Tribhuvan University
Institute of Engineering
Pulchowk Campus

B-05-BME-2019-2024
Machine Learning in Robotics, with it’s demonstration in Inventory transportation

By:
Bidhek khatiwada (076bme005)
Nabin Basnet (076bme022)
Ramu Kandel (076bme031)
Saroj Belbase (076bme040)

A Project Report
Submitted to the Department of Mechanical And Aerospace Engineering in Partial Fulfillment of the Requirement For The Bachelor’s Degree in Mechanical Engineering

Department of Mechanical and Aerospace Engineering
Lalitpur, Nepal

April, 2024

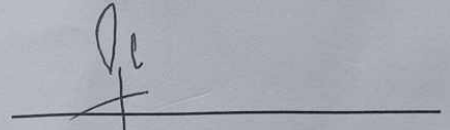
Copyright

The authors have agreed that the library, Department of Mechanical and Aerospace Engineering, Pulchowk Campus, Institute of Engineering may make this thesis freely available for inspection. Moreover, the authors have agreed that permission for extensive copying of this thesis for scholarly purpose may be granted by the professor(s) who supervised the work recorded herein or, in their absence, by the Head of the Department wherein the thesis was done. It is understood that the recognition will be given to the author of this thesis and to the Department of Mechanical and Aerospace Engineering, Pulchowk Campus, Institute of Engineering in any use of the material of this thesis. Copying or publication or the other use of this thesis for financial gain without approval of the Department of Mechanical and Aerospace Engineering, Pulchowk Campus, Institute of Engineering and authors' written permission is prohibited. Request for permission to copy or to make any other use of the material in this thesis in whole or in part should be addressed to:

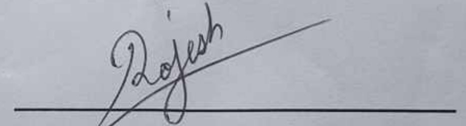
Head of Department
Department of Mechanical and Aerospace Engineering
Pulchowk Campus, Institute of Engineering
Lalitpur, Nepal

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

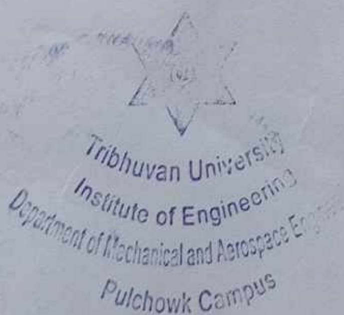
The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled “Machine Learning in Robotics with it’s demonstration in inventory transportation” submitted by Saroj Belbase, Nabin Basnet, Bidhek Khatiwada and Ramu kandel in partial fulfillment of the requirements for the degree of Bachelor in Mechanical Engineering.

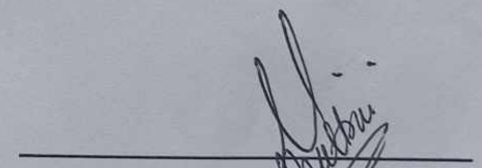


Superviser, **Prof. Dr. Laxman Poudel**
Professor
Department of Mechanical and Aerospace Engineering



External Examineer, **Er. Rojesh Man Shikhrakar**
Sr. Manager
Fusemachines Nepal





Committee chairperson, **Dr. Sudip Bhattarai**
Head of Department
Department of Mechanical and Aerospace engineering

Date: 2024-04-10

Abstract

The advancement of Industry 4.0 has led to the design of various control methods for the autonomous navigation of robots. While many works rely on Simultaneous Localization and Mapping (SLAM) or path planning systems for trajectory tracking, there are limitations when real-time obstacle avoidance and parameter reconfiguration are required. However, with the recent advancements in machine learning algorithms, new possibilities have emerged across various fields, including autonomous navigation systems. This has led to significant developments in the field, paving the way for innovative approaches and solutions. Our project thus, aims bring that development up and diversify it, by joining autonomous navigation system with other tasks. For this final year college project, we have decided to bring autonomous navigation system with material handling system. We have decided to make a forklift and have a demonstration of it working is a miniaturize factory warehouse. For the environment we have choose a place with boundaries and have workload. These workloads are to be carried to a designated place as marked by the computer program. For this project we have decide to use a camera only. Here in our project we have designed our own ML model along with other subsidiary model and will train them as well to complete our stated task.

Acknowledgement

First and foremost, we would like to express our gratitude to the Supreme Being, without whose blessings this thesis would not have been possible. Furthermore, we would also like to acknowledge the invaluable guidance and support provided by our supervisor, Associate Professor Dr. Laxman Poudel, from the Department of Mechanical and Aerospace Engineering at IOE, Pulchowk Campus.

We are also grateful to Dr. Sudip Bhattarai, Phd, HOD from the Department of Mechanical and Aerospace Engineering at IOE, Pulchowk Campus, who has provided us with his expertise and assistance throughout the project. We also extend our heartfelt appreciation to each and every individual who has contributed to the completion of this thesis. Lastly, we would like to express our gratitude to the Department of Mechanical and Aerospace Engineering at IOE, Pulchowk Campus, Lalitpur-3, Nepal, for providing us with the necessary resources and facilities to undertake this research.

Contents

Copyright i

Abstract iii

Acknowledgement iv

List of Figures vii

List of Abbreviation ix

1 INTRODUCTION 1

1.1 Background 1

1.1.1 Supervised Learning 2

1.1.2 Loss function 2

1.1.3 Model Training 3

1.1.4 Neural Networks 3

1.1.5 Deep Learning Models 4

1.1.6 Convolutional Neural Networks 5

1.1.7 Activation function 8

1.1.8 Computer Vision 10

1.1.9 Warehousing and distribution 11

1.2 Problem statement 11

1.3 Objectives 12

1.3.1 Main objective 12

1.3.2 Specific objective 12

1.4 Limitations 12

2 LITERATURE REVIEW 13

2.1 Autonomous navigation 13

2.1.1 Traditional system 13

2.1.2 Different ML model system deployed for navigation . . . 15

2.2 Machine Learning approaches for autonomous navigation 16

2.2.1 Vision based technology for mobile robots 17
2.2.2 Deep learning for navigation 19
2.2.3 Vision based approaches 19

3 METHODOLOGY 22
3.1 Design and fabrication phase 22
3.1.1 Differential Drive Control 23
3.1.2 Stepper Motor Control 25
3.1.3 Mechanical Calculation 28
3.1.4 3D Robot Model 33
3.1.5 Fabricated Model 33
3.2 Pytorch-based architecture 34
3.2.1 Dataset Preparation 35
3.2.2 Model Architecture 37
3.2.3 Optimization 41
3.2.4 Training 41
3.3 CoppeliaSim: A robot simulator 41
3.3.1 Joints 43
3.3.2 Sensor 44
3.4 Object detection model 50
3.4.1 How the navigation works 53

4 Results and Discussion 54
4.1 CoppeliaSim simulation 54
4.1.1 Training loss vs Epoch 54
4.1.2 Validation Loss vs Epoch 55
4.1.3 Accuracy of model vs Epoch 56
4.1.4 Navigation in CoppeliaSim environment 57
4.2 Physical Model 58
4.2.1 Confusion matrix for object detection model 58
4.2.2 Height loading model 59
4.2.3 Actual robot working 61

5 Conclusion 63

6 Limitations and future enhancement 65

References 67

Plagiarism Check Report 70

List of Figures

1.1 Conventional Robot Control Pipeline(Levine,2021) 5

2.1 Conventional mobile robot navigation framework(Zhu, K. & Zhang, T., 2021) 13
2.2 Macine learning navigation approach(on-line) 14
2.3 The learning process of robot based on RL(Ameer Hamza,2022) . 16
2.4 Monocular vision (A. Ohya et al.,2001) 18

3.1 Process flowchart 22
3.2 Parameters of mobile robot 23
3.3 H-Bridge Circuit 24
3.4 Direction Control(a) 24
3.5 Direction control(b) 25
3.6 Braking 25
3.7 Single phase stepping 26
3.8 Dual phase stepping 27
3.9 Half Stepping 27
3.10 Torque speed characteristics(arduino forum) 28
3.11 Power transmission and rotary positioning(Gates Mectrol, April 2001) 29
3.12 Vertical linear positioner(Gates Mectrol, April 2001) 31
3.13 Robot Model (Top View, Isometric View, Front View, Side View) 33
3.14 Top view of Fabricated robot for our project 34
3.15 Neural Network architecture 39
3.16 Rectified Linear Units(ReLU) 40
3.17 Object Properties 42
3.18 Object responses 42
3.19 Serial joint hierarchy in CoppeliaSim 43
3.20 Multi joint hierarchy in CoppeliaSim 43
3.21 Joint type in CoppeliaSim 44
3.22 Proximity sensor in CoppeliaSim 44
3.23 Vision sensor in CoppeliaSim 45

3.24 Object dection model planning 50
3.25 Object detection Network Architecture(R. Joseph, et al) 51

4.1 Training loss vs Epoch 55
4.2 Validation loss vs Epoch 56
4.3 Accuracy vs Epoch 57
4.4 Navigation in CoppeliaSim environment 58
4.5 Confusion matrix 59
4.6 Training loss vs Epoch (Height loading model) 60
4.7 Validation vs Epoch (Height loading model) 60
4.8 Accuracy vs Epoch (Height loading model) 61
4.9 Actual robot working 62

List of Abbreviation

DRL: Deep Reinforcement Learning

NN: Neural Networks

CNN: Convolution Neural Networks

SIANN: Space Invariant Artificial Neural Networks

ReLU: Rectified Linear Unit

GELU : Gaussian Error Linear Unit

BERT: Bidirectional Encoder Representation from transformers

SLAM: Simultaneous Localization and Mapping

LIDAR: Light Detection and Ranging

RL: Renforcement Learning

CCDarray: Charged Coupled Device array

MOSFET: Metal-Oxide Semiconductor Field Effect Transister

PWM: Pulse Width Modulation

FET: Field Effect Transister

PLA: Polylactic Acid

API: Application Programming Interface

YOLO: You Only Look Once

ASIC: Application-specific Integrated Circuit

SSH: Secure Shell

GPU: Graphics Processing Unit

Chapter 1

INTRODUCTION

1.1 Background

Machine learning is one of the fastest growing disciplines in engineering and technology, and has taken world by storm, we are seeing it adaptation in majority of field in the science and technologies as well as in other field that requires for problem solving. The transformative potential of machine learning rivals the game-changing impact the internet had in the late 20th and early 21st centuries.

Machine learning can be defined as a wide range of algorithms and techniques that allow computers to detect pattern in the data provides as inputs to the model and learn through those patters to make accurate prediction and through the application of those predictions make decisions without needing to be explicitly programmed. Talking about robotics now, we are more than aware that the today's robots are capable of delivering more to us in the field of general robotics, then they are actually delivering us, this is due to the fact that we have not been able to program them effectively enough for this purpose, and its simply because it is very hard to program them to do it, even the simple task that we take for granted is actually very hard to program for robots. This is where machine learning can help us with this problem by getting a software update in the field of robotics. Not only will the machine learning enable robots to perform complex tasks, but also enabling it to interact with the environment and solve problem that required at least a human level intelligence that include perception, learning, reasoning, and decision-making. This will allow robots to become more versatile, flexible, accurate, precise and efficient in carrying out various tasks, ranging from industrial automation to healthcare, transportation to scientific exploration, and even in domestic applications. Through machine learning techniques, robots can acquire new skills and knowledge, enabling them to handle complex situations and tasks that were previously difficult or impossible for them to accomplish. This learning abil-

ity also allows robots to adapt to different environments and perform tasks with higher precision and efficiency. Furthermore, it empowers robots to interpret and comprehend their surroundings utilizing a multitude of sensors including cameras, depth sensors, and touch sensors. These sensors furnish data that can be analyzed using AI techniques, enabling robots to identify objects, traverse through different environments, and engage with humans and other robots in a manner that is more instinctive and seamless. The integration of Machine learning in robotics will lead to significant advancements in the development of autonomous robots, capable of operating without constant human supervision. These robots can perform tasks in complex and dynamic environments, making them valuable in fields such as manufacturing, logistics, and exploration. They can also assist in healthcare settings, providing support to doctors and nurses, and even in everyday life as personal assistants or companions.

1.1.1 Supervised Learning

When a collection of n number of data points are fed to a model with each inputs mapped to an known output as follows:

$$\{(x_1, y_1), ..., (x_n, y_n)\},$$

where x_i is an input object and y_i is an known output for each x, the supervised learning is a technique to find a function $y = f(x)$ that makes use of these known data points and determines a relationship among these variables which can be used to predict or guess outputs y for new sets of inputs x not included in those data points previously fed into the model.

In robotics, supervised learning, including regression and classification tasks, is prevalent. In regression, the output is continuous, while in classification, it's categorical. This is evident in robotics, where imitation learning, a form of supervised learning, is used to emulate expert behavior based on input states and desired actions. Classification is also prominent in robotic computer vision, aiding in distinguishing between various objects or classes within images.

1.1.2 Loss function

In supervised learning scenarios, a metric referred to as a loss function is employed to assess and contrast potential models $f(x)$ aimed at accurately representing the data. Several loss functions are available for supervised learning tasks, with some of the most prevalent being the l_2 and l_1 loss functions for regression, and the 0-1 loss and cross-entropy loss functions for classification.

1. The l_2 loss function is defined by:

$$L = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \tag{1.1}$$

   In this loss function, the sum extends over a set of data points (x_i, y_i). It's apparent that the penalty stems from the function f not precisely aligning with the data at the sampled points. Notably, the penalty is quadratic in relation to this disparity. Consequently, this loss function tends to prioritize smaller residuals over larger ones, leading to enhanced overall model performance. However, this also renders the $l2$ loss susceptible to outliers in the data, diminishing the robustness of the training process.

2. The l_1 loss function is defined by:

$$L = \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i| \tag{1.2}$$

   In contrast to the l_2 loss, this particular loss function penalizes solely the absolute value of the residual. Consequently, it treats all residuals with more uniformity, fostering a training process that is generally more robust and less susceptible to outliers in the data.

1.1.3 Model Training

In supervised learning tasks where a predefined parametric model, like a linear model or neural network, is employed, the parameter values can be adjusted to achieve the best fit to the data, i.e., minimize the designated loss function. This adjustment process, known as model training, involves optimizing the parameters. While in certain instances, the optimal parameter set can be determined analytically, it's more typical to iteratively search for a favorable parameter set using numerical optimization methods.

1.1.4 Neural Networks

A widely utilized parametric model in machine learning is the neural network, also referred to as the multi-layer perceptron. Neural networks possess distinctive architectures comprising hierarchical layers of linear and nonlinear functions, endowing them with formidable function approximation capabilities. Mathematically, neural

networks are often represented as a sequence of functions:

$$\begin{aligned} g_1 &= f_1(w_1 x + b_1) \\ g_2 &= f_2(w_2 g_1 + b_2) \\ &\vdots \\ y &= f_K(w_k g_{k-1} + b_K) \end{aligned} \tag{1.3}$$

In this model, the parameters are the weights $w_1, ..., w_k$ and biases $b_1, ..., b_k$, and the structure of the model is predefined by the choice of the activation functions $f_1, ..., f_k$ and the number of layers k. The intermediate variables $g_1, ..., g_{k-1}$ are the outputs of the hidden layers, aptly named since they are not the input or the output of the model.

In each layer, the neural network transforms input data by applying a nonlinear function to a weighted sum of the inputs. These transformed outputs, termed features, serve as inputs for subsequent layers. Through a series of such transformations, the neural network learns to extract multiple layers of nonlinear features, such as edges and shapes, which are then amalgamated in a final layer to make predictions about more complex objects. Different layers may enact diverse transformations on their inputs. Signals traverse from the initial layer (the input layer) to the terminal layer (the output layer), potentially traversing through multiple intermediary layers (hidden layers). A network earns the designation of a deep neural network if it encompasses at least 2 hidden layers.

Neural networks are typically trained via empirical risk minimization, a method predicated on optimizing the network's parameters to reduce the discrepancy, or empirical risk, between the predicted output and the actual target values within a given dataset. Gradient-based techniques, such as back-propagation, are commonly employed to estimate the network's parameters. Throughout the training phase, neural networks learn from labeled training data by iteratively adjusting their parameters to minimize a designated loss function. This iterative process enables the network to generalize its predictions to unseen data.

1.1.5 Deep Learning Models

A Deep Learning model comprises a neural network with internal parameters, or weights, tailored to map inputs to outputs. In Image Classification tasks, the inputs typically entail the pixels extracted from a camera image, while the outputs encompass the feasible categories or classes that the model is instructed to identify. These categories might range from a vast array of options, such as 1000 different objects, to a binary choice of just two. To train the model to recognize images,

numerous labeled examples are repeatedly presented to it. Once the model undergoes training, it becomes capable of processing live data and furnishing real-time results. This phase of applying the trained model to new data is termed inference.

DRL, or Deep Reinforcement Learning, represents an end-to-end data-centric methodology surpassing conventional decision-making approaches for robot control. Unlike traditional methods, DRL obviates the need for prior assumptions, manually crafted features, or predefined labels, which could introduce biases along with errors (Tai et al., 2017; Zeng, 2018). In the traditional method, the robot control pipelines typically involved sequential modules governed by handcrafted rules, as depicted in Figure 1.1. This modular approach often leads to information loss, culminating in sub-optimal performance. Furthermore, the effectiveness of traditional pipelines is heavily contingent on the rules, the designer's expertise, and the specific task at hand.

Figure 1.1: Conventional Robot Control Pipeline(Levine,2021)

1.1.6 Convolutional Neural Networks

The convolution operation unique to Convolutional Neural Networks (CNNs) merges the input data, often referred to as the feature map, from one layer with a convolution kernel, also known as a filter, to generate a transformed feature map for the subsequent layer. CNN architectures designed for image classification typically include an input layer representing the image, a sequence of hidden layers responsible for feature extraction through convolutions, and finally, a fully connected output layer dedicated to classification tasks.

During training, a CNN dynamically adapts to identify the most pertinent features according to its classification objectives. For instance, when tasked with general object recognition, the CNN would prioritize filtering information about object shapes. Conversely, when confronted with a bird recognition task, it would focus on extracting color information specific to birds. This adaptability stems from the CNN's learning process, where it discerns through training that distinct object classes exhibit varying shapes, while different bird species are more likely to differ in color than in shape.

Convolutional neural networks (CNNs) draw inspiration from biological processes, particularly the organization of the animal visual cortex. In biological systems, individual cortical neurons exhibit responsiveness to stimuli within limited areas of the visual field, termed receptive fields. These receptive fields of distinct

neurons partially overlap, collectively spanning the entire visual field. This mimicking of biological principles enhances the CNN's ability to effectively process visual information, akin to the mechanisms observed in the animal visual cortex.

CNNs require relatively minimal preprocessing compared to other image classification algorithms. Instead, the network autonomously learns to optimize filters or kernels through automated learning, as opposed to traditional algorithms where these filters are manually engineered. This independence from prior knowledge and human intervention in feature extraction presents a significant advantage.

A CNN typically comprises an input layer, hidden layers, and an output layer. Among the hidden layers are one or more convolutional layers responsible for performing convolutions. Usually, this involves a layer that conducts a dot product of the convolution kernel with the input matrix of the layer. The resulting product, often a Frobenius inner product, is then subjected to an activation function, commonly ReLU.

As the convolution kernel traverses the input matrix of the layer, the convolution operation produces a feature map, which subsequently contributes to the input of the following layer. This process is followed by other layers such as pooling layers, fully connected layers, and normalization layers.

In a CNN, the input is represented as a tensor with the following shape:

(number of inputs) × (input height) × (input width) × (input channels)

Upon traversing a convolutional layer, the image transitions into a feature map, also known as an activation map, with the shape:

(number of inputs) × (feature map height) × (feature map width) × (feature map channels).

Convolutional layers perform convolutions on the input and propagate the outcome to the subsequent layer. This process resembles the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron exclusively processes data within its receptive field.

Pooling layers

Convolutional networks often incorporate both local and/or global pooling layers alongside traditional convolutional layers. Pooling layers serve to reduce the dimensionality of data by amalgamating the outputs of neuron clusters at one layer into a single neuron in the subsequent layer.

Local pooling involves combining small clusters, with tiling sizes such as 2 × 2 being commonly utilized. Global pooling, on the other hand, operates across all neurons of the feature map.

Two prevalent types of pooling are widely used: max pooling and average pooling. Max pooling selects the maximum value from each local cluster of neurons in

the feature map, while average pooling computes the average value.

Hyperparameters of CNN

Hyperparameters are diverse settings utilized to regulate the learning process. Compared to a standard multilayer perceptron (MLP), CNNs employ a greater number of hyperparameters.

- **Kernel size** The kernel refers to the number of pixels processed collectively. It's commonly denoted by the dimensions of the kernel, such as 2x2 or 3x3.
- **Padding** Padding involves adding (usually) zero-valued pixels around the borders of an image. This ensures that the border pixels are not undervalued or lost in the output, as they would otherwise only be involved in a single receptive field instance. The amount of padding applied is typically one less than the corresponding kernel dimension. For instance, a convolutional layer utilizing 3x3 kernels would require a 2-pixel padding, meaning one pixel is added on each side of the image.
- **Stride** The stride refers to the number of pixels by which the analysis window shifts during each iteration. A stride of 2 signifies that each kernel is displaced by 2 pixels from its preceding position.
- **Number of filters** As the depth increases, the size of the feature map tends to decrease, resulting in layers closer to the input layer typically having fewer filters, while higher layers may have more. To maintain computational balance at each layer, the product of feature values and pixel position (va) is kept approximately constant across layers. Preserving more information about the input would necessitate ensuring that the total number of activations (i.e., the number of feature maps times the number of pixel positions) remains non-decreasing from one layer to the next.
- **Pooling type and size** Max pooling is commonly employed, frequently utilizing a 2x2 dimension. This practice results in significant downsampling of the input, thereby reducing processing costs. For larger input volumes, particularly in lower layers, 4x4 pooling may be warranted. However, increased pooling diminishes the signal's dimensionality and may lead to undesirable information loss. Typically, non-overlapping pooling windows yield optimal results.

A Convolutional Neural Network (CNN) is a specialized form of neural network that autonomously learns to extract features through the optimization of filters or kernels. In a traditional fully-connected layer, processing an image of 100

× 100 pixels would necessitate a vast number of weights, around 10,000 for each neuron. Conversely, by employing cascading convolutional kernels, merely 25 neurons suffice to handle 5x5-sized portions of the image. As the network progresses through layers, it discerns higher-level features from broader contextual windows, contrasting with the narrower focus of lower-level features. CNNs are also referred to as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), owing to their architecture where convolution kernels or filters, with shared weights, traverse input features, yielding translation-equivariant responses termed feature maps.

1.1.7 Activation function

An activation function within an artificial neural network computes the output of a node by considering its inputs along with their respective weights. Nonlinear activation functions are pivotal as they enable solving complex problems efficiently with only a handful of nodes. Contemporary activation functions encompass various options such as the smoothed variant of ReLU, GELU, prominently featured in the 2018 BERT model. Additionally, the logistic (sigmoid) function, as utilized in the 2012 speech recognition model by Hinton et al., and the ReLU function, employed in the 2012 AlexNet computer vision model and subsequently in the 2015 ResNet model, are among the notable choices.

Comparison of activation functions

Beyond their real-world effectiveness, activation functions possess a variety of mathematical attributes. These characteristics encompass properties such as smoothness, differentiability, monotonicity, range, and their suitability for optimization procedures. These features influence the stability of training, the preservation of information flow order, the range of output values, and the network's ability to learn and generalize. Moreover, the mathematical properties of activation functions impact convergence rates and the occurrence of gradient-related challenges during training. Hence, comprehending these properties is crucial for selecting an appropriate activation function tailored to the specific demands of a neural network architecture and the task it's designed to solve.

- **Nonlinear** When the activation function exhibits non-linearity, a two-layer neural network can be demonstrated to act as a universal function approximator, a principle known as the Universal Approximation Theorem. However, the identity activation function fails to fulfill this criterion. In cases where multiple layers employ the identity activation function, the entire network simplifies to the equivalence of a single-layer model.

- **Range**

  When the activation function's range is finite, gradient-based training methods typically exhibit greater stability. This is because pattern presentations exert a significant influence on only a limited subset of weights. Conversely, when the activation function's range is infinite, training tends to be more efficient as pattern presentations affect a larger proportion of weights. In such cases, smaller learning rates are usually required to ensure stable convergence.

- **Continuously differentiable**

  This characteristic is highly sought after as it facilitates gradient-based optimization methods. Despite ReLU's lack of continuous differentiability and certain gradient-based optimization challenges, it remains feasible. Conversely, the binary step activation function lacks differentiability at 0 and differentiates to 0 for all other values, rendering gradient-based methods ineffective.

The most prevalent activation functions can be categorized into three main groups: ridge functions, radial functions, and fold functions.

Ridge activation functions

Ridge functions are multivariate functions that operate on a linear combination of input variables. Commonly utilized examples include:

- **Linear activation:**

$$f(x) = x$$

- **ReLU activation:**

$$f(x) = max(0, x)$$

- **Heaviside activation:**

$$f(x) = \begin{cases} 0, & \text{if } x < 0, \\ 1, & \text{if } x \geq 0. \end{cases}$$

- **Logistic/Sigmoid activation:**

$$f(x) = \frac{1}{(a + e^{-x})}$$

In biologically inspired neural networks, the activation function typically serves as an abstraction representing the rate of action potential firing within a cell. At

its simplest, this function adopts a binary nature, indicating whether the neuron is firing or not. Additionally, neurons are constrained from firing at rates exceeding a certain threshold. This constraint motivates the use of sigmoid activation functions, characterized by a finite interval range.

Radial activation functions

A special category of activation functions known as radial basis functions (RBFs) finds application in RBF networks, renowned for their exceptional efficiency as universal function approximators. These activation functions come in various forms, but they typically manifest as one of the following functions:

- Gaussian
- Multiquadratics
- Inverse multiquadratics
- Polyharmonic splines

Folding activation functions

Folding activation functions find extensive usage in pooling layers within convolutional neural networks, as well as in the output layers of multiclass classification networks. These activations execute aggregation operations over the inputs, which may involve calculating the mean, minimum, or maximum values. Particularly in multiclass classification tasks, the softmax activation function is frequently employed. The softmax activation function for a vector $\mathbf{y} = (y_1, y_2, \ldots, y_k)$ is defined as:

$$\text{softmax}(\mathbf{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}} \quad \text{for } i = 1, 2, \ldots, k.$$

1.1.8 Computer Vision

Computer vision is a cornerstone of artificial intelligence (AI), empowering computers and systems to glean valuable insights from digital images, videos, and visual data, thereby facilitating informed actions and recommendations. While AI fosters computational reasoning, computer vision enables machines to perceive, observe, and understand visual content. This encompasses a series of steps including image acquisition, screening, analysis, identification, and information extraction.

Computer vision primarily leverages pattern recognition techniques to autonomously train and comprehend visual data. The proliferation of data availability and the willingness of companies to share such data have facilitated deep learning experts in utilizing this wealth of information to enhance the accuracy and efficiency of the

process. While machine learning algorithms were previously prominent in computer vision applications, deep learning methodologies have emerged as superior solutions in this domain. Unlike traditional machine learning techniques, deep learning methods, reliant on neural networks, require labeled data to recognize common patterns, enabling self-learning and problem-solving capabilities.

Robot vision encompasses a combination of algorithms, cameras, and additional hardware components designed to provide visual insights to robots. This capability enables machines to perform intricate visual tasks, such as robotic arms programmed to manipulate objects on a surface. Visual feedback is crucial for image and vision-guided robots, significantly enhancing their utility across various disciplines. The applications of computer vision in robotics span a wide range of tasks, including but not limited to various domains, including space robotics, industrial robotics, military robotics, medical robotics, and warehousing and distribution.

1.1.9 Warehousing and distribution

Standing all day to sort and pick goods is increasingly viewed as inconvenient and inefficient in today's rapidly evolving landscape. As exemplified by industry giants like Amazon, who acquired Kiva Systems in 2012, significant strides have been made towards automation in warehouse operations. The introduction of Autonomous Mobile Robots (AMRs) enabled Amazon to transport shelves of products without human intervention, marking a pivotal moment in warehouse automation.

Following Amazon's lead, other industry players such as FedEx and Ocado have also embraced AMRs in their operations, albeit in subsequent years. However, despite these advancements, the automation of warehouse and distribution processes remains an ongoing endeavor. While the movement of objects between shelves represents a fundamental challenge in robotics, there is still ample room for innovation and progress in achieving comprehensive automation within the industry.

1.2 Problem statement

In this project our focus will be in making an autonomous mobile robot, that will be powered by our machine learning algorithm and will be taking inputs from the help of computer vision. We will deploy this mobile robot as a worker in an inventory transport system where it will transport packages from one place to designated through self- navigation and do some other inventory transport related work which will unfortunately depend upon the processing power, we will have in our hand due

to budget restriction.

The majority of problem that existing in this project are basically designing and deploying of the reinforcement learning model working on an underpower hardware. The goal is to get a balance between the power of the modal versus the capability of the hardware.

1.3 Objectives

1.3.1 Main objective

The primary goal of this project is to design and construct an autonomous mobile robot. Additionally, we aim to develop and implement a machine learning algorithm to facilitate the successful completion of this project.

1.3.2 Specific objective

- To model and fabricate an autonomous mobile robot.
- To design and deploy a machine learning algorithm.
- To demonstrate the effectiveness of the system.

1.4 Limitations

The following limitation are within this project:

1. The amount of complexity our model can handle, as training a larger model will not be possible due to our computation limitation.
2. The number of things we can add to our automation task is limited due to the processing power we have, limiting us to only demonstrate task that we could optimize.
3. Our model cannot move continuously due to inavailability of higher resolution camera due to budget constrain. We had to stop our robot model at every position to take an clear image of the surroundings and process it. While taking picture continuously moving our robot, we got blurred image which was one of the problem.

Chapter 2

LITERATURE REVIEW

2.1 Autonomous navigation

2.1.1 Traditional system

In traditional robot navigation systems, there are typically three key elements: mapping, localization, and path planning (refer to Figure 2.1). The mapping system is responsible for creating a global map of the unknown environment. This is commonly done using techniques such as Simultaneous Localization and Mapping (SLAM), or sometimes humans manually create the map using data from ranging and visual sensors (Ruan et al., 2019; Zhu, K. & Zhang, T., 2021). The path planning module comprises both global and local planners. The global planner offers a point of reference to the local planner, which then uses them to devise the optimal trajectory for navigation tasks, such as reaching a target while avoiding obstacles. The efficacy of the planning module relies on precise mapping, accurate localization, sensor data, and knowledge of the target position to determine the best path. (Tsai et al., 2021).

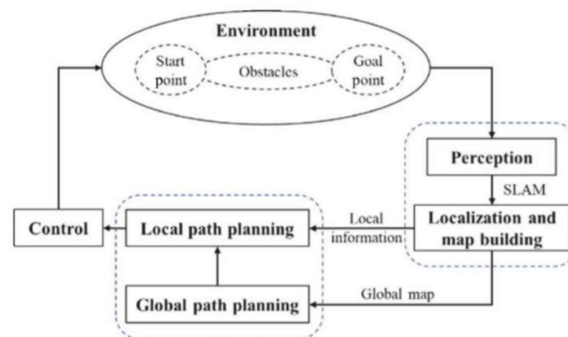


Figure 2.1: Conventional mobile robot navigation framework(Zhu, K. & Zhang, T., 2021)

In conventional system of map-based navigation systems, pose estimation and

mapping are crucial for monitoring the operation of the planning and control system computationally. While this approach ensures the utilization of an optimal path with the availability of a global map, it also presents several restraints and challenges. The contriving and updating of the environment map require significant computational resources, especially if performed dynamically, necessitating dense and precise laser sensors like those used in SLAM. Moreover, this process is vulnerable to sensor noise, adding complexity to the task. Alternatively, manual map construction is labor-intensive, time-consuming, and may require specialized resources, limiting its applicability in dynamic or unexplored environments (Tai et al., 2017; Zeng, 2018).

The conventional system of navigation framework consists of several critical elements, each deserving individual research attention. However, the integration of these elements can amplify computational errors, resulting in less-than-optimal performance (Zhu, K. & Zhang, T., 2021).

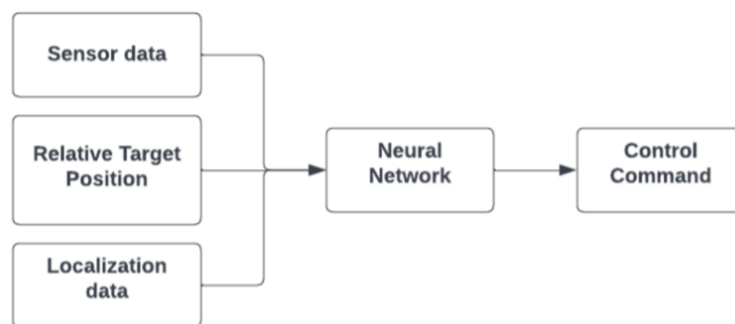


Figure 2.2: Macine learning navigation approach(on-line)

In recent years, there has been a notable focus on the development of autonomous navigation control systems for mobile robots. The map-less approach offers a solution by removing the necessity for global map information, thus reducing the reliance of the navigation system's effectiveness on the quality of the global map. Instead, the method used directly associates sensor data and relative destination positions with robot actions, predominantly leveraging neural networks (as illustrated in Figure 2.2). Additionally, the map-less approach demonstrates robust learning capabilities with decreased dependence on sensor accuracy.

Autonomous navigation control systems predominantly rely on two methodologies: LiDAR-based and vision-based methods. Both methods involve capturing environmental data using their respective sensors and then utilizing Neural Networks for establishing the mapping between the images captured by the vision sensors or the point cloud data and robot actions. This enables robots to navigate in an unexplored environments without the need for global map information (Tsai et al., 2021). However, in the absence of a global map, determining the optimal route

for navigation becomes significantly challenging. Hence, the map-less approach is commonly employed for tasks such as obstacle avoidance, where no specific destination is specified, or in scenarios where the robot's local coordinate frame provides the known destination for local navigation tasks (Xie, 2020).

It's worth noting that in recent years, several studies have emerged wherein researchers achieve the relative positioning of the robot and the target object without relying on a global map. This is accomplished using lightweight localization solutions such as WiFi, leading to what is referred to as "map-less" navigation (Zhu, K. & Zhang, T., 2021).

2.1.2 Different ML model system deployed for navigation

Reinforcement Learning (RL) is a machine learning approach where an agent learns optimal behaviors by interacting with its environment, using trial-and-error methods or making sequential decisions (Sutton, R. S. & Barto, A. G., 2018). This type of approach is the one of the kind where the agent learns the appropriate actions to take to achieve a specific goal without explicit instructions. This learning process unfolds through interactions with the agents present in the environment, aiming to maximize a numerical reward signal, which acts as feedback, guiding the agent into determining the way to map the particular states to the required type of actions. It's worth noting that the agent's actions can impact not only immediate rewards but also subsequent or delayed rewards. The distinctive feature of RL lies in the agent's capacity to learn through different the way to behave through different trial and error over the course of time during training period, optimizing for both immediate and delayed rewards (Sutton, R. S. & Barto, A. G., 2018).

The primary objective of this type of machine learning approach is for the agent to learn an optimal policy to achieve its objective while maximizing cumulative rewards. Unlike supervised and unsupervised learning approaches, RL is particularly well-suited for interactive problems, relying on learning from interactions. It finds extensive applications in complex tasks such as robot control, where solutions are determined through exploration and interaction with the environment, given the challenge of high dimensionality and the unavailability of optimal trajectories or solutions.

Apart from the agent and environment, an RL system comprises eight main elements: state S_t, action A_t, reward R_t, policy π, value function $V(s_t)$, reward discount factor γ, state transition probability matrix $P_{ss'}$, and exploration rate ε. The basic learning process of RL involves the agent performing actions A_t based on the current strategy at state S_t at time t, subsequently transitioning to state S_{t+1} at time $t+1$, and obtaining reward R_{t+1} at time t. Observations, states, actions,

and rewards are obtained through sampling. The optimal strategy is derived from the value function, which can be utilized to guide robot manipulation.

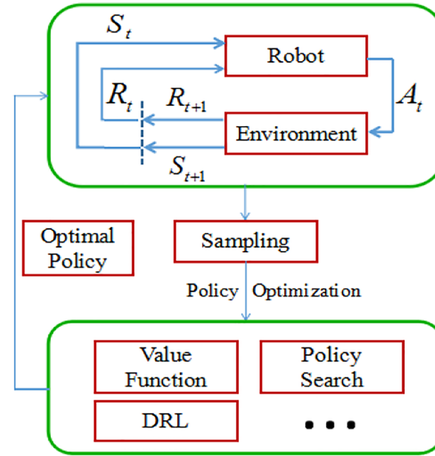


Figure 2.3: The learning process of robot based on RL(Ameer Hamza,2022)

2.2 Machine Learning approaches for autonomous navigation

In autonomous navigation, a mobile robot operates without relying on pre-built maps, such as those generated through SLAM (Simultaneous Localization and Mapping). This approach, known as reactive navigation, involves determining the navigation decisions that relies upon some crucial information extracted from the environment by processing a series of data obtained from ranging or vision sensor, without needing to have the prior knowledge of the environment (Chatterjee, A. et al., 2013). As the mobile robot relies solely on current sensor observations, autonomous navigation based on this type of approach primarily addresses the challenge of obstacle avoidance, that is essential for safe and efficient navigation. Additionally, he robot will navigate towards the local target, if a goal is provided in the local coordinate frame of the robot by the path planner, the robot will navigate towards the local target; otherwise, it will traverse through accessible regions (Xie, 2020).

Implementing autonomous navigation brings about various benefits in the field. For instance, it bypasses the laborious task of map construction and eliminates the necessity for detailed laser mapping of the working place (Zhang, P. et al., 2019a). Autonomous navigation often relies on either vision or ranging sensor-based systems, each with its unique focus. Ranging sensor-based systems directly gauge distances between objects and the robot in an environment using sensor data. This can be achieved through techniques like edge detection, potential field, and dynamic windows-based methods. On the other hand, obstacle detection using vision poses

more difficulties and continues to be a focal point for researchers interested in this domain (Xie, 2020).

2.2.1 Vision based technology for mobile robots

In various positioning techniques and methods, accurately determining both the orientation and position of a vehicle poses a common challenge. Landmark-based and map-based positioning techniques, as discussed earlier, form the foundation for vision-based positioning. Vision-based positioning utilizes optical sensors, such as laser-based range finders and photometric cameras equipped with CCD arrays (M. Baba and K. Ohtani, 2002), instead of relying on other types of sensors. These optical sensors provide a wealth of information about a mobile robot's surroundings, making them a valuable source of data for positioning tasks. However, extracting meaningful features from visual data to obtain positioning information is not a simple task and remains an active area of research.

Visual sensors indeed offer a rich source of information about a mobile robot's surroundings, making them highly valuable for various applications. However, extracting precise positioning information from observed features in visual data presents significant challenges due to the complexity of visual data analysis. As a result, researchers have devoted considerable attention to overcoming these challenges, leading to the development of various techniques. These techniques can be broadly categorized into three main categories:

- Environment representations: Techniques focusing on representing the environment in a suitable format for processing and analysis, such as feature extraction and scene modeling methods.

- Algorithms for image localization: Approaches dedicated to accurately localizing objects or landmarks within images, which often involve object detection, image segmentation, and geometric transformations.

- Sensing procedures: Methods aimed at optimizing the sensing process itself, including sensor calibration, fusion of multiple sensor modalities, and adaptive sensing strategies.

These categories encompass a wide range of techniques aimed at improving the accuracy and robustness of vision-based localization for mobile robots.

Localization techniques typically yield either absolute or relative position data, with the specific results and information varying based on factors such as the type of sensor utilized, the geometric models employed, and how the environment is represented (T. Lu & C. Tien-Hsin). Additionally, various forms of geometric data

about the surroundings can be provided, including landmarks, object shapes, or environment maps in either 2-dimensional or 3-dimensional formats.

Vision sensors, whether singular or multiple, play a crucial role in recognizing image features or areas and comparing them with stored images or maps in memory. Moreover, landmarks, object models, and maps should offer discernible information that can be easily detected or identified. In a broader sense, ”positioning” in mobile robots encompasses determining both the position and orientation of a mobile robot or sensor.

Monocular Vision System

A monocular vision system consists of a single camera mounted on the robot to capture images of its surroundings. Pre-calibration of the camera is necessary to enable feature extraction from the captured images. Common methods used in monocular vision systems include edge detection and color detection, which aid in distinguishing objects or obstacles in the image.

In a monocular vision system, the robot typically needs to move to two different points to obtain a single three-dimensional piece of information about an object (A. Ohya et al., 2001). Alternatively, multiple cameras can be configured in a monocular vision setup, where the fields of view may not overlap or may overlap only slightly. However, monocular vision alone does not provide inherent three-dimensional information and cannot precisely determine the position and distance of objects. Additionally, monocular vision systems may be susceptible to disturbances, leading to incorrect interpretations and calculations of visual data.

Despite its advantages, implementing a monocular vision system presents challenges due to the need for precise alignment of cameras, which may require expert techniques or high-precision machinery, potentially leading to high costs (S. Lecorné et al.).

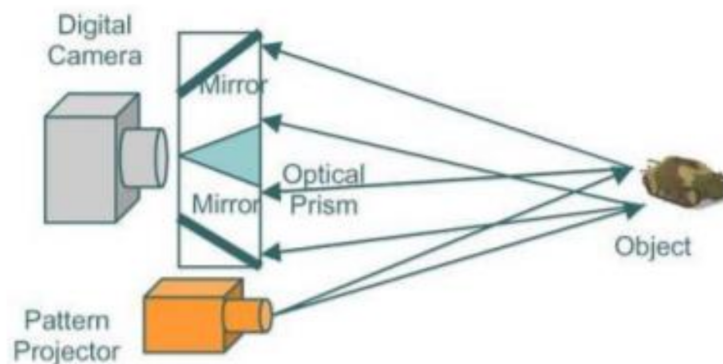


Figure 2.4: Monocular vision (A. Ohya et al.,2001)

2.2.2 Deep learning for navigation

Deep Reinforcement Learning (DRL) has emerged as a crucial research area that is gaining a lot of attention in control systems, in the field of robotics, enabling the acquisition of policies control by the processing and modelling of raw input data. Over recent years, DRL has found application in various domains, including robotic in-hand manipulation, complex locomotion behaviors in rich environments, autonomous navigation, and self-driving cars.

In the context of autonomous navigation for mobile robots, classical approaches typically involve three main elements: localization, map building, and path planning. Localization often relies on dense laser sensors like LiDAR for precise mapping and Simultaneous Localization and Mapping (SLAM). Nevertheless, such methods can be time-consuming and costly due to the requirement for precise sensor data.

One challenge with current RL methods is the need for collecting a large number of samples or experiences from the environment, which can be impractical in real-world scenarios. To address this, the robots are often trained initially in simulated environments and after thoroughly testing and validating the model in simulation, when the simulated model is up and running as expected, the model is then transferred to real robots.

Alternatively, supervised reinforcement learning can be used, where a smaller dataset is prepared, potentially reducing costs associated with simulation. However, both approaches have their limitations and trade-offs, including considerations of effectiveness, scalability, and generalization to real-world environments.

2.2.3 Vision based approaches

Xie et al. (2017) conducted a study where they utilized a single RGB images that was captured from a monocular camera, along with depth prediction obtained through a FCRN, as an input data. They trained their robot model initially in the Gazebo simulator environment and later validated it in the real world scenario utilizing a Turtlebot robot, without requiring additional training. They deliberately corrupted predicted depth images with random noise and blur to improve the transferability and generalization of their model,

Furthermore, the authors introduced an end-to-end dueling deep double-Q network (D3QN) approach. This architecture took a stack of four depth images as an input and generated linear and angular velocities from the discrete action space, facilitating the avoidance of obstacle and efficient navigation around the trained environment. The robot operated autonomously without a specific goal, navigating

through traversable areas while avoiding obstacles. The study aimed to demonstrate the effectiveness of their proposed approach in real-world scenarios.

In addition to the comparison with the DDQN and DQN models, the D3QN model showcased superior training efficiency and performance in Xie et al.'s (2017) study. Similarly, Ruan et al. (2019) conducted research with a methodology akin to Xie's work. However, instead of performing depth prediction initially, they utilized a Kinetic sensor to aquire RGB and RGB-D image data which were fed to the model for training the robot. This demonstrates the versatility of different sensor modalities in training autonomous navigation systems.

Tran & Ly (2020) and Chaffre et al. (2020) applied the Soft-Actor Critic (SAC) reinforcement learning algorithm for depth image-based autonomous mobile robot navigation in an unexplored environments. Their objective was to reach a given target or waypoint without prior map information, employing a continuous action space. 10 depth values information obtained from the image were utilized to simulate a laser system, as well as the last action, serving as the state observation in their studies.

In the work of Tran & Ly (2020), the primary focus was on devising a cost-effective solution for autonomous navigation by employing only a monocular camera to replace laser-finding sensors, complemented by Monocular Depth Estimation. Rather than utilizing the entire depth image, only 10 values were sampled from the middle row of the image. The study proceeded to compare the performance of using lasers with a monocular camera in two distinct scenarios. However, it's worth noting that all experiments were conducted solely in simulation, with limited extensive evaluation in real-world settings.

In contrast, Chaffre et al. (2020) opted to utilize depth images directly obtained from the Intel RealSense D435 sensor. Their study embraced an incremental approach, wherein the agent underwent training in 3 distinct environments with escalating complexity. Additionally, data from the depth sensors were stacked to enhance performance. Then after collecting data, training occurred within the ROS-Gazebo simulator environment which subsequently underwent real-world testing using the WIFI Bot Lab v4 mobile robot. Despite employing incremental training and observation stacking, the authors found that the best success rate attained was only 47%, underscoring the inherent limitations of utilizing RL techniques in conjunction with depth sensors for autonomous navigation.

Ma et al. (2019) adopt a distinctive strategy by directly employing RGB images as the visual input for autonomous navigation tasks. Although RGB images are widely used, they are often underutilized directly in navigation due to challenges such as sample inefficiency, simulation-reality disparities, and limited generalizability.

To address these challenges, Ma et al. (2019) proposed a solution leveraging Variational Autoencoder (VAE). Their method involves using a VAE encoding images to extract visual features and generating a low-dimensional latent vector. This latent vector, along with target and motion information, serves as input for the motion planner to train the network and generate continuous velocity commands. This strategy effectively separates the input visual features from the Deep Reinforcement Learning (DRL) network, mitigating issues of sample inefficiency and limited generalizability often encountered when directly employing RGB images for navigation tasks.

The paper begins by setting up a guigeline using an end-to-end deep learning networks without decoupling. In this configuration, RGB image inputs undergo convolutional layers for downsampling, and the model is trained using both Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) algorithms. After training, the performance of both models is evaluated, and PPO demonstrates significantly better performance compared to DQN. Therefore, PPO is selected as the superior model for further comparison and experimentation.

The evaluation of the proposed approach, which combines Variational Autoencoder (VAE) with the benchmark Deep Reinforcement Learning (DRL) algorithm (PPO), demonstrates promising results. Specifically, the findings indicate that only a quarter of the sampled data were necessary to get to the comparable performance of the end-to-end approach when utilizing Variational Auto Encoders with DRL. This significant reduction in the number of samples required highlights the effectiveness of the proposed method in enhancing sample efficiency and overcoming the challenges associated with using RGB images directly for navigation tasks.

Chapter 3

METHODOLOGY

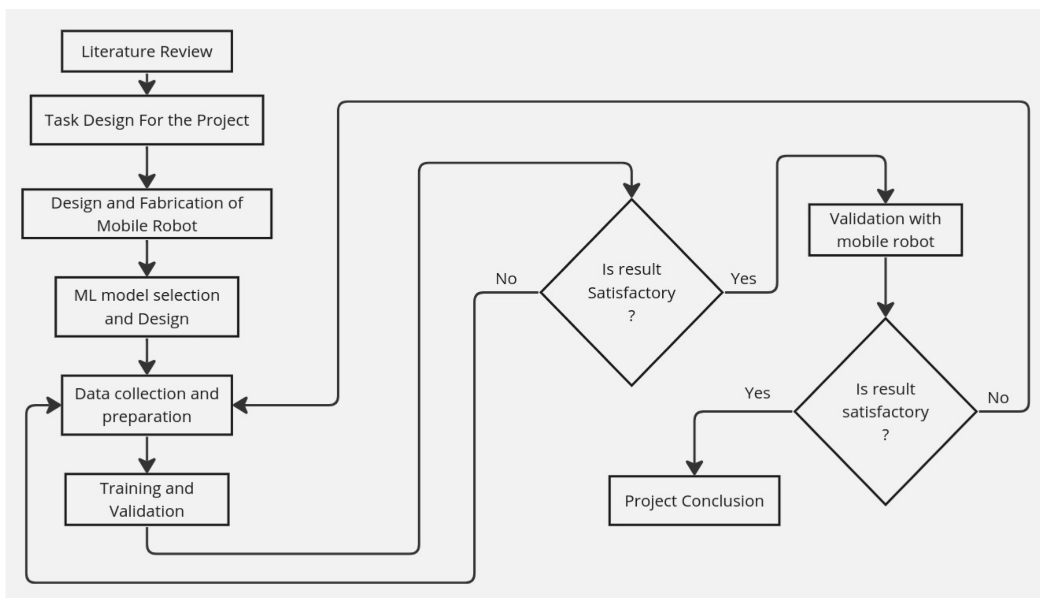


Figure 3.1: Process flowchart

3.1 Design and fabrication phase

In this phase, we carried out the mechanical portion of our project. We started with the design of an autonomous mobile robot first, then we designed our path and task structure, after this we started the fabrication process, the component was chosen as per the requirement with the economic constrain in mind.

The robotic vehicle made for this demonstration is non-holonomic two wheeled differential drive robot which has two actuators one for each wheel. Thus the robot has a total of 3 degree of freedom excluding the belt and pulley mechanism to lift loads which makes a total of 4 degrees of freedom. That is the robot can move forward or backward in one direction and can rotate clockwise and anticlockwise and hence 3 degrees of freedom and from the lifting mechanism we get one degree

of freedom which makes it a total of 4. Each wheel are actuated by a DC geared motor with gear ratio of 1:90 and a wheel radius of 75 cm.

3.1.1 Differential Drive Control

Mathematical modeling of mobile kinematics plays a crucial role in control system design for robots. The kinematic model is responsible for transforming the velocity of the robot into a generalized coordinate vector. This transformation is achieved through the following equation:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \cos\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \tag{3.1}$$

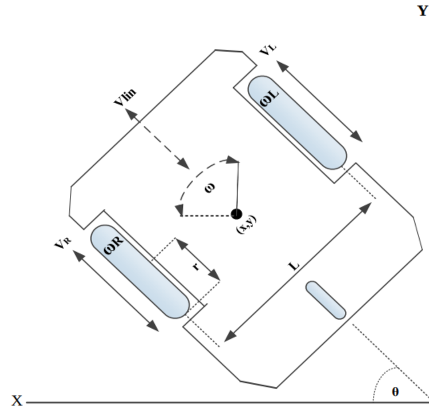


Fig.1. Parameters of mobile robot

Figure 3.2: Parameters of mobile robot

The velocity of the robot can be obtained from the velocity of each wheel using the following equation:

$$v = \frac{r(\omega_R + \omega_L)}{2} \tag{3.2}$$

$$\omega = \frac{r(\omega_R - \omega_L)}{L} \tag{3.3}$$

H-Bridge

An H-bridge employs two pairs of transistors, typically MOSFETs, to regulate the direction of current flow through the motor. By altering the direction of current flow, achieved by reversing the voltage across the motor terminals, the rotation direction can be changed. Utilizing PWM control, brushed motors can be effectively

controlled for both speed and direction. This is facilitated by pulsing the MOSFETs to drive the motor. Additionally, four diodes are incorporated into H-bridge designs to safeguard against voltages generated by the motor during operation.

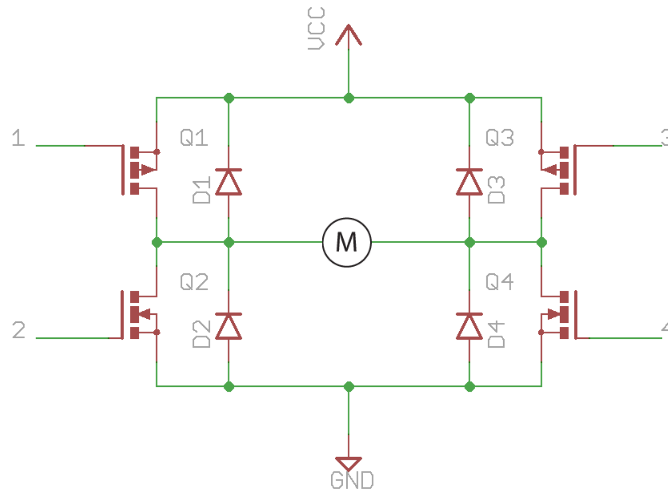


Figure 3.3: H-Bridge Circuit

1. **Directional Control**

   To rotate the motor in one direction, two of the FETs must be activated in the manner depicted in Figure 3.4. This configuration enables current to flow through the motor, thereby initiating motion.

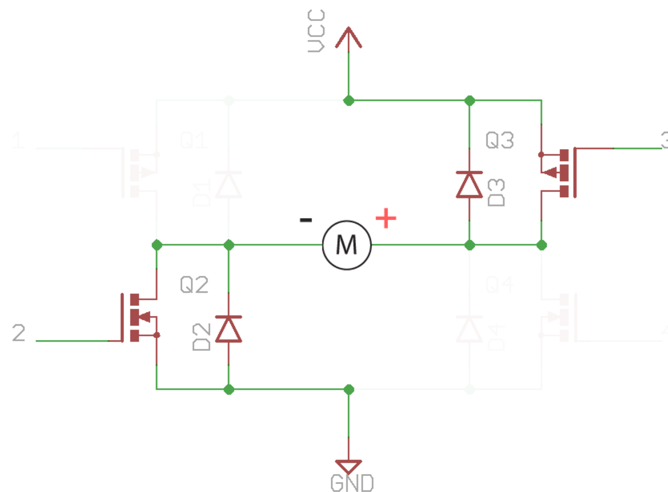


Figure 3.4: Direction Control(a)

   To rotate the motor in the opposite direction, the complementary pair of FETs is activated (while the others are deactivated), resulting in the reversal of current flow. This configuration is illustrated in Figure 3.5:

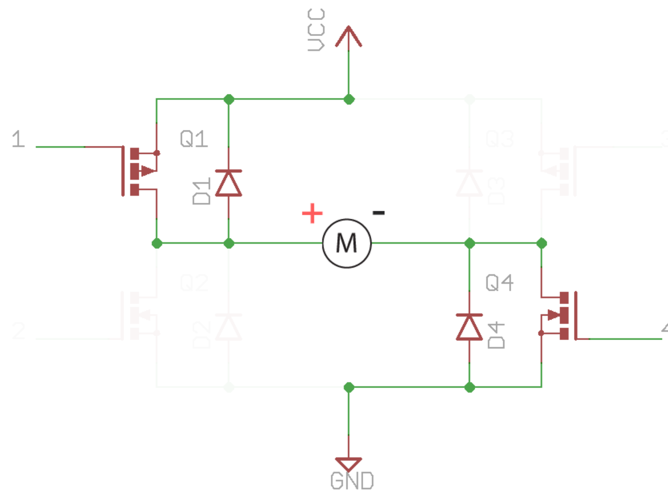


Figure 3.5: Direction control(b)

2. **Braking**

   To halt the motor's rotation, all FETs are deactivated, effectively disconnecting the motor from the power source. However, due to the motor's inertia, it will gradually decelerate until it comes to a complete stop. For an immediate stop, a matching voltage can be applied to both sides, a technique known as braking, depicted in Figure 3.6.

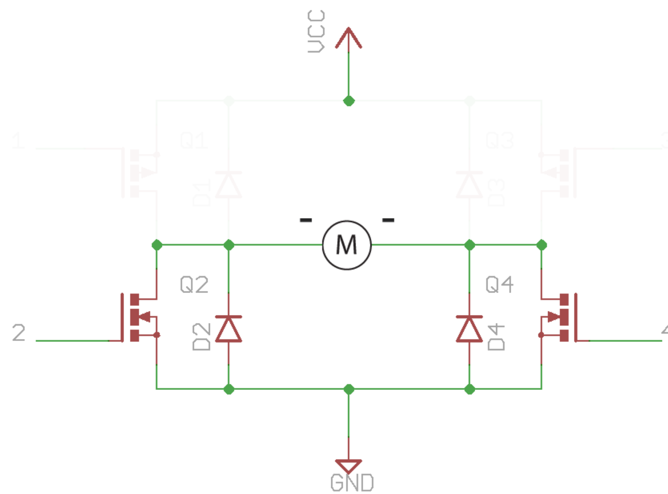


Figure 3.6: Braking

3.1.2 Stepper Motor Control

A stepper motor is a form of brush-less DC electric motor that divides a full rotation into a series of uniform steps. Each step corresponds to a distinct angular movement, enabling precise control over the motor's position. Stepper motors can be directed to move to a specific step or rotate continuously. This control mechanism eliminates the necessity for costly sensing and feedback devices like optical

encoders. Instead, the position is determined by tracking the input step pulses. Stepper motors are renowned for their versatility in positioning systems, as they can be digitally controlled as part of an open-loop system. Compared to closed-loop servo systems, they are typically simpler and more robust.

Fundamentals of operation:

When a coil winding is energized, it generates an electromagnetic field characterized by a north and south pole. This magnetic field induces magnetization in the rotor, causing it to align itself with the magnetic field. The principle of magnetic attraction dictates that unlike poles attract each other, thus facilitating the alignment of the rotor with the magnetic field.

By manipulating the direction of the magnetic field, rotation of the rotor can be induced. This manipulation can be achieved by altering the direction of current flow through the coil winding, thereby changing the orientation of the electromagnetic field and consequently prompting rotational motion in the rotor.

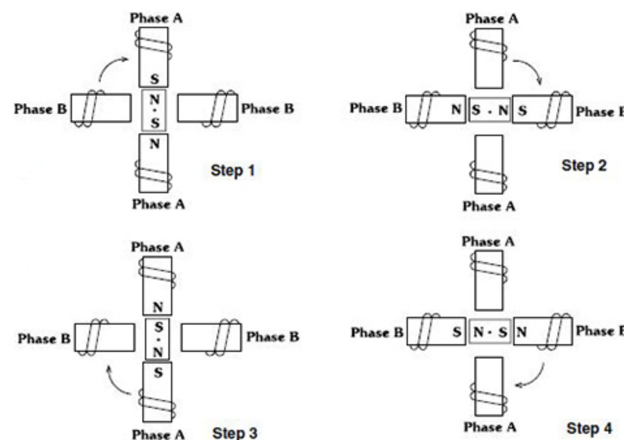


Figure 3.7: Single phase stepping

Figure 3.7 illustrates a typical step sequence for a two-phase motor. In Step 1, phase A is energized, locking the rotor in the depicted position. In Step 2, phase A is deactivated, and phase B is activated, causing the rotor to rotate 90° clockwise. In Step 3, phase A is reactivated with reversed polarity, and in Step 4, phase B is activated with reversed polarity. This sequence completes a full revolution of the rotor. Repetition of this sequence results in the rotor rotating clockwise in 90° increments. This method is known as ”one phase on” stepping.

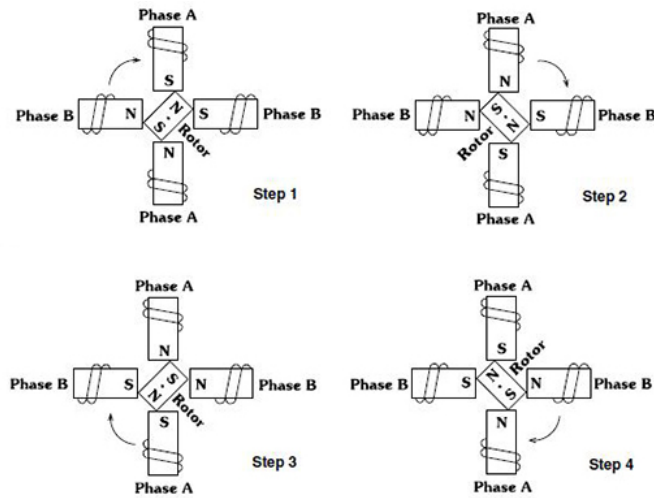


Figure 3.8: Dual phase stepping

Figure 3.8 depicts a more common ”two phases on” stepping method, where both phases are continuously energized. In this stepping mechanism, the rotor aligns itself between the poles created by the energized phases. This approach provides 41.4% more torque compared to ”one phase on” stepping but necessitates twice the input power.

We can increase the turning force by combining both single phase stepping and dual phase stepping which is called half stepping as illustrated by figure 3.9.

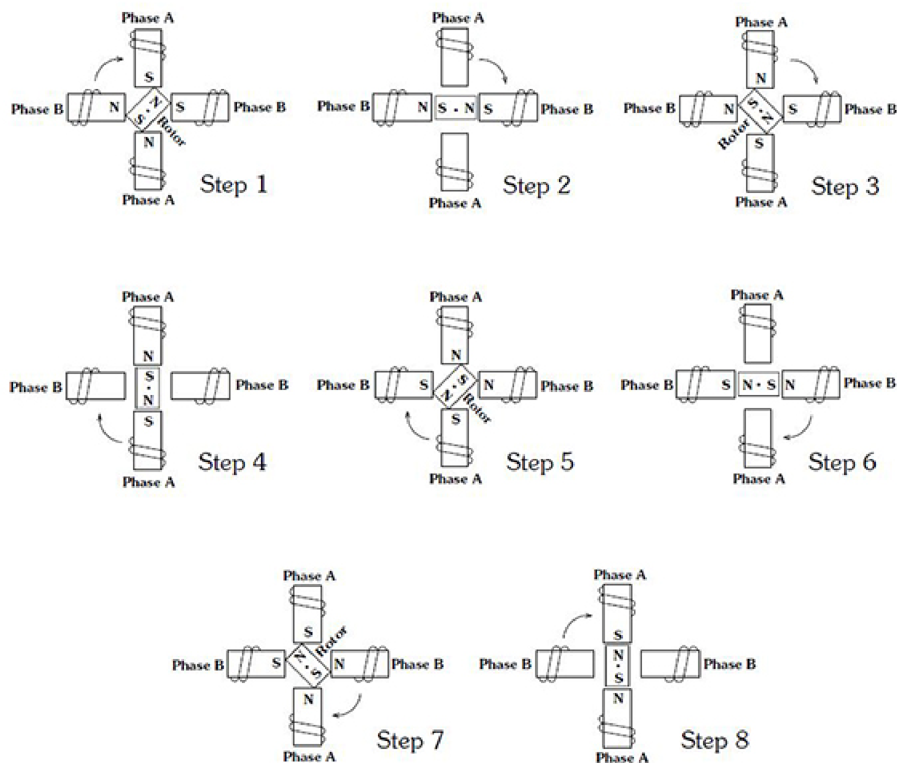


Figure 3.9: Half Stepping

Table 3.1: Control sequence for half stepping of stepper motor

		1	2	3	4	5	6	7	8
Coil 1	Pin 1	1	1	0	0	0	0	0	1
Coil 2	Pin 2	0	1	1	1	0	0	0	0
Coil 1	Pin 3	0	0	0	1	1	1	0	0
Coil 2	Pin 4	0	0	0	0	0	1	1	1

Torque-speed characteristics of 28BJY stepper motor

The torque with varying speed for 28BJY stepper motor can be determined using the following graph as shown below in figure 3.10.

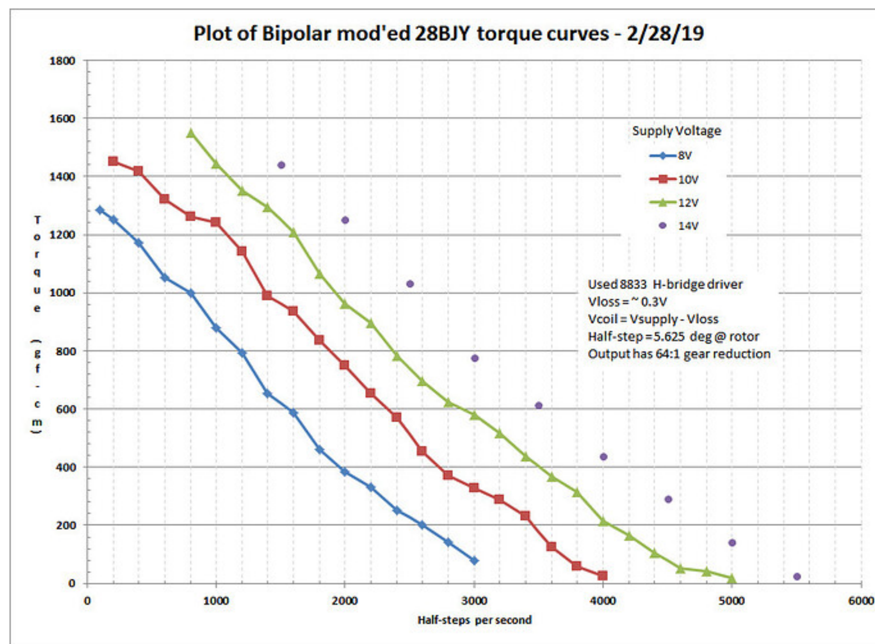


Figure 3.10: Torque speed characteristics(arduino forum)

3.1.3 Mechanical Calculation

We've employed timing belts to transfer torque and motion from a driving pulley to a driven pulley within a power transmission drive. Additionally, we've utilized them to convey force to a positioning platform of a linear actuator from a stepper motor.

In the operation of a belt drive under load, a disparity in belt tensions arises between the entering (tight) and leaving (slack) sides of the driver pulley. This disparity is termed effective tension, denoted as T_e, and it signifies the force transmitted from the driver pulley to the belt.

$$T_e = T_1 - T_2 \tag{3.4}$$

where T_1 and T_2 are the tight and slack side tensions, respectively.

The driving torque M is given by:

$$M = T_e \left(\frac{d}{2}\right) \tag{3.5}$$

where d is the pitch diameter of the driving pulley.

The effective tension generated at the driver pulley represents the actual working force responsible for overcoming the overall resistance to belt motion. It's imperative to identify and quantify the collective sum of individual forces acting on the belt, which collectively contribute to the effective tension required at the driver pulley.

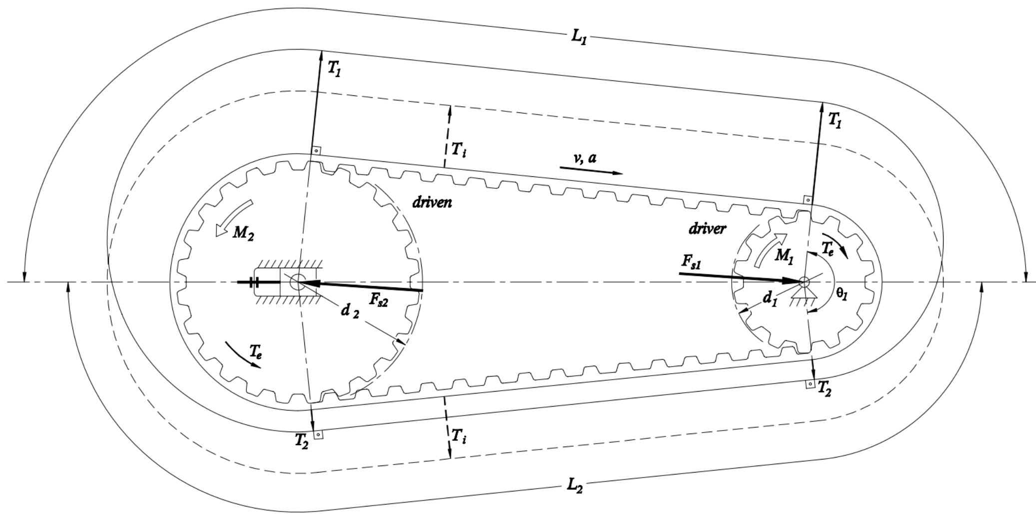


Figure 3.11: Power transmission and rotary positioning(Gates Mectrol, April 2001)

In power transmission drives (as depicted in Figure 3.11), the resistance to motion arises at the driven pulley. The force transmitted from the belt to the driven pulley equals T_e. The expressions for torque requirement at the driver can be formulated as follows:

$$M_1 = T_e \left(\frac{d}{2}\right) = \frac{M_2}{\eta}\frac{d_1}{d_2} \tag{3.6}$$

where M_1 is the driving torque, M_2 is the torque requirement at the driven pulley, P_2 is the power requirement at the driven pulley, ω1 and ω2 are the angular speeds of the driver and driven pulley respectively, d_1 and d_2 are the pitch diameters of the driver and driven pulley respectively, and η is the efficiency of the belt drives (η = 0.94 - 0.96 typically).

From figure 3.10, we find that the pull in torque for the stepper motor driven

at 8 Volts at half stepping of 600 is 1050 gfcm.

$$\begin{aligned} M_1 &= 1050\,\text{gfcm} \\ &= 0.103005\,\text{Nm} \end{aligned}$$

Then for the pulley of diameterical pitch 10.186mm directly coupled with the shaft of stepper motor, effective tension can be calculated using equation 3.6 as:

$$\begin{aligned} T_e &= \frac{2M_1}{d_1} \\ &= \frac{2 \times 0.103005}{10.186 \times 10^{-3}} \\ &= 20.225\text{N} \end{aligned}$$

Now the torque at the driven pulley $(d_2 = 12.73\text{mm})$ is given by equation 3.6 as:

$$\begin{aligned} M_2 &= \frac{M_1 \cdot \eta \cdot d_2}{d_1} \\ &= \frac{0.103005 \times 0.95 \times 12.73 \times 10^{-3}}{10.186 \times 10^{-3}} \\ &= 0.1223\text{Nm} \end{aligned}$$

The torque M_2 is then transmitted to the linear positioner which is responsible to lift and carry the load which can be illustrated by figure 3.12

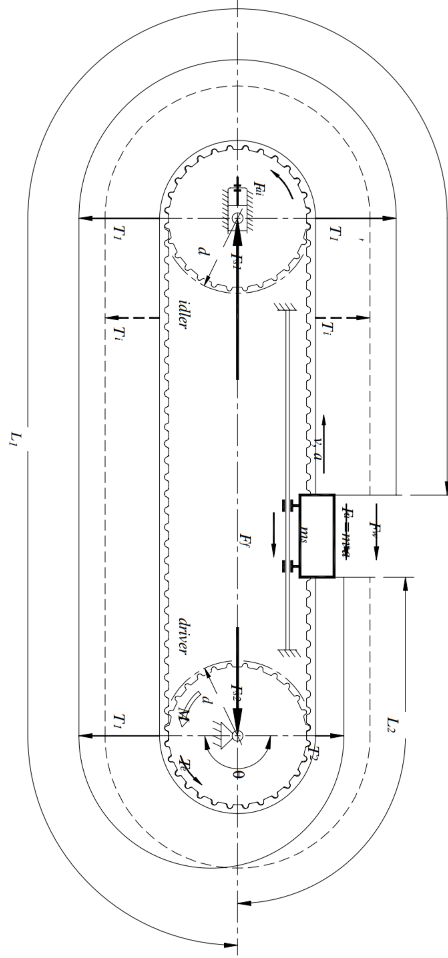


Figure 3.12: Vertical linear positioner(Gates Mectrol, April 2001)

n a linear positioner, the primary load is exerted on the positioning platform, also known as the slider. It encompasses several forces, consisting of acceleration force F_a that results from linear acceleration of the slider, frictional force of the linear bearing, F_f, external force, F_w, weight of the load along with slide and fork F_g, inertial forces to accelerate belt, F_{ab}, and the force exerted on the idler pulley due to rotation, F_{ai}.

$$T_e = F_a + F_w + F_g + F_f + F_{ab} + F_{ai} \tag{3.7}$$

The individual components of T_e is given by:

$$F_a = m_l \cdot a \tag{3.8}$$

where m_l is the mass of the load being lifted and a is the linear acceleration rate of the slider.

$$F_g = m_l \cdot g \tag{3.9}$$

$$F_f = \mu \cdot m_l \cdot g \tag{3.10}$$

$$F_{ab} = \frac{w_b \cdot L \cdot b}{g} \cdot a \tag{3.11}$$

where μ is the dynamic coefficient of friction of the linear bearing, L is the length of the belt, b is the width of the belt, w is the specific weight of the belt and g is the gravity.

$$F_{ai} = \frac{2 \cdot \alpha \cdot J_i}{d} \tag{3.12}$$

where J_i is the inertia of the idler pulley, α is the angular acceleration of the idler.

For the linear positioner, the effective tension T_e for a pulley of diametrical pitch 12.73mm can be calculated as using equation 3.4 as:

$$\begin{aligned} T_e &= \frac{2M_2}{d_1} \\ &= \frac{2 \times 0.1223}{12.73 \times 10^{-3}} \\ t &= 19.2145\text{N} \end{aligned}$$

Here the load is lifted at constant velocity and hence the linear acceleration is zero which reduces the term F_a, F_{ab}, F_{ai} to zero. Hence, by taking a factor of safety of 2 ($f_s = 2$), the effective tension after incorporating the factor of safety becomes T_e/f_s. Using an equation 3.7 and substituting the F_g and F_f from equations 3.9 and 3.10 respectively, we can calculate the mass of the load that cam be lifted by our robot as:

$$\begin{aligned} \frac{T_e}{f_s} &= F_g + F_f \\ \frac{T_e}{f_s} &= m_l \cdot g + \mu \cdot m_l \cdot g \\ m_l &= \frac{f_{se}}{f_s g(1+\mu)} \\ &= \frac{19.2145}{2 \times 9.81(1+0.003} \\ &= 0.975\text{kg} \end{aligned}$$

Therefore, the robot designed in case of our project can lift up load up to 0.975kg with a factor of safety of 2.

3.1.4 3D Robot Model

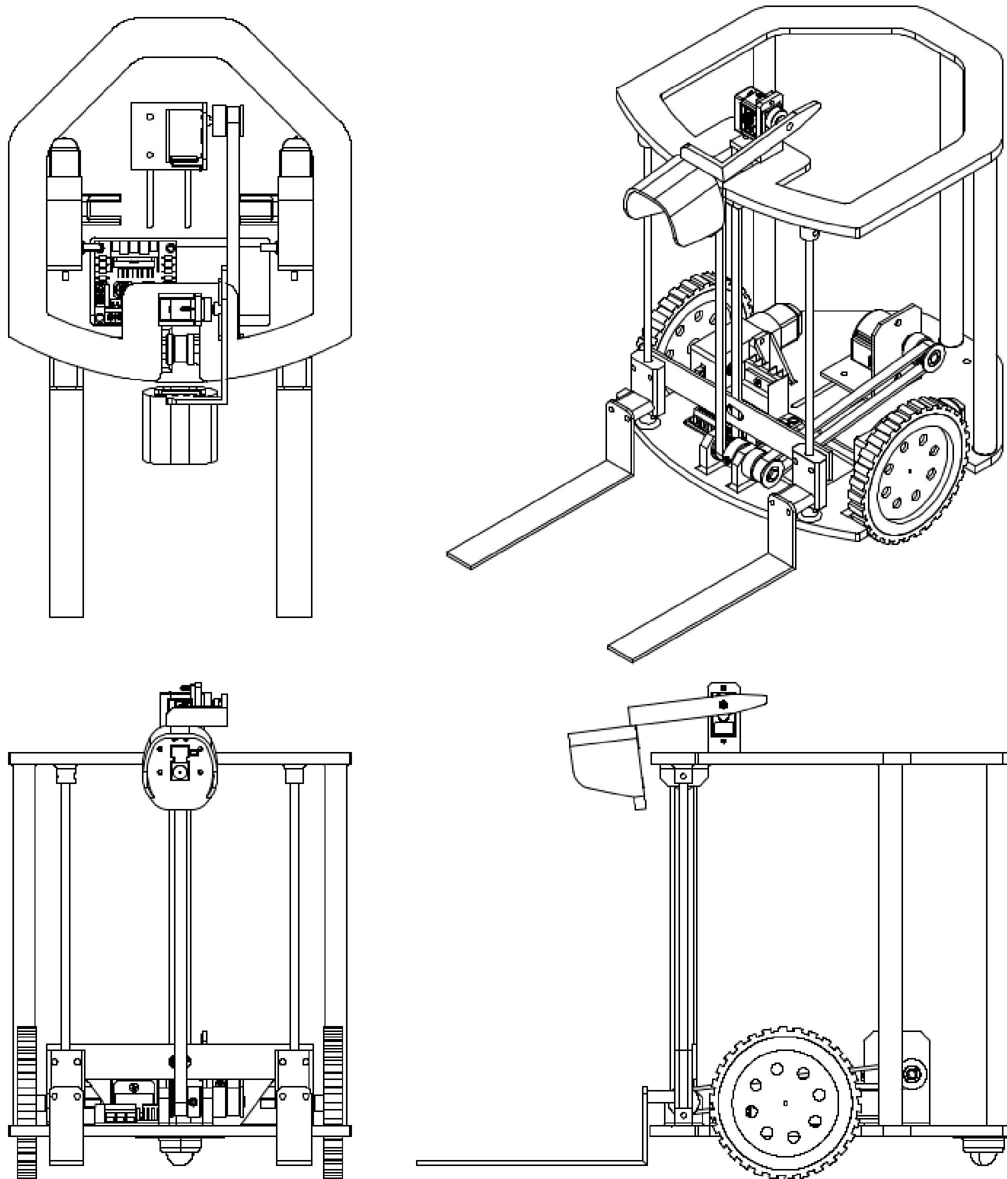


Figure 3.13: Robot Model (Top View, Isometric View, Front View, Side View)

3.1.5 Fabricated Model

The robot frame is mostly 3D printed using PLA and the lifting mechanism slides over a steel rod of 4mm diameter. The steel rod as well as carbon fiber pipe supports the top and base frame of the robot.

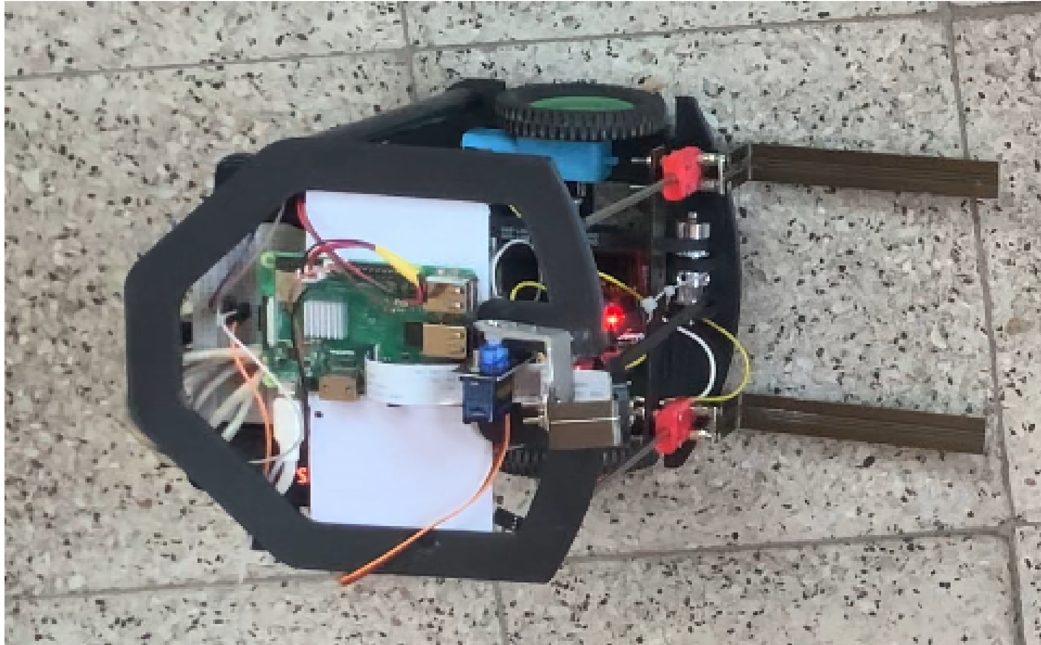


Figure 3.14: Top view of Fabricated robot for our project

3.2 Pytorch-based architecture

PyTorch indeed offers a versatile framework for designing deep learning architectures, boasting a rich ecosystem of tools and modules to facilitate model development. Leveraging PyTorch enables users to harness its capabilities, including high-level features like tensor computation akin to NumPy, coupled with robust GPU acceleration. Additionally, PyTorch offers TorchScript, simplifying the transition between eager mode and graph mode, enhancing flexibility in model deployment.

Moreover, with the latest PyTorch release, the framework introduces several advanced features such as graph-based execution, distributed training capabilities, support for mobile deployment, and model quantization. These enhancements further bolster PyTorch's utility and appeal to researchers and practitioners in the deep learning community, empowering them to tackle a wide range of tasks efficiently and effectively.

Indeed, in PyTorch, neural networks are built using layers or modules that process data. The torch.nn namespace provides a comprehensive set of components for constructing these networks. Each module in PyTorch is derived from the nn.Module class, enabling a consistent and intuitive interface for building neural network architectures.

A neural network itself is considered a module, which can contain other modules or layers as its components. This hierarchical organization allows for the creation of complex architectures by composing simpler building blocks. With this modular approach, developers can easily design and manage intricate neural net-

works, facilitating experimentation and innovation in deep learning research and applications.

The dataset is prepared using a custom dataset class (CustomImageDataset) inheriting from PyTorch's Dataset class. It loads images from specified directories and applies transformations such as converting images to tensors. Then neural network architecture is defined within a class named Resnet, which inherits from ImageClassificationBase. The model concludes with fully connected layers (linear) for classification. Then the training loop is defined which implements one-cycle learning rate scheduling and gradient clipping. The Adam optimizer (opt_func) is used for parameter optimization, with a specified weight decay for regularization. At each epoch, the model is trained on the training dataset (train_dl) and evaluated on the validation dataset (valid_dl). Training progress is monitored, and results are printed at the end of each epoch. Evaluation metrics such as loss and accuracy are calculated during both training and validation phases. The evaluate function computes these metrics on the validation dataset. This methodology involves dataset preparation, defining a CNN architecture with residual connections, training the model using the one-cycle learning rate scheduling strategy, and evaluating the model's performance.

3.2.1 Dataset Preparation

The dataset is prepared using a custom dataset class (CustomImageDataset) inheriting from PyTorch's Dataset class. It loads images from specified directories and applies transformations such as converting images to tensors.

```
1  class CustomImageDataset(Dataset):
2      def __init__(self, data_dir, transform=None):
3          self.data_dir = data_dir
4          if data_dir == rf'E:\trainv2\forSL\f':
5           i=0
6          elif data_dir == rf'E:\trainv2\forSL\b':
7           i=1
8          elif data_dir == rf'E:\trainv2\forSL\l':
9           i=2
10         elif data_dir == rf'E:\trainv2\forSL\r':
11          i=3
12         self.transform = transform
13         # Get a list of file names in the data directory.
14         self.file_list = [filename for filename in os.listdir(data_dir)
           ↪  if filename.endswith('.jpg')]
15         self.dataset = [[self.__getitem__((filename)), i] for filename
           ↪  in self.file_list]
16     def __len__(self):
```

```
        return len(self.file_list)
    def __getitem__(self, idx):
        img_name = os.path.join(self.data_dir, idx)
        image = Image.open(img_name)
        if self.transform:
            image = self.transform(image)
        return image
    def append(self, CustomImageDataset):
      for value in CustomImageDataset.dataset:
        self.dataset.append(value)

transform = transforms.Compose([
    transforms.ToTensor()
])
path1 = rf'E:\trainv2\forSL\f'
path2 = rf'E:\trainv2\forSL\b'
path3 = rf'E:\trainv2\forSL\l'
path4 = rf'E:\trainv2\forSL\r'
custom_dataset = CustomImageDataset(path1, transform=transform)
↪  #transform=None to view image
custom_dataset2 = CustomImageDataset(path2, transform=transform)
custom_dataset3 = CustomImageDataset(path3, transform=transform)
custom_dataset4 = CustomImageDataset(path4, transform=transform)
custom_dataset.append(custom_dataset2)
custom_dataset.append(custom_dataset3)
custom_dataset.append(custom_dataset4)
```

This section of the PyTorch architecture defines a custom dataset class called CustomImageDataset, which is designed to handle image data for training and evaluation of neural network models. Here's a detailed description of each component:

Initialization Method (__init__):

The __init__ method initializes the dataset with the provided data_dir (directory containing image data) and an optional transformation (transform) to be applied to the images. It also assigns a numerical label (i) based on the provided data_dir: If data_dir corresponds to the directory for front-facing images, i is set to 0. If it corresponds to the directory for back-facing images, i is set to 1. Similarly, for left-facing images, i is set to 2, and for right-facing images, i is set to 3. The list of file names (file_list) in the specified data directory is obtained using os.listdir. Additionally, the dataset is initialized as an empty list (self.dataset), which will be populated later with image data and corresponding labels.

Length Method (__len__):

The __len__ method returns the total number of images in the dataset, which is the length of the file_list.

Get Item Method (__getitem__):

The __getitem__ method retrieves the image and its corresponding label at a given index (idx) from the dataset. It constructs the path to the image file using os.path.join and the provided data directory (data_dir). The image is then opened using PIL's Image.open method. If a transformation is provided (self.transform), it is applied to the image. Finally, the transformed image is returned along with its corresponding label.

Append Method:

The append method allows for the merging of two instances of CustomImageDataset. It iterates over the dataset attribute of the provided CustomImageDataset instance (CustomImageDataset.dataset) and appends each value to the dataset attribute of the current instance (self.dataset).

Transformation Pipeline:

The transforms.Compose function creates a transformation pipeline by chaining multiple transformations together. In this case, the only transformation applied is transforms.ToTensor(), which converts PIL images to PyTorch tensors.

3.2.2 Model Architecture

The architecture typically starts by defining our neural network by subclassing nn.Module, and initialize the neural network layers in __init__. Every nn.Module subclass implements the operations on input data in the forward method.

```
1  def __init__(self):
2      super().__init__()
3      self.conv1 = conv_block(3, 8, pool=True)
4      self.res1 = nn.Sequential(conv_block(8, 8), conv_block(8, 8))
5      self.conv2 = conv_block(8, 16, pool=True)
6      self.conv3 = conv_block(16, 32, pool=True)
7      self.res2 = nn.Sequential(conv_block(32, 32), conv_block(32, 32))
8      self.conv5 = conv_block(32, 64, pool=True)
9      self.classifier = nn.Sequential(nn.MaxPool2d(2),
10                                     nn.Flatten(),
```

```
                                    nn.Dropout(0.2))
        self.linear = nn.Sequential(
            nn.Linear(576, 128),
            nn.ReLU(),
            nn.Linear(128, 4))
```

The provided Python code snippet outlines a convolutional neural network (CNN) architecture constructed using the PyTorch library. The network, defined within an initialization method, comprises a sequence of convolutional layers (conv1, conv2, conv3, conv5) with accompanying pooling layers to extract hierarchical features from input data, designed for image classification tasks. Additionally, residual blocks (res1, res2) are integrated to facilitate deeper network training by reducing the vanishing gradient issue. Pooling layers and dropout regularization are strategically applied to reduce spatial dimensions and prevent overfitting, respectively. The flattened output from convolutional layers is fed into fully connected layers (Linear) for classification. The architecture culminates in an output layer with a linear activation function, suggesting a multi-class classification task with four output classes. This neural network architecture demonstrates a blend of convolutional and residual structures, optimized for handling complex input data and achieving effective classification performance required for our project.

Resnet

ResNet introduces the concept of residual learning, which allows for the training of very deep neural networks. In a typical neural network, each layer learns a mapping from its input to its output. In ResNet, instead of learning a direct mapping, each layer learns a residual function, which is the difference between the input and the desired output.

Let x be the input to a ResNet block. The output of the block is computed as:

$$\text{output} = F(x) + x$$

Where $F(x)$ is the residual function that the block learns to approximate. This formulation allows the network to learn the identity mapping (where $F(x) = 0$), making it easier to optimize and allowing for the training of deeper networks.

Mathematically, the residual function $F(x)$ is typically implemented as a sequence of convolutional layers followed by batch normalization and activation functions, similar to a standard neural network layer.

During training, the network learns the parameters of the residual functions through backpropagation and gradient descent, adjusting the parameters to mini-

mize the difference between the output and the desired target.

In our project, our ResNet has following architect in order:

- Convolution Layer
- Residual Layer
- Convolution Layer
- Convolution Layer
- Residual Layer
- Convolution Layer

The neural network archiecture of our model can be illustrated by figure 3.15

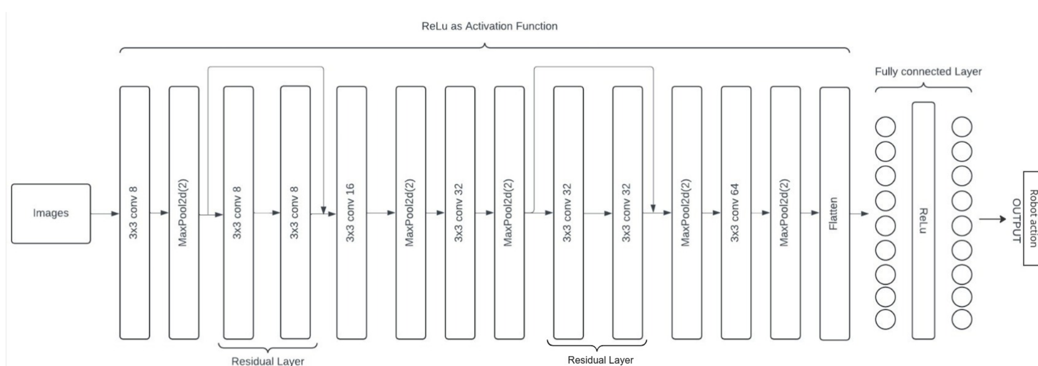


Figure 3.15: Neural Network architecture

Starting from the convolution model, we have chosen the kernel size of three, kernel is a matrix that slides through the input matrix performing element wise multiplication over the overlapping region and summing them up. It is also known as filter. We have chosen the padding value as one. Padding refers to the technique of adding additional layers of zeros around the input data before applying the convolution operation. It helps preserve the information at the borders of the input when applying filters. After this we have used an activation function called relu, which introduce non linearity in the model which is required by model to learn nonlinear function. The relu function return zero for negative value and for positive value it returns as it is. Finally, then we apply MaxPool2D function which divides the input into section of size 2*2 and return the max value from the section. This halves the size of input on to the output. This process is repeated, following the above architect. For residual block we have choose to use two convolution layer and disable maxpool2d function and make the number of input channel and output channel same because we have to add the output of the residual block with input to the residual block, other are same.

Activation Function

Commonly used activation functions $f_1, ..., f_k$ in neural networks include sigmoid functions, hyperbolic tangent functions, rectified linear units (ReLU), and leaky ReLU functions.

In our model we used ReLU function:

$$f(x) = max(0.1x, x)$$

A rectified linear unit (ReLU) is an activation function that introduces the property of non-linearity to a deep learning model and solves the vanishing gradients issue. ”It interprets the positive part of its argument.

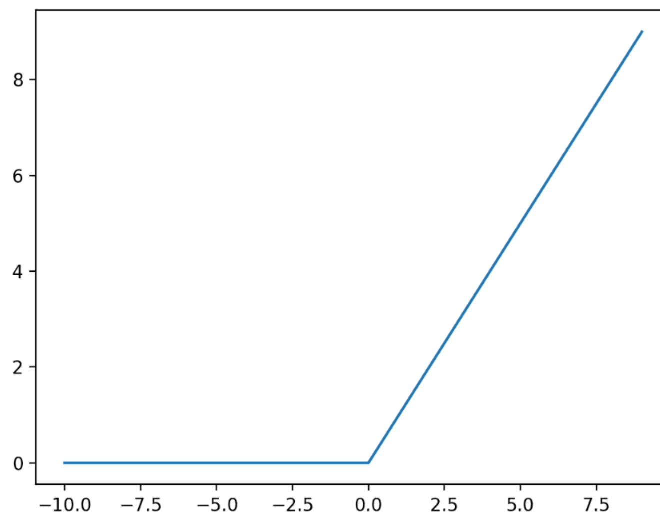


Figure 3.16: Rectified Linear Units(ReLU)

Training Neural Networks

Neural networks are trained using gradient-based numerical optimization methods. Once a loss function L is chosen, the gradients L must be computed for each parameter. Since neural networks have numerous parameters, this gradient computation needs to be done efficiently. Backpropagation is the algorithm used for computing gradients, leveraging the chain rule of differentiation and the layered structure of the network.

To prevent overfitting, it’s crucial to divide the dataset into training and test sets and use regularization techniques. Dropout is another method for avoiding overfitting, where connections in the network are randomly removed during training, encouraging the network to learn more robust representations. Additionally, having an extensive dataset can help mitigate overfitting.

3.2.3 Optimization

The optimization algorithm used is Adam, instantiated with a specific learning rate (LEARNING RATE = 0.001).

Hyperparameters related to optimization include the learning rate (0.001) used by the Adam optimizer.

3.2.4 Training

The training process is configured to run for a specific number of epochs (EPOCHS = 30). The training data is fed to the model using fit generator, which includes hyperparameters like the number of steps per epoch and the validation steps. The loss function used during training is categorical cross entropy. The evaluation metric used during training is accuracy. Hyperparameters related to training include the number of epochs (30), the number of steps per epoch, and the validation steps.

3.3 CoppeliaSim: A robot simulator

CoppeliaSim, formerly known as V-REP, is a popular robotic simulator utilized across industry, academia, and research domains. Its integrated development environment allows for individual control of objects or models via various methods such as embedded scripts, plugins, ROS / ROS2 nodes, remote API clients, or customized solutions. This versatility makes CoppeliaSim well-suited for multi-robot applications. Additionally, controllers can be programmed using languages like C/C++, Python, Java, Lua, MATLAB, or Octave.

Object Properties

In CoppeliaSim objects are of three heading and they are:

- Dynamic object: Position and orientation are affected by gravity, collisions or other constraints (i.e. joints)
- Static object: Position and orientation are not affected by forces
- Respondable object: It produce a collision reaction with other responsible object when colliding

And these three heading produce four types of objects:

1. Non respondable dynamic object
2. Respondable dynamic object

3. Non respondable static object

4. Respondable dynamic object

	Static	Non-static
Non-respondable	☒	☐
Respondable	☒	☐

Figure 3.17: Object Properties

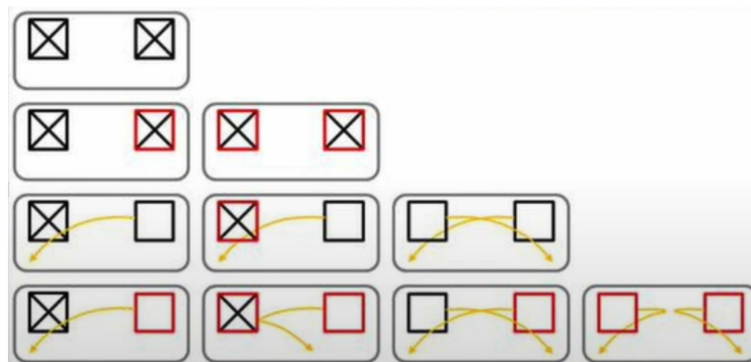


Figure 3.18: Object responses

The properties of an objects are:

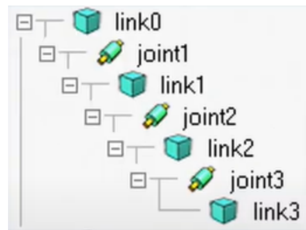
- **Collidable:** The object will be tested for collision against other collidable objects. This does not imply that the reaction will occur as other object must be responsible for reaction.

- **Detectable:** The object will be detected by proximity sensor (sonar, laser infrared, etc.)

- **Renderable:** The object will be seen by the vision sensor (processing video cameras)

- **Measurable:** The object will be used for minimum distance calculation

And finally, we have shape property for an object that include for its color, texture and geometry. Object in CoppeliaSim has a hierarchy system. That means that there are parent object and child object, this helps us to control a system made out of several object to work together as a system like to perform translate or rotation operation as operations in top-hierarchy affects the bottom-hierarchy object.



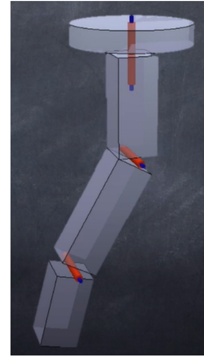
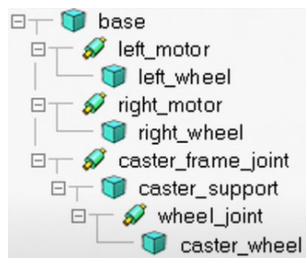


Figure 3.19: Serial joint hierarchy in CoppeliaSim



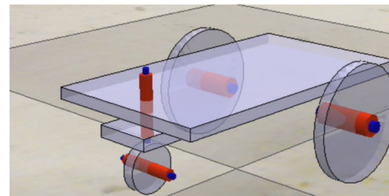


Figure 3.20: Multi joint hierarchy in CoppeliaSim

3.3.1 Joints

Joints allow movement between parent and child objects. The joint child object will move based on the joint position (angular or linear). Joints can have limits where their position cannot exceed. For instance, a robot arm has limited positions, while a mobile robot wheel joint can rotate unlimited. There are three types of joints in CoppeliaSim and they are:

- Passive: Here the joint position cannot directly be controlled. It can be modified through code script (API)
- Torque/ Force: Joints position is controlled by the physics engine. It can be actuated or not and the reference value can be modified through code script (API)
- Inverse Kinematics: Joints is passive and controlled by the inverse kinematics module. Hybrid mode allows you to move dynamic objects
- Dependent: joint's position depends on the position of another joint. It can move dynamic object using the hybrid mode.

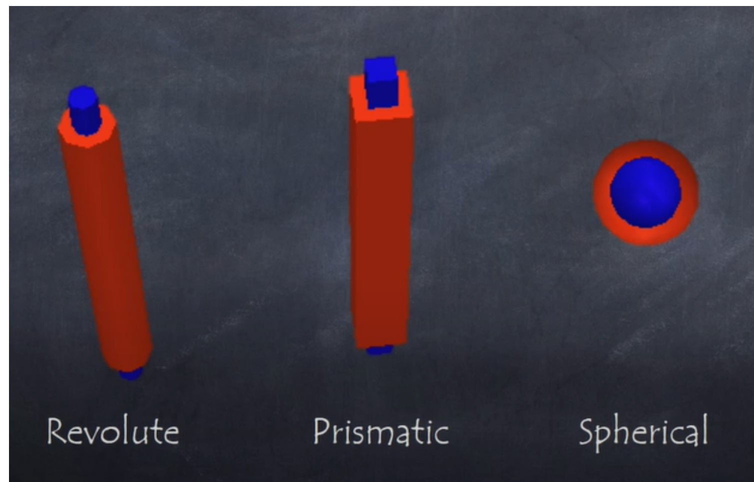


Figure 3.21: Joint type in CoppeliaSim

3.3.2 Sensor

Proximity sensor

It is used to measure the minimum distance to objects with detectable property enable. It includes sensor such as ultrasound, infrared, laser etc. there are six types of detection volumes.

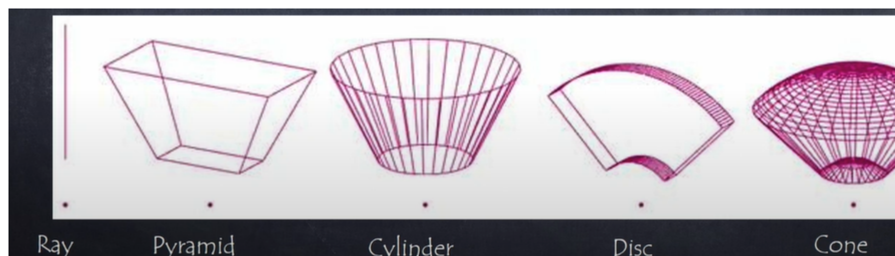


Figure 3.22: Proximity sensor in CoppeliaSim

Vision Sensor

It simulates camera that can be processed and obtains images of renderable object There are two type of vision sensors orthographic (rectangular) and perspective (trapezoidal).

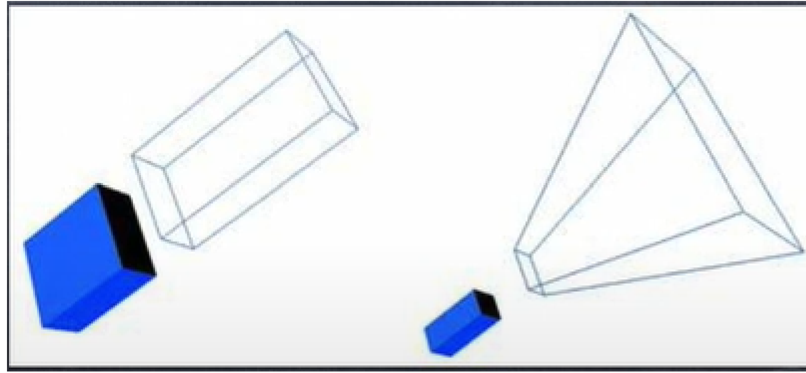


Figure 3.23: Vision sensor in CoppeliaSim

For our project, we initially chose the Pioneer P3DX robot available as a pre-built option in CoppeliaSim. We placed the robot in the scene area by dragging and dropping it from the model selection. Next, we created boundaries around the robot by adding cuboids. We adjusted the dimensions and positions of these cuboids to form the desired boundary shape. We ensured that the cuboids were fixed and responsive by configuring their dynamic properties in the scene object properties dialog.

To equip the robot with sensors, we added a vision sensor and a proximity sensor. For the vision sensor, we selected a perceptive type from the Add menu and placed it under the Pioneer P3DX robot. We adjusted its position and orientation to ensure a clear path for vision sensing. Similarly, we added a proximity sensor and configured its detectable range in the volume parameter settings.

To set up the Python API for CoppeliaSim, we downloaded the API program file and added a script to the scene floor. In the script's sysCall_init() function, we included commands to initialize the API and start communication on a specific port.

With these preparations complete, our simulation environment was ready for testing and development.

The robot was simulated in the CoppeliaSim with the following python script:

```
1  import sim
2  from time import sleep as delay
3  import numpy as np
4  import cv2
5  import torch as T
6  import torch.nn as nn
7  import sys
8  from PIL import Image
9  import torchvision.transforms as tt
```

```
10
11  print('Program started')
12  sim.simxFinish(-1)
13  clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
14
15  lSpeed = 0
16  rSpeed = 0
17  if clientID != -1:
18      print('Connected to remote API server')
19  else:
20      sys.exit('Failed connecting to remote API server')
21  delay(1)
22
23  errorCode, left_motor_handle = sim.simxGetObjectHandle(
24      clientID, "/PioneerP3DX/leftMotor", sim.simx_opmode_oneshot_wait)
25  errorCode, right_motor_handle = sim.simxGetObjectHandle(
26      clientID, "/PioneerP3DX/rightMotor", sim.simx_opmode_oneshot_wait)
27  errorCode, camera_handle = sim.simxGetObjectHandle(
28      clientID, '/PioneerP3DX/Vision_sensor',
     ↪  sim.simx_opmode_oneshot_wait)
29  delay(1)
30
31  returnCode, resolution, image = sim.simxGetVisionSensorImage(
32      clientID, camera_handle, 0, sim.simx_opmode_streaming)
33  delay(1)
34
35
36  def conv_block(in_channels, out_channels, pool=False):
37      layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3,
     ↪  padding=1),
38                nn.BatchNorm2d(out_channels),
39                nn.ReLU(inplace=True)]
40      if pool: layers.append(nn.MaxPool2d(2))
41      return nn.Sequential(*layers)
42
43
44  class Resnet(nn.Module):
45      def __init__(self):
46          super().__init__()
47          self.conv1 = conv_block(3, 8, pool=True)
48          self.res1 = nn.Sequential(conv_block(8, 8), conv_block(8, 8))
49          self.conv2 = conv_block(8, 16, pool=True)
50          self.conv3 = conv_block(16, 32, pool=True)
51          self.res2 = nn.Sequential(conv_block(32, 32), conv_block(32,
         ↪  32))
52          self.conv5 = conv_block(32, 64, pool=True)
53          self.classifier = nn.Sequential(nn.MaxPool2d(2),
54                                          nn.Flatten(),
```

```python
                                          nn.Dropout(0.2))
        self.linear = nn.Sequential(
            nn.Linear(576, 128),
            nn.ReLU(),
            nn.Linear(128, 4)
        )

    def forward(self, xb):
            out = self.conv1(xb)
            out = self.res1(out) + out
            out = self.conv2(out)
            out = self.conv3(out)
            out = self.res2(out) + out
            out = self.conv5(out)
            out = self.classifier(out)
            out = self.linear(out)
            return out


def index(val):
    maximum = 0
    for i in range(len(val)):
        if i != 0:
            if val[i] > val[maximum]:
                maximum = i
    return maximum


def tensor_to_list(tensor):
    if isinstance(tensor, T.Tensor):
        tensor = tensor.tolist()
    if isinstance(tensor, list):
        return [tensor_to_list(t) for t in tensor]
    else:
        return tensor


def raspido(clientID, left_motor_handle, right_motor_handle, action):
    if action == 0:
        lspeed, rspeed = 0.2, 0.2
    elif action == 1:
        lspeed, rspeed = -0.2, -0.2
    elif action == 2:
        lspeed, rspeed = 0, 0.2
    elif action == 3:
        lspeed, rspeed = 0.2, 0
    else:
        lspeed, rspeed = 0, 0
```

```
103        errorCode = sim.simxSetJointTargetVelocity(
104                clientID, left_motor_handle, lspeed,
                ↪  sim.simx_opmode_streaming)
105        errorCode = sim.simxSetJointTargetVelocity(
106                clientID, right_motor_handle, rspeed,
                ↪  sim.simx_opmode_streaming)
107
108    def file_retriever(clientID, camera_handle):
109        returnCode, resolution, image = sim.simxGetVisionSensorImage(
110                clientID, camera_handle, 0, sim.simx_opmode_buffer)
111        img = np.array(image).astype(np.uint8)
112        img.resize([resolution[0], resolution[1], 3])
113        img = cv2.flip(img, 0)
114        img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)
115        img = cv2.resize(img, (100, 100))
116        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
117        img = Image.fromarray(img)
118        img = imgtotensor(img)
119        return img.unsqueeze(0)
120
121
122    model = Resnet()
123    model.load_state_dict(T.load(r'D:\new\modelv3.pth'))
124    model.eval()
125    imgtotensor = tt.ToTensor()
126    while True:
127            im = file_retriever(clientID, camera_handle)
128            errorCode = sim.simxSetJointTargetVelocity(
129                clientID, left_motor_handle, lSpeed,
                ↪  sim.simx_opmode_streaming)
130            errorCode = sim.simxSetJointTargetVelocity(
131                clientID, right_motor_handle, rSpeed,
                ↪  sim.simx_opmode_streaming)
132            action = model(im)
133            action = tensor_to_list(action)
134            action = index(action[0])
135            print(action)
136            raspido(clientID, left_motor_handle, right_motor_handle,
            ↪  action)
```

This Python script is designed to control a simulated robot using the CoppeliaSim robotics simulation software. It can be broken down as:

Imports:

The script imports necessary libraries including sim (presumably the CoppeliaSim library), numpy, cv2 (OpenCV), torch (PyTorch), sys, and PIL. These libraries are

used for various purposes including simulation control, image processing, and deep learning.

Print Statements: It prints ”Program started” to indicate the start of the program.

Connection to CoppeliaSim:

The script attempts to connect to the CoppeliaSim remote API server using simxStart function. If the connection is successful, it prints ”Connected to remote API server”.

Motor and Camera Handles:

It retrieves handles for the left motor, right motor, and vision sensor (camera) from the simulation environment using simxGetObjectHandle. These handles are used to control the robot’s motors and obtain images from the camera.

Neural Network Definition:

The script defines a neural network class called Resnet. This class defines the architecture of a convolutional neural network (CNN) model using PyTorch’s nn.Module. The network consists of convolutional layers, batch normalization, ReLU activation functions, max-pooling layers, dropout layers, and fully connected layers. This CNN model is used for image processing tasks.

Utility Functions:

The script defines several utility functions including conv_block for creating convolutional blocks, index for finding the index of the maximum value in a list, tensor_to_list for converting PyTorch tensors to Python lists, raspido for controlling the robot’s motors based on actions, and file_retriever for retrieving and preprocessing images from the camera.

Model Loading:

It loads a pre-trained neural network model from a file (modelv3.pth) using torch.load and assigns the model to the model variable. The model is then set to evaluation mode using model.eval().

Main Loop:

The script enters a main loop where it continuously retrieves images from the camera, processes them using the loaded neural network model, determines actions to

control the robot, and applies those actions to the robot's motors using the raspido function. Inside the loop, it retrieves an image from the camera using file_retriever.

- It sets the target velocities of the robot's motors to lSpeed and rSpeed using simxSetJointTargetVelocity.
- It passes the image through the neural network model to obtain action predictions.
- It selects the action with the highest predicted probability.
- It prints the selected action.
- It applies the selected action to the robot's motors using the raspido function.

3.4 Object detection model

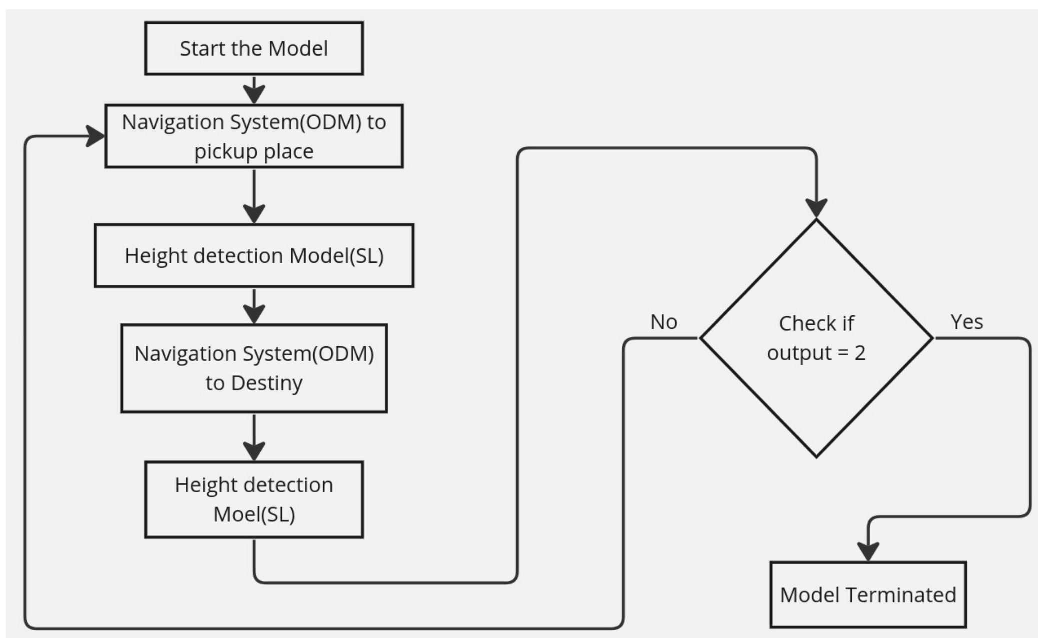


Figure 3.24: Object dection model planning

Object detection is a crucial task in computer vision, involving the identification and localization of objects within images or videos. Image Localization precisely determines the location of objects using bounding boxes, which outline the objects with rectangular shapes. Meanwhile, image classification assigns a class label to an entire image or specific objects within it.

YOLO (You Only Look Once) is a widely-used algorithm for object detection. It stands out for its speed, accuracy, generalization capabilities, and open-source nature.

YOLO architecture is similar to GoogleNet. As illustrated in figure 3.25 , it has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.

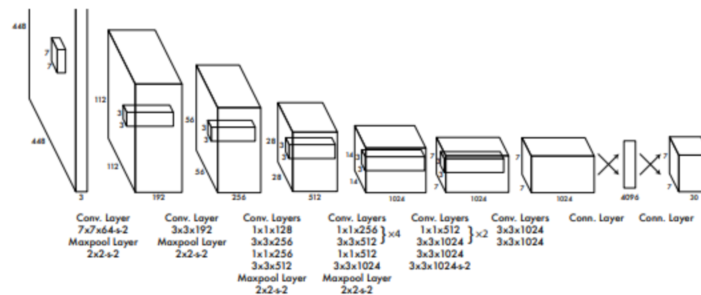


Figure 3.25: Object detection Network Architecture(R. Joseph, et al)

The architecture works as follows:

- Resizes the input image into 448x448 before going through the convolutional network.
- A 1x1 convolution is first applied to reduce the number of channels, which is then followed by a 3x3 convolution to generate a cuboidal output.
- The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function or no activation.
- Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting.

The algorithm works based on the following four approaches:

- Residual blocks
- Bounding box regression
- Intersection Over Unions or IOU
- Non-Maximum Suppression

Residual blocks

In this approach, the original image is partitioned into an $n \times n$ grid of uniform cells. Each cell in the grid assumes the responsibility of localizing and classifying the object it encompasses, providing predictions for both the object class and the associated probability or confidence score.

Bounding box regression

The subsequent task involves identifying bounding boxes, which are essentially rectangles outlining all objects present in the image. Multiple bounding boxes can exist, corresponding to the objects within the image.

YOLO achieves this by employing a single regression module, which outputs the attributes of these bounding boxes in a specific format. The final vector representation for each bounding box, denoted as Y, follows a predefined format.

$$Y = [pc, bx, by, bh, bw, c1, c2]$$

This is especially important during the training phase of the model where,

- pc corresponds to the probability score of the grid containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is the simplified version since the probability of each yellow cell is zero (insignificant).
- bx, by are the x and y coordinates of the center of the bounding box with respect to the enveloping grid cell.
- bh, bw correspond to the height and the width of the bounding box with respect to the enveloping grid cell.
- $c1$ and $c2$ correspond to the two classes Player and Ball. We can have as many classes as your use case requires.

Intersection over union

In many cases, a single object in an image may have multiple grid box candidates for prediction, even though not all of them are relevant. The Intersection over Union (IOU), a value ranging between 0 and 1, is utilized to filter out such irrelevant grid boxes and retain only those that are pertinent.

Here's how the process works:

- The user specifies an IOU selection threshold, which could, for example, be set to 0.5.
- YOLO computes the IOU for each grid cell, calculated as the Intersection area divided by the Union area.
- Subsequently, it disregards the predictions of grid cells with an IOU value less than or equal to the threshold, while retaining those with an IOU value greater than the threshold.

Non-Max Suppression or NMS

Establishing a threshold for the Intersection over Union (IOU) isn't always sufficient because an object might have several boxes with an IOU exceeding the threshold. Retaining all these boxes could introduce noise into the predictions. This is where Non-Maximum Suppression (NMS) comes into play, enabling the selection of only the boxes with the highest probability scores for detection.

3.4.1 How the navigation works

Firstly upon starting, the camera position is adjusted using the servo motor. then the camera take picture and check if it detect anything. if it did not detect any thing it will then turn right take picture and analyze again. it will do it until it detect the pick up place. once it detect the pickup place it will then see if it is *ps*, *ph* or *pa*. if it is *ps*, it will go straight and check if it is still seeing *ps*. if it see *ph* it will take left turn or right turn depending upon where it detect the *ph* . the size of our image is 400*300 so is *ph* top-left x-coordinates is less than 10 we take right otherwise we take left turn. in case of pa we make right, straight and left again and track the last instance of the detected object by repeating last action as we do not want to lose track of object due to our position correcting maneuver. this all continues until we get a distance of less than $18cm$.here at this place if our model chose other than *ps*, we will have another positioning correcting maneuver. and if it is *ps* we use small forward step until we get an appropriate distance of 15cm. after this the navigation model stops and the camera is lower and picture is taken and send to height detection model, the model output either 0,1 and 2. depending upon the correct height we need the mast to be for our operation. if the model input 2 then then mast is send up by 900 by our steeper model and if its zero the mast is send 900 down and then the mobile robot goes 2 steps forward. and then if it is picking operation it goes 200 up again, and for loading it goes 200 down again. after this it take 3 step backward. and adjust the position of the mast to its centre position. after this load the same navigation model but this time for placement position, here the maneuver steps are reversed in order for left and right other things are same. After reaching its placement position *ot* then again start height detecting model and the thing continue until all the boxes are moved from pickup place to the placement place.

Chapter 4

Results and Discussion

In this project, we employed supervised learning techniques to develop a robot navigation system within an indoor environment. Our aim was to enable the robot to autonomously navigate from a starting point to a goal location while picking up boxes characterized by colour and place it to the desired location. We did this in both the simulation environment and the real life environment. For our simulation environment we made an Sl model whose goal was to go from starting place to pickup place and from pickup place to the goal place and return to pickup place. While in our actual physical model we used Object detection model for the same work but now it also pick up the boxes from pickup place and place those boxes in their goal place. So to achieve our goal we used object detection model based on YOLO v8 for navigating to the pickup place and for navigating from pickup place to placement place and then developed image classifier model for height detecting to adjust the level of fork while picking up boxes. Our model was successfully able to navigate to the desired location and pickup the boxes and transfer and place it to the desired placement place. In this chapter we present the effectiveness of our model characterized by different parameter as shown below:

4.1 CoppeliaSim simulation

The robot was trained in the simulated environment of CoppeliaSim where we used the supervised learning model designed on our own and the following result was achieved.

4.1.1 Training loss vs Epoch

In supervised learning, the concept of total loss versus epoch refers to the trend of the overall loss function over the course of training epochs. The total loss typically

encompasses both the training loss and the validation loss, providing a comprehensive measure of the model's performance at each epoch.

During the training process, the model's parameters are iteratively updated to minimize the loss function, which quantifies the disparity between the model's predictions and the ground truth labels. As training progresses through multiple epochs, the total loss is evaluated at regular intervals (epochs), reflecting the model's performance at each stage of training.

The total loss versus epoch plot serves as a valuable tool for assessing the training progress and diagnosing potential issues. The total loss versus epoch plot of our model is illustrated by the figure 4.1.

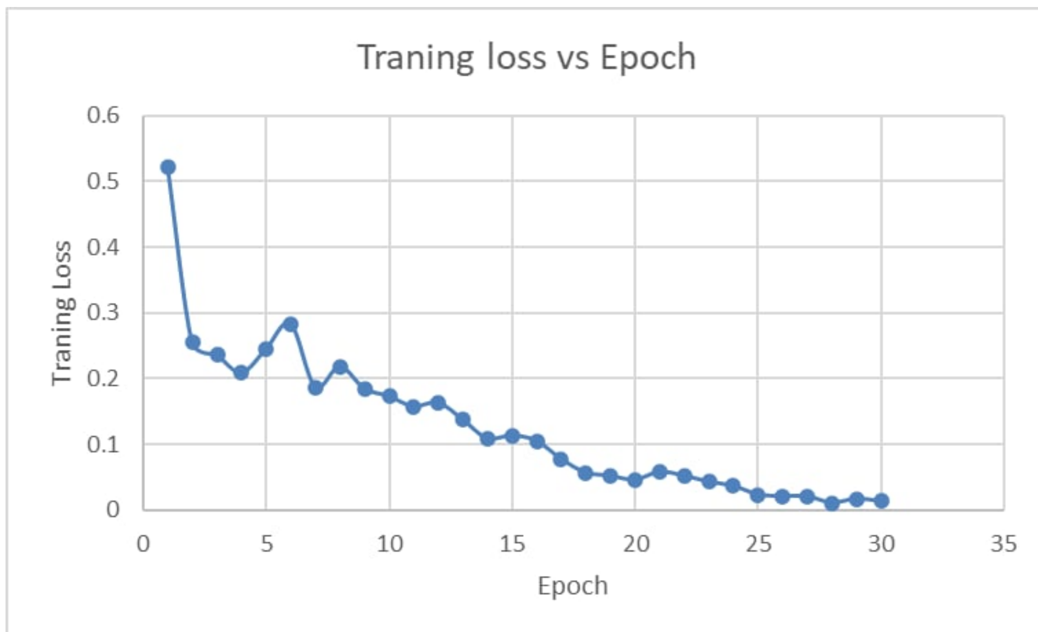


Figure 4.1: Training loss vs Epoch

From the above graph we can see that the total loss is decrease over epochs, indicating that the model is learning and improving its performance on both the training and validation sets. This suggests that the model is effectively capturing patterns in the data and generalizing well.

4.1.2 Validation Loss vs Epoch

In supervised learning, validation loss refers to the error or loss calculated on a separate dataset called the validation set. This set is distinct from the training set and is used to evaluate the performance of the model during training.

Validation loss serves as a crucial metric for monitoring the generalization capability of the model. While training loss measures the error incurred during the optimization process on the training data, validation loss provides insight into how

well the model generalizes to unseen data.

During the training process, the model's parameters are updated iteratively to minimize the training loss. However, if the model overfits the training data, meaning it learns to memorize the training examples rather than generalize, it may perform poorly on new, unseen data. Validation loss helps detect overfitting by indicating whether the model's performance on the validation set improves or deteriorates over time.

Ideally, validation loss should decrease steadily or remain relatively stable throughout training. A consistent increase in validation loss suggests that the model is overfitting and failing to generalize well. In such cases, adjustments may be necessary, such as modifying the model architecture, introducing regularization techniques, or adjusting hyperparameters. The change in validation loss during training can be visualized using the graph in figure 4.2.

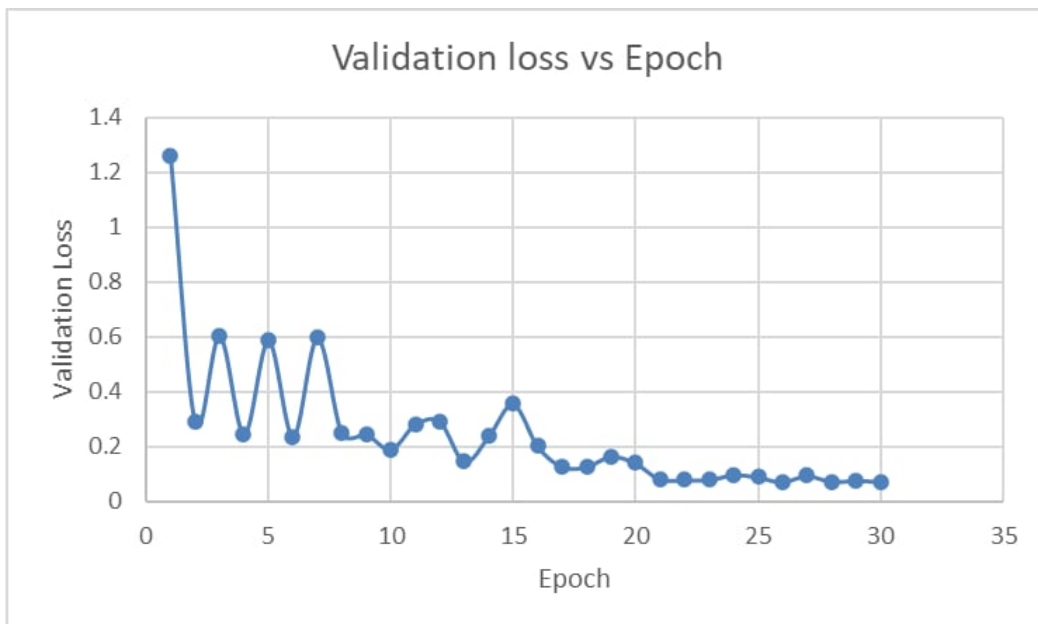


Figure 4.2: Validation loss vs Epoch

4.1.3 Accuracy of model vs Epoch

In supervised learning, the accuracy of the model versus epoch refers to the trend of the model's classification accuracy over the course of training epochs. This metric provides insights into how the model's performance evolves during the training process.

During training, the model's parameters are iteratively updated to minimize the loss function, which measures the disparity between the model's predictions and the ground truth labels. As training progresses through multiple epochs, the model's accuracy on both the training and validation sets is evaluated at regular intervals

(epochs). Ideally, the model's accuracy should improve over epochs, indicating that the model is learning and making better predictions.

The accuracy versus epoch plot is a valuable diagnostic tool for assessing the training progress. The variation of accuracy of model with respect to Epoch can be visualized by following graph as illustrated in figure 4.3.

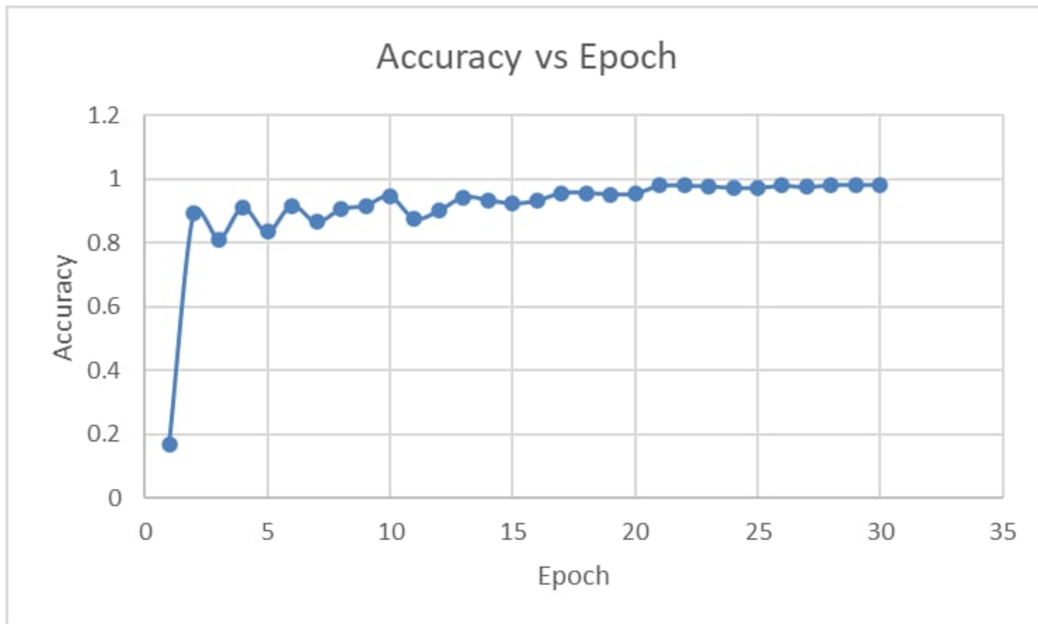


Figure 4.3: Accuracy vs Epoch

As we can from the figure 4.3, the model's accuracy improves over epochs, indicating that the model is learning and making better predictions.

After reaching a certain points nearly after 20 epochs, the accuracy is somehow constant, where further training epochs do not lead to significant improvements in performance. This indicates that the model has converged to a good solution.

The total accuracy of the model after 30 Epochs during simulation was found to be 98%.

4.1.4 Navigation in CoppeliaSim environment

Through extensive experimentation and analysis, we observed consistent and satisfactory performance in the robot's ability to navigate from designated starting points to predefined target locations.

As we can see in the figure 4.4, when we place our robot at any point inside the environment, it first navigates to the blue box and then to the red box and back to blue box. Across multiple simulation runs, the robot consistently followed a predetermined sequence of actions, moving from a starting position to a blue box, then to a red box, and returning to the blue box. This sequential navigation

behavior, indicative of a well-learned strategy, was observed to be both reliable and repeatable, with the robot achieving the desired navigation objectives in a manner consistent with its training.

The trained model demonstrated a remarkable capacity to learn and generalize its navigation behavior based on the provided training data. By leveraging a supervised learning framework, the robot successfully learned to associate sensory inputs from the environment with appropriate actions, effectively mapping visual cues to navigation decisions.

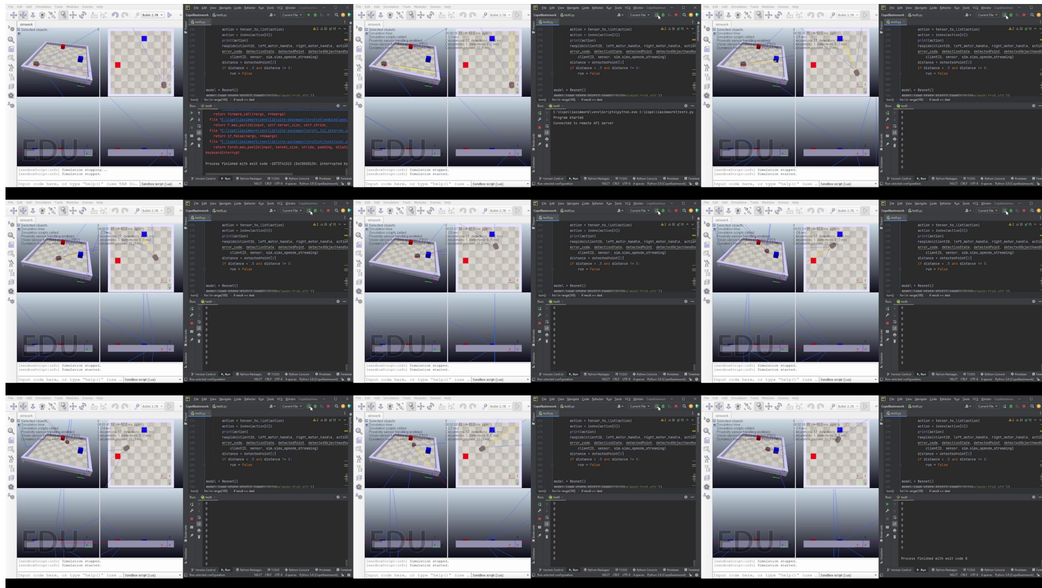


Figure 4.4: Navigation in CoppeliaSim environment

4.2 Physical Model

In our actual robot, we used object detection model based on YOLO v8 for navigation of our robot to the pickup and placement place whereas we used our own image classifier model for height detecting and loading model. The results for the both model can be explained as follows:

4.2.1 Confusion matrix for object detection model

For our project, we opted for a medium-sized model trained over 100 epochs. The model comprised 295 layers, with 25,859,794 parameters, and was designed to classify objects into six classes: ps, ph, pa, ds, dh, and da, resulting in a final accuracy of 73%. The confusion matrix, depicted in Figure 4.4, illustrates the model's performance.

In the confusion matrix, the horizontal axis represents instances of objects present in the model, while the vertical axis denotes the model's predictions. Our

model categorizes objects into different size classes, such as straight forward (p) and at an angle (d), with subclasses representing pickup (p) and placement (d) locations. Correct identifications align along the same axis, misclassifications appear along different axes, and missed detections are marked along the background axis.

To calculate the model's accuracy, we sum the numbers along the diagonal and divide by the total sum of all numbers in the matrix. 4.5.

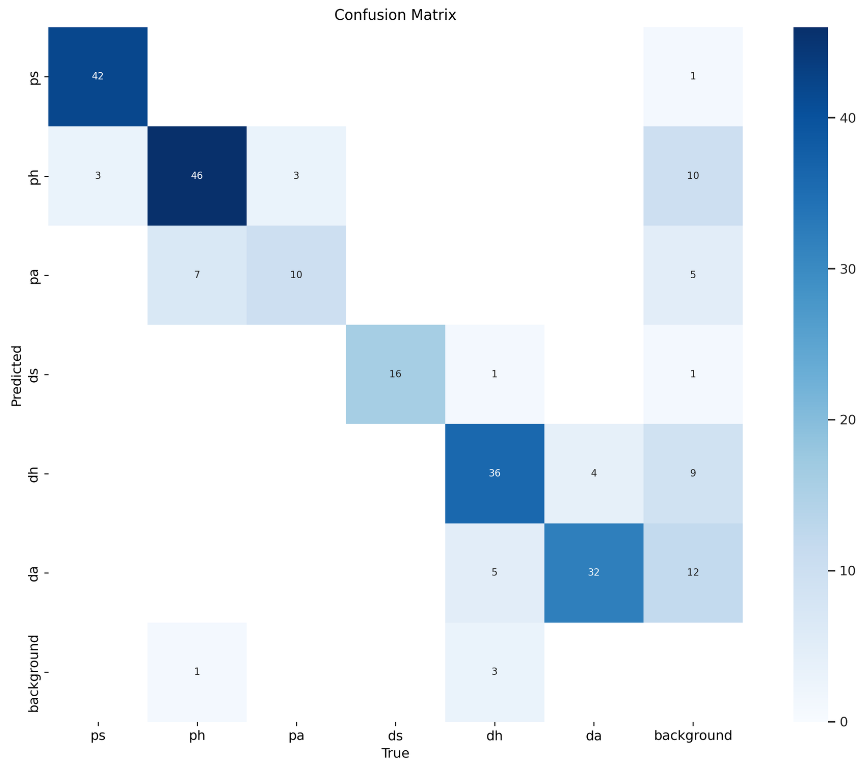


Figure 4.5: Confusion matrix

4.2.2 Height loading model

An image classifier was used in height detection model which was used to adjust the position of the fork to the required position according to the level of the load to be lifted and transferred. The different parameters representing the improvement of of the model over different Epochs can be explained by following plots.

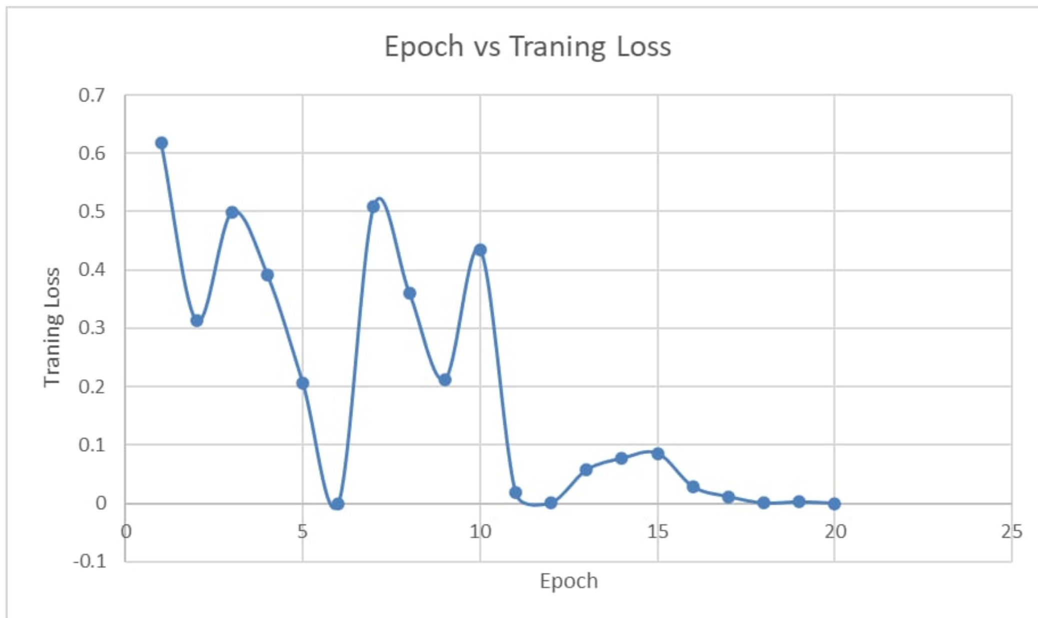


Figure 4.6: Training loss vs Epoch (Height loading model)

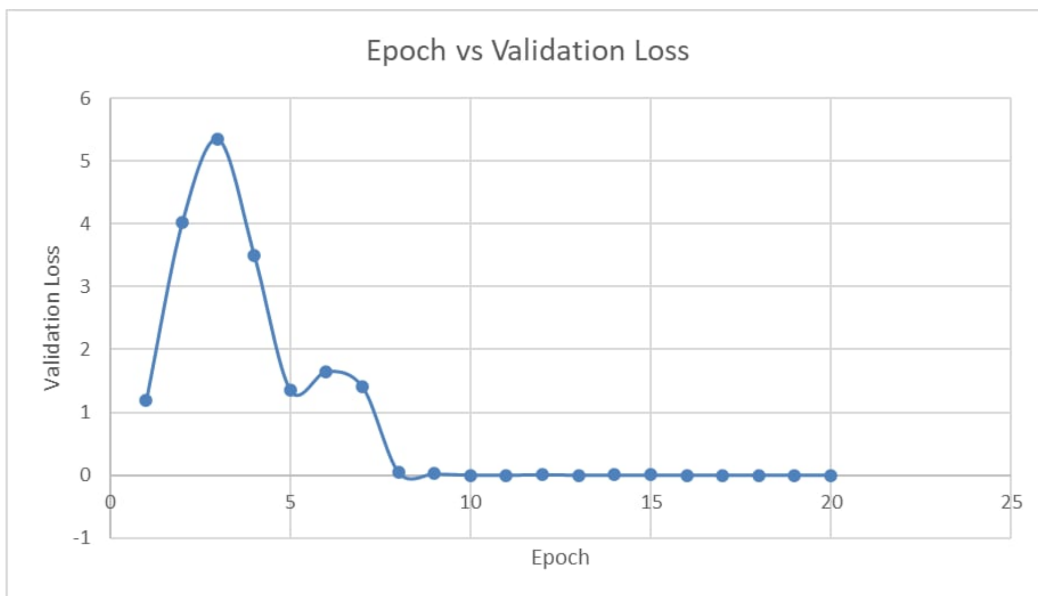


Figure 4.7: Validation vs Epoch (Height loading model)

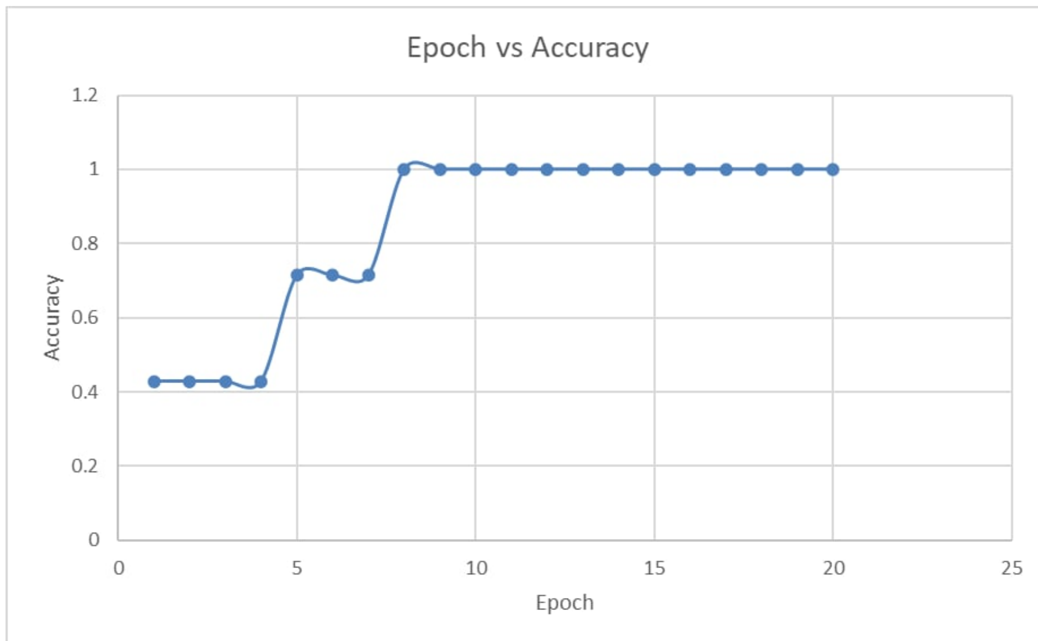


Figure 4.8: Accuracy vs Epoch (Height loading model)

4.2.3 Actual robot working

The forklift, a common piece of material handling equipment, plays a vital role in warehouses, distribution centers, and manufacturing facilities worldwide. This specialized vehicle, typically characterized by its two-pronged forks at the front, is designed to lift, move, and stack heavy loads with efficiency and precision.

Here our main objective was to design a scaled down forklift type robot that is capable of working on its own through the deployment of machine learning techniques by effectively mapping visual cues to navigation decisions.

The robot, characterized by its two-pronged forks at the front, was tasked with transferring three blue boxes stacked in a column to a location in front of a yellow platform. Leveraging machine learning algorithms, the robot autonomously identified and navigated to the target location, demonstrating its capability to interpret visual cues and make informed navigation decisions.

In Figure 4.9, we provide a frame-by-frame depiction of the robot's operation, showcasing its successful execution of the assigned task. The robot approached the stacked boxes, accurately positioned its forks beneath the load, and lifted them with precision. It then traversed the environment to the designated location, where it deposited the boxes in front of the yellow platform as specified.

The completion of this task underscores the effectiveness of our approach in designing and training a machine learning-based navigation system for autonomous robotic material handling. The successful performance of the robot in a simulated environment highlights its potential for real-world applications in warehouses, dis-

tribution centers, and manufacturing facilities. Furthermore, this project lays a foundation for future research and development efforts aimed at enhancing the capabilities and scalability of autonomous material handling systems.

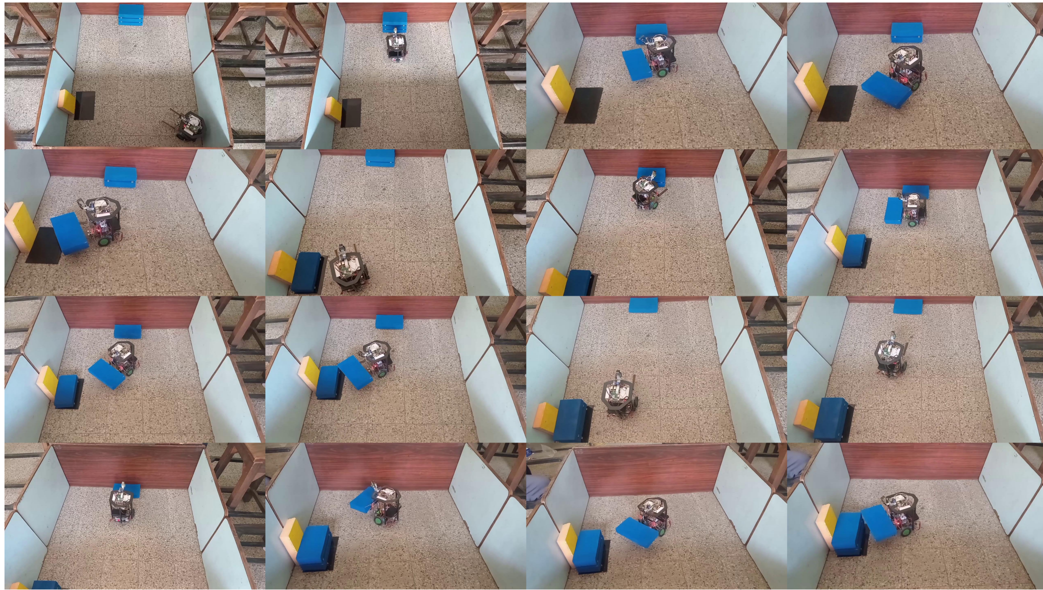


Figure 4.9: Actual robot working

Chapter 5

Conclusion

In the realm of autonomous navigation for inventory transportation, the utilization of various techniques such as Simultaneous Localization and Mapping (SLAM), model predictive control, and others has significantly advanced the capabilities of robots. These techniques have enabled robots to navigate through complex environments, avoiding obstacles and efficiently transporting inventory. However, as industries seek to further automate their processes, there's a growing interest in leveraging machine learning (ML) algorithms to enhance the capabilities of these robots.

One of the primary advantages of employing ML algorithms in autonomous navigation is their ability to adapt and improve over time. Traditional techniques often rely on predefined models and algorithms, which may struggle to handle unexpected situations or dynamic environments. In contrast, ML algorithms can analyze vast amounts of data to learn patterns and make decisions based on the observed trends. This adaptability enables robots to navigate more effectively in real-world scenarios, where conditions may vary unpredictably.

Moreover, ML algorithms can optimize navigation strategies based on specific objectives or constraints. By learning from past experiences and outcomes, robots can continuously refine their navigation behaviors to achieve better performance, whether it's minimizing travel time, avoiding collisions, or optimizing energy consumption. This adaptability and optimization capability make ML algorithms particularly well-suited for inventory transportation tasks, where efficiency and reliability are paramount.

Furthermore, ML algorithms have the potential to enhance safety in autonomous navigation. By continuously learning from data, robots can improve their ability to detect and respond to potential hazards, thus reducing the risk of accidents or damage to both the inventory and the environment.

In essence, while traditional techniques have laid the foundation for autonomous navigation, the integration of ML algorithms offers a transformative approach that

unlocks new levels of efficiency, adaptability, and safety. As industries continue to embrace automation, leveraging ML in inventory transportation robots holds the promise of revolutionizing logistics operations, leading to increased productivity, cost savings, and ultimately, a more streamlined and efficient supply chain.

In this comprehensive report, we embarked on a journey to automate the task of a simple forklift like robot in an indoor environment with a simple RGB camera and a cheap ultrasonic distance sensor using machine learning algorithm. Our endeavor encompassed various stages, from dataset collection and model development to the creation of a user-friendly application aimed at aiding various indoor activities of inventory transportion. The foundation of our project lay in the meticulous collection of a diverse dataset comprising images of indoor environment , alongside a category for the object to pick and the placement place. This step was pivotal, as it provided the groundwork necessary for training robust machine learning models capable of accurate navigation and pick up.

In conclusion, our project stands as a testament to the potential of interdisciplinary collaboration and innovation in driving positive change within the inventory transportation department. With a steadfast commitment to excellence and a vision for a more resilient and productive landscape, we remain dedicated to furthering the impact and reach of our endeavors in the pursuit of a sustainable future.

Chapter 6

Limitations and future enhancement

Limitations

- The current model was trained and deployed in a simple environment, which may not adequately prepare it for more dynamic and complex real-world scenarios. As a result, the model's performance may degrade when faced with unpredictable environments or obstacles.

- The utilization of a Raspberry Pi 3 B+ for image processing resulted in sub optimal performance, causing delays in real-time processing. This hardware limitation hindered the robot's ability to navigate smoothly, impacting its overall efficiency and responsiveness.

- The dataset used for training may lack diversity, potentially leading to poor generalization in varied environments. Incorporating a more diverse range of training data, including different lighting conditions, object orientations, and environmental contexts, could improve the model's robustness and adaptability.

- Due to the limitations in hardware and environment complexity, the current system may require manual intervention or human oversight to address unforeseen challenges or errors encountered during navigation. This reduces the system's autonomy and limits its practical applications in real-world settings.

Future Enhancement

- Incorporating simulation environments that mimic real-world dynamics and complexities can facilitate more comprehensive training of the model. This allows for testing and validation in diverse scenarios, including varying terrain, lighting conditions, and obstacle configurations.

- Integrating additional sensors, such as depth cameras, can augment the model's perception capabilities and provide more comprehensive environmental awareness. Sensor fusion techniques can combine information from multiple sensors to enhance object detection, localization, and navigation accuracy.

- Implementing online learning algorithms allows the model to continuously adapt and improve its performance based on real-time feedback from the environment. This enables the system to learn from experience and dynamically adjust its behavior in response to changing conditions or new challenges.

- Exploring collaborative robotics approaches, where robots can interact and share information with each other or with human operators, can enhance the system's versatility and scalability. This enables coordinated navigation and task execution in complex, dynamic environments, while also fostering human-robot collaboration and cooperation.

References

1. Desouza, G. N., and A. C. Kak. “Vision for Mobile Robot Navigation: A Survey.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 2, Institute of Electrical and Electronics Engineers (IEEE), 2002, pp. 237–67. Crossref, https://doi.org/10.1109/34.982903.

2. Kolski, Sascha, et al. “Autonomous Driving in Structured and Unstructured Environments - Research Collection.” Autonomous Driving in Structured and Unstructured Environments - Research Collection, 11 Sept. 2006, https://doi.org/10.1109/IVS.2006.1689687.

3. Kosaka, Akio, and Avinash C. Kak. “Fast Vision-guided Mobile Robot Navigation Using Model-based Reasoning and Prediction of Uncertainties.” CVGIP: Image Understanding, vol. 56, no. 3, Elsevier BV, Nov. 1992, pp. 271–329. Crossref,

   https://doi.org/10.1016/1049- 9660(92)90045-5.

4. Md Fauadi, Muhammad Hafidz Fazli, et al. “Intelligent Vision-based Navigation System for Mobile Robot: A Technological Review.” Periodicals of Engineering and Natural Sciences (PEN), vol. 6, no. 2, International University of Sarajevo, Oct. 2018, p. 47. Crossref,

5. Ruan, X., Ren, D., Zhu, X. & Huang, J. (2019) Mobile Robot Navigation Based on Deep Reinforcement Learning. In: 2019 Chinese Control And Decision Conference (CCDC), June 2019. IEEE, pp. 6174–6178.

6. Tai, L., Paolo, G. & Liu, M. (2017) Virtual-to-Real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation. March.

7. Zeng, T. (2018) Learning Continuous Control through Proximal Policy Optimization for Mobile Robot Navigation.

8. Zhu, K. & Zhang, T. (2021) Deep Reinforcement Learning Based Mobile Robot Navigation: A Review. Tsinghua Science and Technology, 26 (5) October, pp. 674–691.

9. Tsai, C.-Y., Nisar, H. & Hu, Y.-C. (2021) Mapless LiDAR Navigation Control of Wheeled Mobile Robots Based on Deep Imitation Learning. IEEE Access, 9, pp. 117527–117541.

10. Xie, L. (2020) Reinforcement Learning Based Mapless Robot Navigation. University of Oxford.

11. Sutton, R. S. & Barto, A. G. (2018) Reinforcement Learning: An Introduction. Cambridge, MA, USA: A Bradford Book.

12. OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L. & Zaremba, W. (2018) Learning Dexterous In-Hand Manipulation. August.

13. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. & Silver, D. (2020) Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. Nature, 588 (7839) December, pp. 604– 609.

14. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M. & Silver, D. (2017) Emergence of Locomotion Behaviours in Rich Environments. July.

15. Xie, L., Wang, S., Markham, A. & Trigoni, N. (2017) Towards Monocular Vision Based Obstacle Avoidance through Deep Reinforcement Learning. June.

16. Tran, M. Q. & Ly, N. Q. (2020) Mobile Robot Planner with Low-Cost Cameras Using Deep Reinforcement Learning. December.

17. Chaffre, T., Moras, J., Chan-Hon-Tong, A. & Marzat, J. (2020) Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation. April.

18. Ma, L., Liu*, Y. & Chen, J. (2019) Using RGB Image as Visual Input for Mapless Robot Navigation. March.

19. M. Baba and K. Ohtani, "A new sensor system for simultaneously detecting the position and incident angle of a light spot," Journal of Optics A: Pure and Applied Optics, vol . 4, no. 6, pp.s391-s399, 2002.

20. A. Ohya, Y. Miyazaki, and S. I. Yuta, "Autonomous navigation of mobile robot based on teaching and playback using trinocular vision," Industrial Electronics Society,

    2001(IECON'01), The 27th Annual Conference of the IEEE (Vol. 1, pp. 398-403), 2001.

21. P. Lébraly, C. Deymier, O. Ait-Aider, E. Royer and M. Dhome, "Flexible extrinsic calibration of non-overlapping cameras using a planar mirror: Ap-

plication to vision-based robotics," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, 2010, pp. 5640- 5647

22. S. Lecorné and A. Weitzenfeld, "Robot navigation using stereo-vision", Retrieved from: http://weitzenfeld.robolat.org/wp-content/uploads/2015/01/OpticFlow.pdf

23. A.J.R. Neves, A.J. Pinho, D.A. Martins and B. Cunha, "An efficient omnidirectional vision system for soccer robots: From calibration to object detection," Mechatronics, vol. 21, no. 2, pp 399-410, 2011.

24. M. Faisal, R. Hedjar, M. Al Sulaiman, K. Al-Mutib, "Fuzzy Logic Navigation and Obstacle Avoidance by a Mobile Robot in an Unknown Dynamic Environment," International Journal of Advanced Robotic Systems, vol. 10, no. 1, pp. 1-7, 2013

25. J. Minguez and L. Montano, "Extending Collision Avoidance Methods to Consider the Vehicle Shape, Kinematics, and Dynamics of a Mobile Robot," IEEE Transactions on Robotics, vol. 25, no.2, pp. 367-381, 2009.

26. Dobrevski, M. & Skočaj, D. (2021) Deep Reinforcement Learning for Map-Less Goal- Driven Robot Navigation. International Journal of Advanced Robotic Systems, 18 (1) January, p. 172988142199262

27. Toan, N. D. & Woo, K. G. (2021a) Mapless Navigation with Deep Reinforcement Learning Based on The Convolutional Proximal Policy Optimization Network. In: 2021 IEEE International Conference on Big Data and Smart Computing (BigComp), January 2021. IEEE, pp. 298–301.

Machine Learning in Robotics, with it's demonstration in Inventory transportation

ORIGINALITY REPORT

18%
SIMILARITY INDEX

PRIMARY SOURCES

#	Source	Type	Words	%
1	en.wikipedia.org	Internet	436 words	2%
2	www.diva-portal.se	Internet	384 words	2%
3	www.datacamp.com	Internet	379 words	2%
4	web.stanford.edu	Internet	274 words	2%
5	mafiadoc.com	Internet	190 words	1%
6	pen.ius.edu.ba	Internet	107 words	1%
7	www.mectrol.com	Internet	94 words	1%
8	aircconline.com	Internet	69 words	< 1%
9	wikimili.com	Internet	65 words	< 1%

	Source	Match
10	fdokumen.id Internet	54 words — < 1%
11	Tengteng Zhang, Hongwei Mo. "Reinforcement learning for robot research: A comprehensive review and open issues", International Journal of Advanced Robotic Systems, 2021 Crossref	51 words — < 1%
12	www.haydonkerk.com Internet	46 words — < 1%
13	jovian.ai Internet	45 words — < 1%
14	wiki.swarma.org Internet	45 words — < 1%
15	www.limsforum.com Internet	43 words — < 1%
16	flyccs.com Internet	41 words — < 1%
17	www.mecapedia.uji.es Internet	41 words — < 1%
18	www.mdpi.com Internet	39 words — < 1%
19	abimbola-wealth.yolasite.com Internet	38 words — < 1%
20	www.coursehero.com Internet	37 words — < 1%

21	Eklas Hossain. "Machine Learning Crash Course for Engineers", Springer Science and Business Media LLC, 2024 Crossref	36 words — < 1%
22	habr.com Internet	32 words — < 1%
23	medium.com Internet	29 words — < 1%
24	www.embedded.com Internet	25 words — < 1%
25	makezilla.com Internet	24 words — < 1%
26	www.ijritcc.org Internet	24 words — < 1%
27	Rabab Benotsmane, Attila Trohak. "From Toddlers to Experts - The Education of Robotics", 2023 24th International Carpathian Control Conference (ICCC), 2023 Crossref	23 words — < 1%
28	huggingface.co Internet	23 words — < 1%
29	surveys.uft.org Internet	21 words — < 1%
30	dspace.univ-ouargla.dz Internet	19 words — < 1%
31	vtechworks.lib.vt.edu Internet	19 words — < 1%