 3 meters, and we transform the image based on the different expected distance ranges from LiDAR estimation. For instance, the H matrix for $h = 3$ is:

$$ {}^2H_1 = \begin{bmatrix} 0.896 & -0.219 & 13.76 \\ 0.000 & 0.683 & 16.55 \\ 0.000 & -0.002 & 1.000 \end{bmatrix} \tag{4.8}$$

## 4.3 Person detection and human pose estimation

YOLOv8-Pose [35] is used for person detection and pose estimation. YOLOv8 (You Only Look Once, Version 8) is the latest iteration of the YOLO series, known for its real-time object detection capabilities. YOLOv8 extends these capabilities to pose estimation, which involves detecting keypoints on human bodies to understand their posture and movements. YOLOv8-Pose identifies a person in an image and provides a confidence score by estimating a bounding box around them. The bounding box includes the top corner of the bounding box and its width ($w$) and height ($h$). Simultaneously, the YOLOv8-Pose also estimates the human pose. The human pose is represented as the 2D location of the 17 keypoints $(x_p, y_p)$ that correspond to different human body parts for example, $p \in \{\text{shoulder}, \text{foot}, \text{hip}, \dots\}$. YOLOv8-Pose models come in five sizes, from smallest to largest (n)ano, (s)mall, (m)edium, (l)arge, e(x)tra-large. Bigger models have more layers, thus more weights and thus more representation power, but at the cost of compute and memory resources.

### 4.3.1 YoloV8

YOLOv8 shares a similar backbone as YOLOv5 but includes changes to the cross-stage partial bottleneck(CSPLayer), now referred to as the CSP with two convolutions (C2f) module. This module combines high-level features with contextual information to enhance detection accuracy [36]. The Figure 4.1 shows YoloV8 Architecture [37]. It employs a modified CSPDarknet53 [38] backbone, a C2f module, an spatial pyramid pooling fast (SPPF) layer for fast computation, convolutions with batch normalization and SiLU activation, and a decoupled head for independent handling of objectness, classification, and regression tasks. The architecture of YOLOv8 can be divided into Backbone, Neck, and Head. The Backbone of YOLOv8 is designed to extract features from input images. It employs a series of convolutional layers, which include:

- **Convolutional Layers**: These layers perform convolutions to detect features at various levels of abstraction.
- **Batch Normalization**: This helps in stabilizing and speeding up the training process by normalizing the output of the convolutional layers.
- **Activation Functions**: Typically, ReLU or Leaky ReLU are used to introduce non-linearity.

The Backbone's role is to generate a rich feature map that represents the input image, capturing spatial hierarchies from low-level edges to high-level semantic content. The Neck

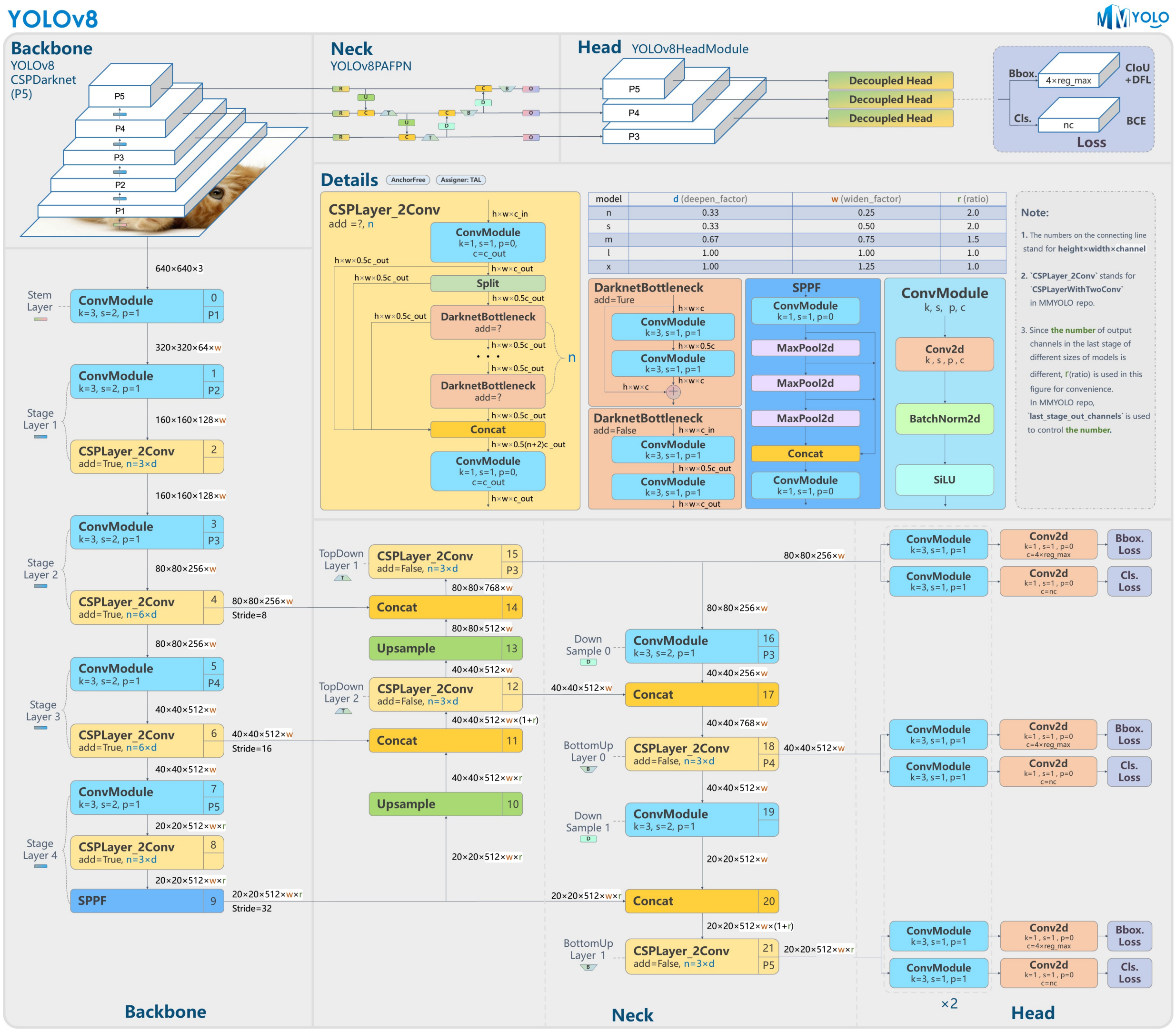


Figure 4.1: YoloV8 Architecture.

of YOLOv8 enhances the feature representation and prepares it for the detection task. It includes:

- **Feature Pyramid Networks (FPN)**: FPNs are used to combine features at different scales, which helps in detecting objects of various sizes.
- **Path Aggregation Network (PAN)**: PAN further processes the multi-scale features from FPN to improve the flow of information.

The Neck helps in aggregating features from different levels, making the detection process more robust to variations in object size and position.

The Head of YOLOv8 is where the actual detection and keypoint estimation occur. It consists of:

- **Bounding Box Prediction**: Predicts bounding boxes using anchor boxes and their respective offsets.
- **Objectness Score**: Indicates the likelihood of an object, in this case human being present in the bounding box.
- **Class Prediction**: Determines the class probabilities for each human.
- **Keypoint Estimation**: For pose estimation, YOLOv8 predicts the coordinates of keypoints on the human body.

The Head uses convolutional layers to refine the feature maps from the Neck and generate the final detection outputs.

### 4.3.2 Human Bounding Box Prediction and Keypoints Detection

YOLOv8 [36, 37] predicts bounding boxes (x, y, w, h) using anchor boxes, which are predefined boxes with specific aspect ratios and scales. The model predicts adjustments to these anchor boxes to fit the human more accurately. The objectness score represents the likelihood that a bounding box contains an object. YOLOv8 uses a sigmoid function to predict this score:

$$\text{objectness} = \sigma(o) \tag{4.9}$$

where $o$ is the raw score predicted by the network. Class probabilities are predicted using a softmax function over the raw class scores.

Keypoints are predicted for each detected person within the bounding boxes. The keypoints are represented as $(x_k, y_k)$ coordinates, where $k$ denotes a specific keypoint. YOLOv8 uses the following equation to predict keypoint coordinates:

$$(x_k, y_k) = \sigma(t_k) + (c_x, c_y) \tag{4.10}$$

where $t_k$ is the predicted offset for the keypoint, and $(c_x, c_y)$ are the cell coordinates.

## 4.4 Fall detection from human pose

The next step is detect fall from extracted poses from the previous step. Two separate approaches for fall detection modules were experimented: (a) Rules-based fall detection and (b) MLP-based fall detection. These two approaches are discussed below.

### 4.4.1 Rules-based Fall Detection

The rules-based approach for fall detection uses predefined conditions on the detected keypoints to identify whether a person has fallen. This method leverages the keypoints detected by YOLOv8-Pose to systematically examine specific conditions with high confidence. These conditions are designed to ensure that critical aspects of the body's posture are assessed accurately.

Using the keypoints detected by YOLOv8-Pose, a fall $F$ can be determined using the following criteria:

$$F = (y_{\text{shoulder}} > (y_{\text{foot}} - l)) \wedge (y_{\text{hip}} > (y_{\text{foot}} - l/2)) \wedge (y_{\text{shoulder}} > (y_{\text{hip}} - l/2)) \vee (h < w) \tag{4.11}$$

where $l = \|(x, y)_{\text{shoulder}} - (x, y)_{\text{hip}}\|_2$, and $h$ and $w$ are the height and width of the person's bounding box, respectively. The rules-based method is based on several geometric conditions involving the relative positions of the keypoints detected on the human body. It is to be noted that the (0, 0) pixel location lies in the top-left corner of the image meaning the object at the bottom will have greater y-coordinate than the object at top. The conditions can be elaborated as follows:

1. **Shoulder Below Waist**: The y-coordinate of the shoulder ($y_{\text{shoulder}}$) must be greater than the y-coordinate of the foot ($y_{\text{foot}}$) minus the length $l$ (distance between shoulder and hip). This condition ensures that the shoulder is positioned higher than the foot, indicating an upright position.

2. **Hip Below Foot**: The y-coordinate of the hip ($y_{\text{hip}}$) must be greater than the y-coordinate of the foot ($y_{\text{foot}}$) minus half the length $l$. This condition checks that the hip is higher than the foot.

3. **Shoulder Under Hip**: The y-coordinate of the shoulder ($y_{\text{shoulder}}$) must be greater than the y-coordinate of the hip ($y_{\text{hip}}$) minus half the length $l$. This condition ensures that the shoulder is above the hip.

4. **Width Greater than Height**: The height ($h$) of the bounding box should be smaller than width ($w$). If the height is less than the width, it is an indication that the person is lying down, suggesting a fall.

These conditions collectively help in identifying whether the detected person is in a standing or fallen position. The rules-based approach is straightforward and computationally efficient, making it suitable for real-time applications. However, its performance heavily depends on the accuracy of the detected keypoints. Misidentification or occlusion of keypoints can lead to incorrect fall detection.

### 4.4.2 MLP-based Fall Detection

In addition to the rules-based method, a fall detection module using a Multi-Layer Perceptron (MLP) was imployed. This approach leverages machine learning to classify whether a fall has occurred based on the detected keypoints. The Figure 4.2 illustrate the architecture of the MLP model used for fall detection. The MLP for fall detection is

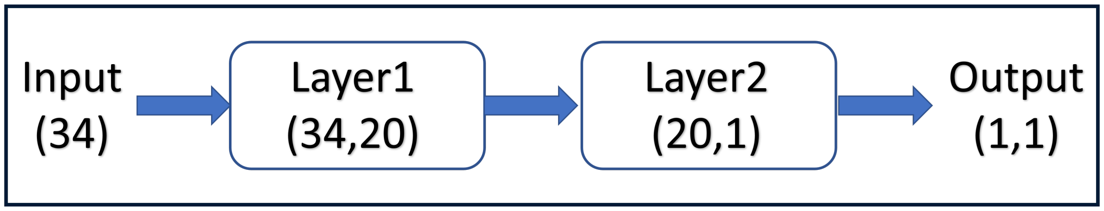


Figure 4.2: MLP Architecture

designed with the following architecture:

- **Input Layer**: 34 units, corresponding to the 2D coordinates of 17 keypoints.
- **Hidden Layers**: Two hidden layers, each with 20 units and ReLU activation functions.
- **Output Layer**: 2 units with softmax activation, representing the classes "fall" and "no fall".

The input to the MLP is the x and y coordinates of the keypoints. The output is a probability distribution over the two classes. The MLP is trained using the FallDetectionDatabase [39] containing annotated instances of falls and non-falls. The training process involves the following steps:

1. **Data Collection**: Collecting a dataset with keypoints annotated for fall and non-fall scenarios.
2. **Preprocessing**: Normalizing the keypoints coordinates to ensure consistency in the input data.
3. **Training**: Using the annotated dataset to train the MLP with binary cross-entropy loss. The training and validation accuracy are shown in Figure 4.3.
4. **Evaluation**: Evaluating the trained model on a separate test set to assess its accuracy and performance.

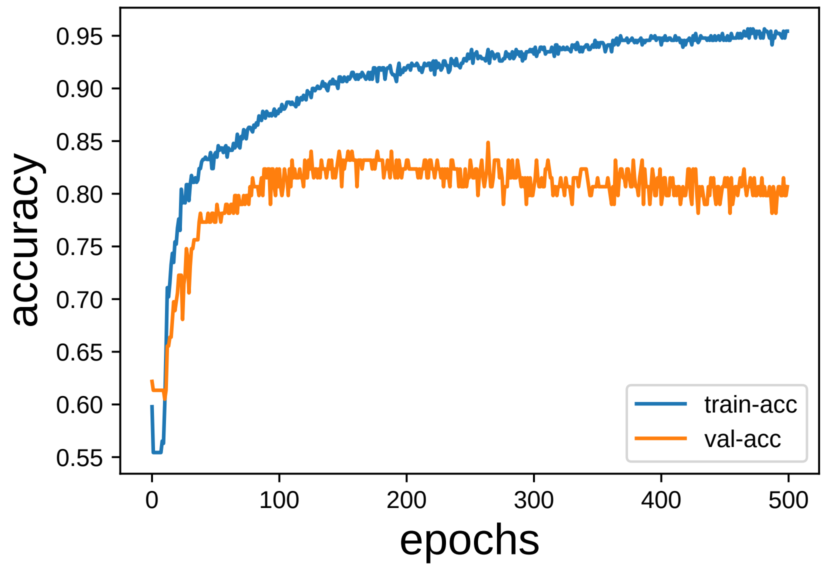


Figure 4.3: MLP accuracy on Fall Detection Dataset

The MLP-based approach can capture more complex patterns in the keypoints data, potentially leading to higher accuracy in fall detection. However, it requires a sufficiently large and diverse dataset for training to generalize well to new data.

## 4.5 Experiment and Results

For human fall dataset, 128 images from robot's POV are captured for the custom dataset. Then dataset is manually categorized into 62 images depicting human falls and 66 images depicting not-falls under similar lighting conditions. This dataset was used only for evaluating the methods, not training, and is available on the project GitHub page.

### 4.5.1 Evaluating the fall detection pipeline

The fall detection pipeline contains of 3 steps, (a) pre-processing (b) person detection and human pose estimation and (c) fall detection from human pose. Two options for the pre-processing step were evaluated: (i) *robot POV*, where no homography is applied to the input image (ii) *human POV* where homographies are computed based on the minimum and maximum distance in the room, applied to the input image and highest confidence results are picked. Five options for the person detection step: n, s, m, l, x for each size of the YOLOv8-Pose model from the smallest to the largest were evaluted. Lastly, two options for the fall detection from human pose were evaluated: (i) Rules-based fall detection and (ii) MLP-based fall detection. All model variations combined leads to 20 models. The accuracy of these models on the dataset is plotted in Figure 4.5 (Right). On average, across the five

Table 4.1: Fall detection module performance on the created dataset

| | YOLOv8 (orig) | | | | | YOLOv8 (custom) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| metrics | n | s | m | l | x | n | s | m | l | x |
| Accuracy | 0.49 | 0.54 | 0.61 | 0.62 | 0.70 | 0.55 | 0.60 | 0.66 | 0.74 | 0.70 |
| Precision | 0.40 | 0.59 | 0.67 | 0.67 | 0.79 | 0.63 | 0.70 | 0.76 | 0.85 | 0.80 |
| Recall | 0.10 | 0.16 | 0.39 | 0.42 | 0.53 | 0.19 | 0.31 | 0.45 | 0.56 | 0.52 |
| F1-Score | 0.16 | 0.25 | 0.49 | 0.51 | 0.63 | 0.30 | 0.43 | 0.57 | 0.68 | 0.63 |

models, the homography improved the accuracy by 6-12 percent. It is observed that the human POV improves accuracy over robot's POV consistently as shown in Table 4.1. Also Rules-based model has a better accuracy than the MLP-based model meaning that MLP model must be improved in order to increase the accuracy.

### 4.5.2 Robustness to homography

The results from previous section show that changing robot POV to human POV using homography transform improves accuracy. This suggests that YOLOv8 trained on MS-COCO is not robust to homographic transformations of the original image. The performance of the pre-trained YOLOv8n [35] model on homographic-variations of MS-COCO validation dataset are evaluted. To compute these homographies, the angle of view are varied in the pitch direction (downwards) ($\theta \in \{0°, 5° \ldots, 55°\}$) from the original human viewpoint $\theta = 0$ in the MS-COCO dataset. A fixed distance of 3m from the person to the camera is assumed. Then the MS-COCO validation images are transformed using the computed homographies. These

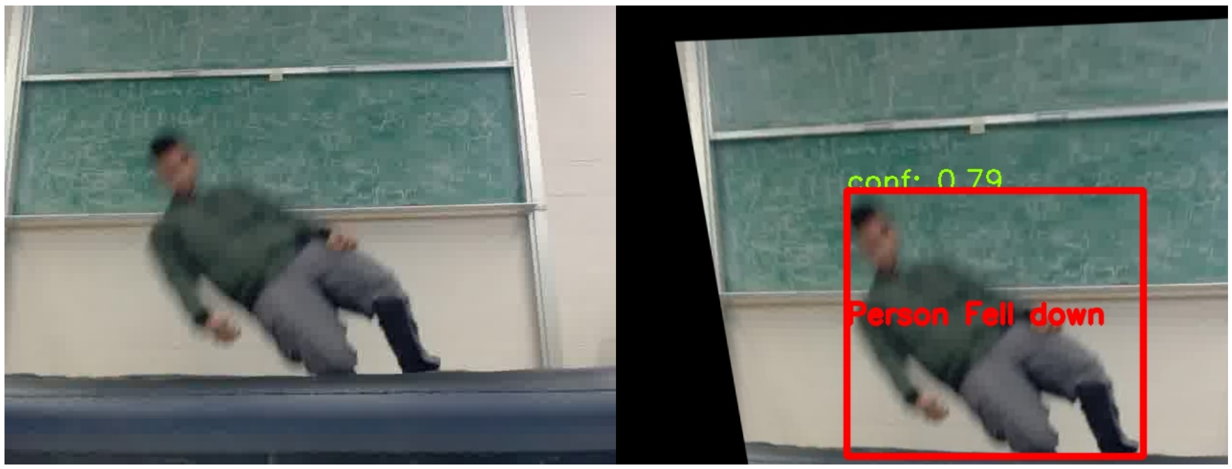


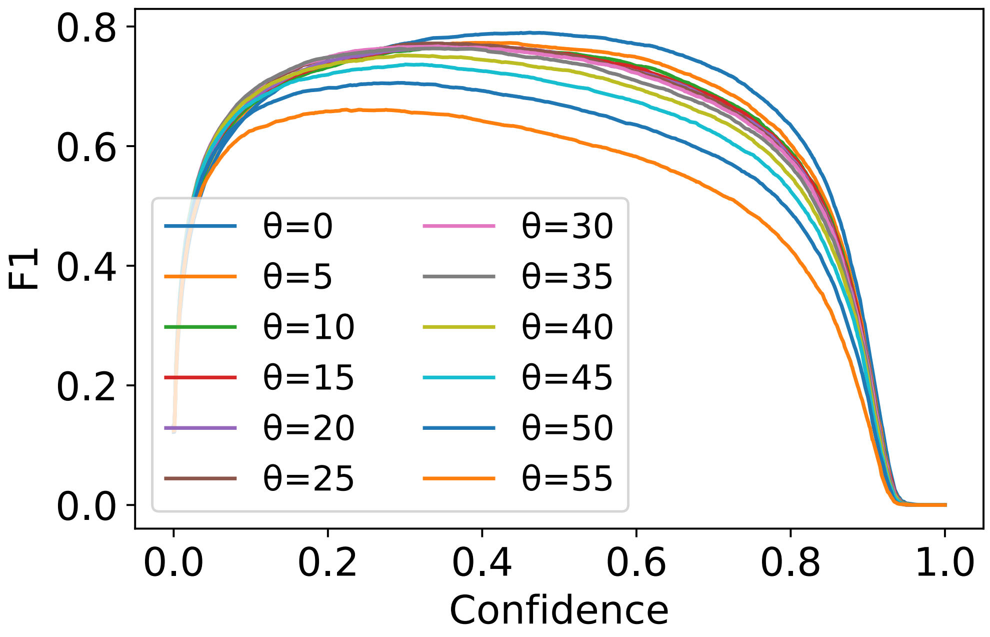
Figure 4.4: Applying Homography on a sample image in the data set. **Left**: image taken from the robot's POV. **Right**: transformed image to a human POV.

images are fed into YOLOv8-Pose-nano for pose estimation. The F1-score (harmonic mean of precision and recall) under different confidence thresholds of and a fixed-distance threshold for pose estimation are then calculated. The resulting plot is shown in Figure 4.5 (Top). The results show that homography-corruption of the MS-COCO dataset reduces the F1 score of pose estimation at all confidence levels. The mean F1 score of pose estimation drops from 0.789 to 0.745 and a drop in the mean average precision score (mAP50) for the predicted bounding box from 0.911 (original dataset) to 0.638 ($\theta = 30°$). The **Top** figure shows the F1 Score of Pose Estimation of MS-COCO validation dataset using YOLOv8-Pose-nano. The **Botton** figure illustrates the Comparison of fall detection accuracy from robot POV vs human POV. By transforming images from the robot POV to the human POV through

Table 4.2: YOLOv8n accuracy reduces when images are transformed to robot POV

| dataset | mAP50 | mAP50-95 |
|---|---|---|
| MS-COCO validation (original) | 0.798 | 0.511 |
| MS-COCO validation (transformed to robot POV) | 0.274 | 0.118 |

the Homography matrix, fall detection performance is improved by 6-12% across models. The x-axis indicates the 5 model sizes (n)nano, (s)mall, (m)edium, (l)arge, e(x)tra-large of YOLOv8-Pose as shown in Table 4.2.



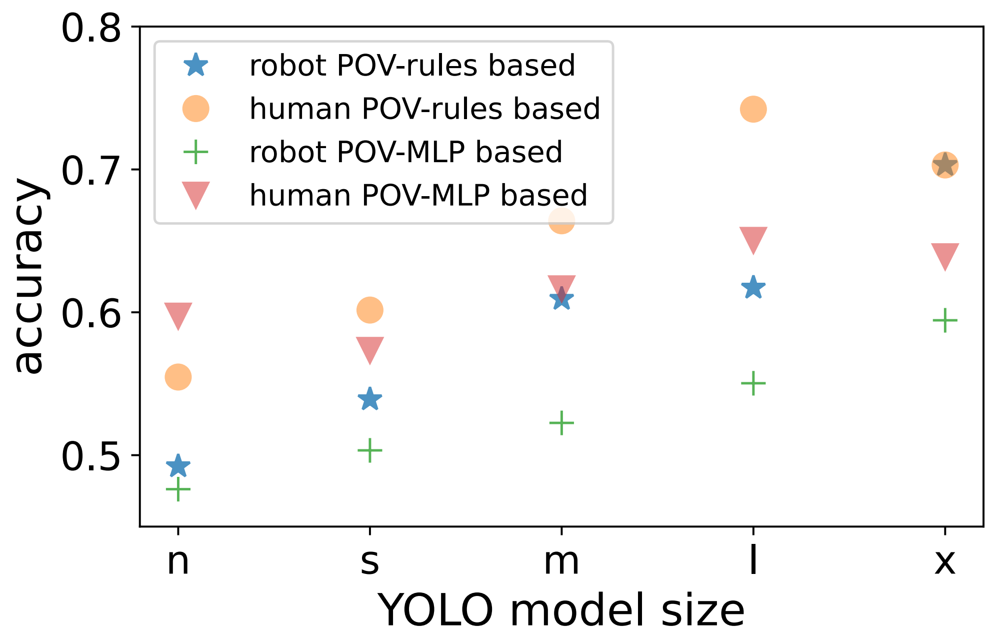
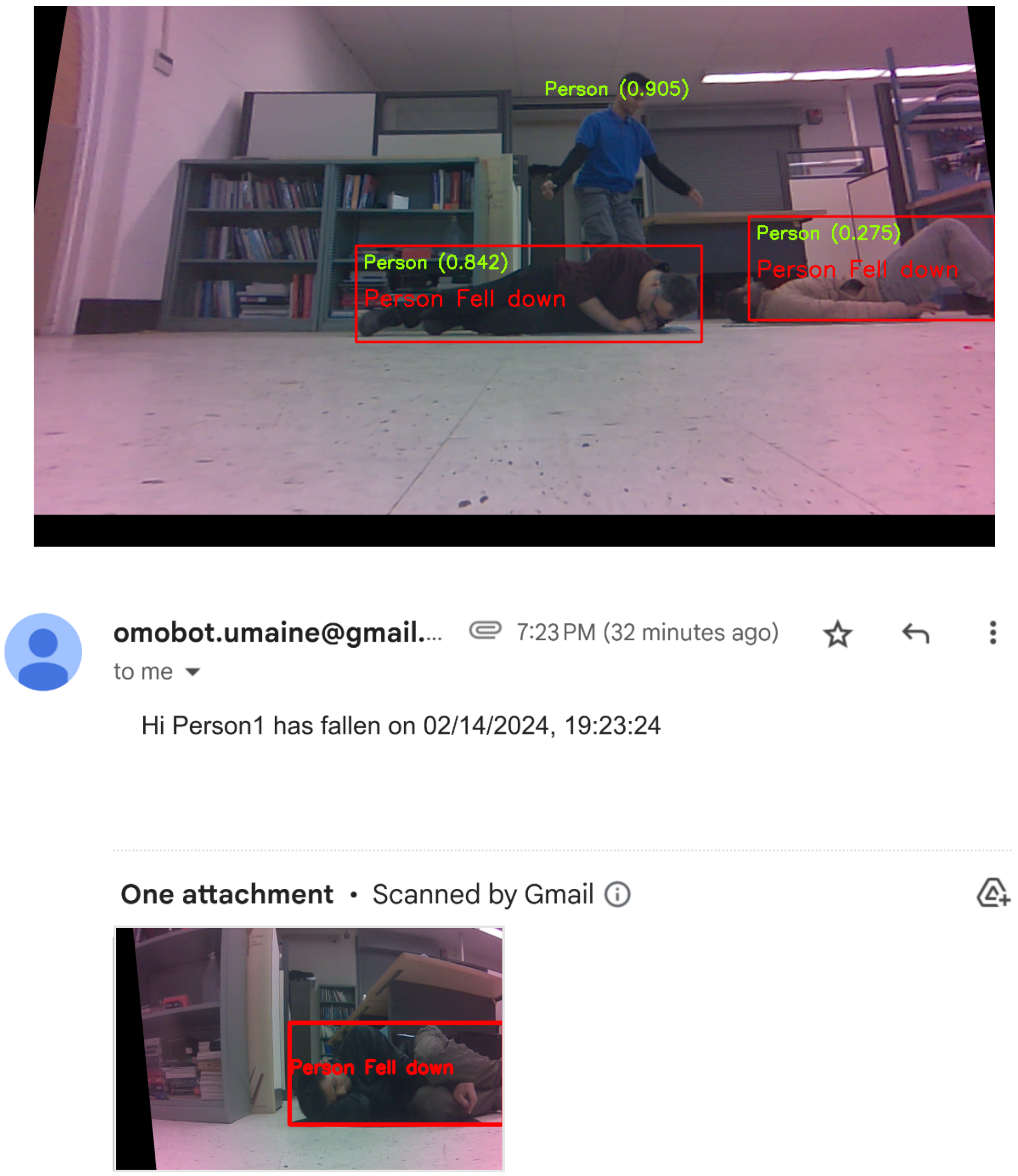


Figure 4.5: F1 Score of Pose Estimation and Comparison of fall detection accuracy.

### 4.5.3 Qualitative results

Figure 4.6 shows the robot detecting multiple fallen people and sending an email alert with the image as an attachment. In this experiment, multiple individuals are within the



omobot.umaine@gmail.... 7:23 PM (32 minutes ago)
to me

Hi Person1 has fallen on 02/14/2024, 19:23:24

One attachment • Scanned by Gmail

Figure 4.6: **Top**: Fall event detection in the presence of multiple people. **Bottom**: Fall detected by the robot, and sample email generated by the system.

view of the robot, and the fall detection module detects and reports the detected falls, accompanied by labeled images are presented.

### 4.5.4 Summary

Both the rules-based and MLP-based approaches for fall detection have their strengths and limitations. The rules-based method is simple and efficient but relies on the accurate detection of keypoints. The MLP-based method, on the other hand, leverages machine learning to capture complex patterns but requires extensive training data. Future work may involve combining these approaches to further enhance the robustness and accuracy of fall detection systems. **The overall demo** of the entire system working together is shown as a video here[1] .

[1]https://youtu.be/wcP0rxez69o

# CHAPTER 5
# CONCLUSION AND FUTURE WORK

## 5.1 Conclusion

The thesis presents Omobot, a mobile robot designed with an emphasis on affordability and autonomy to enhance elderly care by detecting falls. The practical effectiveness and reliability of the robot in real-world scenarios have been demonstrated through a variety of experiments, including motor system identification, payload capacity, battery runtime, odometry accuracy, and obstacle avoidance efficiency. Motor System Identification showed that with a simple motor system identification, the robot could maintain a more accurate trajectory, reducing deviations from planned paths. Physical tests demonstrated that the robot could carry a significant load, providing the potential to include physical attachments that can improve the safety further. Also, the battery offered sufficient runtime for practical applications, lasting an average of 93 minutes on a full load. This ensures that the robot can operate effectively without frequent recharges. The robot effectively detected and maneuvered around obstacles, which is an essential requirement for ensuring safe interaction in environments inhabited by elderly people or those requiring assistance. The robot has also demonstrated reliable fall detection capability through improved image processing techniques and the usage of advanced pose detection algorithms.

## 5.2 Future Work

Omobot has demonstrated the potential to be applied in the real world to ensure the security and safety of elderly people. However, before deploying to a real-world environment, some improvements must be made to increase its adaptability and ensure its effectiveness in a broader range of environments. Some of the future works are summarized in the following subsections:

### 5.2.1 Enhanced Sensor Integration

One of the primary recommendations for future work is the integration of more advanced sensing technologies. The current model utilizes 2D LiDAR, which, while effective for primary navigation and obstacle detection, has limitations in complex environments with varying heights and more intricate obstacle configurations. Full utilization of the mounted stereo cameras would perceive surroundings in 3D, significantly improving its ability to navigate more complex environments.

### 5.2.2 Adaptive Homography Transformations

Currently, the robot uses fixed homography transformations to adjust its perception based on the camera's viewpoint (Assumed to be 1.5 m). Future research could explore adaptive homography techniques that dynamically adapt to different environmental setups and camera perspectives. This would not only improve the accuracy of the robot's pose estimation and movement analysis but also enhance its versatility.

### 5.2.3 Fall Detection Model Enhancements

To improve the reliability and accuracy of fall detection, it is proposed that deep learning models used by Omobot be developed further and refined. Currently, the MLP-based model uses two hidden layers, making the model deeper. Training on a more diverse dataset that includes a broader range of fall types, environmental conditions, and user interactions could help better understand and predict fall incidents. This could solve the potential problem of failure to detect falls with the partial presence of humans hidden behind obstacles. This enhancement could lead to quicker and more accurate responses to falls and allow the robot to adapt to the behaviors and patterns of its users.

### 5.2.4 Multi-Sensor Data Fusion

Integrating data from various sensors to represent the environment better with more information can be a future direction. By combining inputs from visual sensors, LiDAR, and perhaps even auditory sensors, Omobot could achieve a better perception of its surroundings. This sensor fusion would be instrumental in developing more sophisticated decision-making algorithms that can differentiate between normal activities and potential hazards or emergencies.

### 5.2.5 User Interaction and Interface Enhancements

Making the user interface more intuitive and accessible for elderly users will be more helpful. Future iterations could include more natural language processing capabilities, allowing users to communicate with Omobot using voice commands more conversationally. Enhancements might also include better physical interaction designs, such as more accessible emergency buttons or user-friendly touch interfaces.

### 5.2.6 Real-World Testing and Customization

Real-world testing is critical to understanding how Omobot performs outside of controlled environments. Future work should include pilot programs in actual home settings, with a focus on collecting feedback from elderly users and caregivers to tailor the robot's features and functionalities to meet their needs better.

Through these extensive future developments, Omobot could significantly advance in its role as an assistive technology for the elderly, providing not only more reliable fall detection but also enhanced daily support.

# BIBLIOGRAPHY

[1] "Elderly population (indicator)," OECD (2024) (Accessed on 05 Feb 2024). [Online]. Available: http://dx.doi.org/10.1787/8d805ea1-en

[2] H. Peeters, A. Debels, and R. Verpoorten, "Excluding institutionalized elderly from surveys: Consequences for income and poverty statistics," https://doi.org/10.1007/s11205-011-9957-8, p. 751–769, 2013.

[3] M. Tinetti, "Prevention of falls and fall injuries in elderly persons: A research agenda," *Preventive Medicine*, vol. 23, no. 5, pp. 756–762, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0091743584711303

[4] K. Doughty, K. Cameron, and P. Garner, "Three generations of telecare of the elderly," *Journal of Telemedicine and Telecare*, vol. 2, no. 2, pp. 71–80, 1996.

[5] S. Chaudhuri, H. Thompson, and G. Demiris, "Fall detection devices and their use with older adults: a systematic review," *Journal of geriatric physical therapy (2001)*, vol. 37, no. 4, p. 178, 2014.

[6] P. Rajendran, A. Corcoran, B. Kinosian, and M. Alwan, *Falls, Fall Prevention, and Fall Detection Technologies.* Totowa, NJ: Humana Press, 2008, pp. 187–202. [Online]. Available: https://doi.org/10.1007/978-1-59745-233-5_8

[7] X. Wang, J. Ellul, and G. Azzopardi, "Elderly fall detection systems: A literature survey," *Frontiers in Robotics and AI*, vol. 7, 2020. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2020.00071

[8] Z. Zhang, C. Conly, and V. Athitsos, "A survey on vision-based fall detection," in *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '15. New York, NY, USA: Association for

Computing Machinery, 2015. [Online]. Available: https://doi-org.wv-o-ursus-proxy02.ursus.maine.edu/10.1145/2769493.2769540

[9] L. Ciabattoni, F. Ferracuti, G. Foresi, A. Freddi, A. Monteriù, and D. P. Pagnotta, "Real-time fall detection system by using mobile robots in smart homes," in *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2017, pp. 15–16.

[10] Z. A. Mundher and J. Zhong, "A real-time fall detection system in elderly care using mobile robot and kinect sensor," *International Journal of Materials, Mechanics and Manufacturing*, vol. 2, no. 2, pp. 133–138, 2014.

[11] W. H. Chin, N. Nuo Wi Tay, N. Kubota, and C. K. Loo, "A lightweight neural-net with assistive mobile robot for human fall detection system," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–6.

[12] C. Menacho and J. Ordoñez, "Fall detection based on cnn models implemented on a mobile robot," in *2020 17th International Conference on Ubiquitous Robots (UR)*, 2020, pp. 284–289.

[13] F. M. Lopez-Rodriguez and F. Cuesta, "An android and arduino based low-cost educational robot with applied intelligent control and machine learning," *Applied Sciences*, vol. 11, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/1/48

[14] A. A. S. Gunawan, B. Clemons, I. F. Halim, K. Anderson, and M. P. Adianti, "Development of e-butler: Introduction of robot system in hospitality with mobile application," *Procedia Computer Science*, vol. 216, pp. 67–76, 2023, 7th International Conference on Computer Science and Computational Intelligence 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922021901

[15] P. He, H. Lv, H. Wu, and G. Yang, "Modeling and control of differential-drive chassis for a homecare assistive robot," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, 2023, pp. 1–4.

[16] X. Tan, S. Zhang, and Q. Wu, "Research on omnidirectional indoor mobile robot system based on multi-sensor fusion," in *2021 5th International Conference on Vision, Image and Signal Processing (ICVISP)*, 2021, pp. 111–117.

[17] G. Chen and X. Duan, "Vision-based elderly fall detection algorithm for mobile robot," in *2021 IEEE 4th International Conference on Electronics Technology (ICET)*, 2021, pp. 1197–1202.

[18] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, "Yolo nano: A highly compact you only look once convolutional neural network for object detection," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 22–25.

[19] E. Matsinos, "Modelling of the motion of a mecanum-wheeled vehicle," 2012. [Online]. Available: https://arxiv.org/abs/1211.2323

[20] M. Corina, E. Stoica, C. Bortun, M.-L. Negrutiu, C. Sinescu, and A. Tudor, "Advantages of a polyethylene terephthalate glycol-modified coated with a thermoplastic polyurethane as an occlusal appliance material," *Revista de Chimie*, vol. 65, pp. 734–736, 06 2014.

[21] Nvidia, "NVIDIA Jetson Nano — nvidia.com," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/, [Accessed 20-07-2024].

[22] K. Piemngam, I. Nilkhamhang, and P. Bunnun, “Development of autonomous mobile robot platform with mecanum wheels,” in *2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, 2019, pp. 90–93.

[23] Nvidia, “JetPack SDK 4.6.1 — developer.nvidia.com,” https://developer.nvidia.com/embedded/jetpack-sdk-461, [Accessed 20-07-2024].

[24] B. Joshi, R. Shrestha, and R. Chaudhar, “Modeling, simulation and implementation of brushed dc motor speed control using optical incremental encoder feedback,” in *Proceedings of IOE graduate conference*, 2014, pp. 497–505.

[25] S. Gatesichapakorn, J. Takamatsu, and M. Ruchanurucks, “Ros based autonomous mobile robot navigation using 2d lidar and rgb-d camera,” in *2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, 2019, pp. 151–154.

[26] “GitHub - TemugeB/python_stereo_camera_calibrate: Stereo camera calibration with python and openCV — github.com,” https://github.com/TemugeB/python_stereo_camera_calibrate, [Accessed 20-07-2024].

[27] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen, “On benchmarking camera calibration and multi-view stereo for high resolution imagery,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[28] Z. M. Salameh and B. G. Kim, “Advanced lithium polymer batteries,” in *2009 IEEE Power & Energy Society General Meeting*, 2009, pp. 1–5.

[29] C. Dosoftei, V. Horga, I. Doroftei, T. Popovici, and S. Custura, “Simplified mecanum wheel modelling using a reduced omni wheel model for dynamic simulation of an

omnidirectional mobile robot," in *2020 International Conference and Exposition on Electrical And Power Engineering (EPE)*, 2020, pp. 721–726.

[30] A. Abbas, p. peerzada, and w. larik, "Dc motor speed control through arduino and l298n motor driver using pid controller," 07 2022.

[31] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[33] K. Wang, B. Fang, J. Qian, S. Yang, X. Zhou, and J. Zhou, "Perspective transformation data augmentation for object detection," *IEEE Access*, vol. 8, pp. 4935–4943, 2019.

[34] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[35] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[36] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83

[37] Open-Mmlab, "Mmyolo/configs/yolov8 at main · open-mmlab/mmyolo." [Online]. Available: https://github.com/open-mmlab/mmyolo/tree/main/configs/yolov8

[38] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[39] U. K. Kandagatla, "Fall detection dataset," Available at https://www.kaggle.com/datasets/uttejkumarkandagatla/fall-detection-dataset (05/10/2024), 2022.

# BIOGRAPHY OF THE AUTHOR

Shihab Uddin Ahamad is a candidate for the Master's of Science in Computer Engineering degree in Department of Electrical and Computer Engineering from the University of Maine in 07/29/2024. He earned his B.Sc. in Electrical, Electronic and Communication Engineering from Military Institution of Science and Technology, Dhaka, Bangladesh, in January 2020. After working as an Assistant Engineer in an IC design company, he began his graduate studies in the Electrical and Computer Engineering department at the University of Maine. Shihab Uddin Ahamad is a candidate for the Master's of Science degree in Computer Engineering from the University of Maine in August 2024.