ROS2-based Bin-Picking System: Integrating RGB and Point Cloud Data for Object Detection, Pose Estimation, and Grasp Planning

1st Muskan Suman
Robotics and Mobility Systems IDRP
Indian Institute of Technology Jodhpur
m23irm008@iitj.ac.in

2nd Arashdeep Singh
Robotics and Mobility Systems IDRP
Indian Institute of Technology Jodhpur
m23irm003@iitj.ac.in

***Abstract*—This project presents an integrated robotic bin-picking system that uses advanced computer vision techniques and ROS2 for real-time object detection, pose estimation, and grasp planning. The system utilizes YOLO for detecting objects in RGB images, Point Cloud Library (PCL) for estimating the orientation and position from depth data, and Grasp Pose Detection (GPD) calculate grasp points. These components enable the robot to perform efficient and reliable pick-and-place tasks. The entire process is modularized within ROS2, ensuring smooth communication. MoveIt! is used for motion planning and execution.**

***Index Terms*—RGB-D, Pose Estimation, Object Detection, Grasp Planning, ROS2, Bin-picking**

I. Introduction

Bin-picking is essential and complex task for robot in industry automation scenario. As this task involves object detection,pose estimation and grasp point detection and movement of objects from one place to other. So to consider this problem, We are working on this project. We will use RGBD images for object detection and pose estimation, as RGBD images provide both texture and depth information of objects. From depth information We will work on finding suitable grasping points for robot to hold the object. Then the next task will be to move the object from source to destination. All of this project will be implemented in ROS2 and Python.

A. Background Work

As mentioned earlier, bin picking is an industrial automation problem. A lot of solutions have been proposed to solve this problem. We have studied some literature of that. Let us have a look on some of the approaches:

Object Detection and Pose Estimation Using RGB-D Images The work by Ruotao He, Juan Rojas, and Yisheng Guan (2020) [2] proposes a pipeline for 3D object detection and pose estimation using RGB-D images. Their approach integrates depth data with RGB images to enhance the accuracy of 3D object localization.

ROS-Based Bin Pick-and-Place Systems Wong et al. (2022) [1] proposed a generic bin pick-and-place system using the Robot Operating System (ROS). Their system leverages the flexibility and modularity of ROS to develop a robotic system capable of detecting objects and executing pick-and-place tasks. The use of ROS for controlling the robot and integrating sensors has been widely adopted in the community due to its open-source nature and ease of customization. The paper highlights challenges in achieving fast and accurate grasping, especially for complex or unstructured environments. Their system, while effective, lacks the sophisticated grasp planning.

Fast 6D Pose Estimation from RGB Images E. Muñoz et al. (2020) [3] developed a method for 6D pose estimation from a single RGB image, focusing on texture-less objects. The method introduced is suitable for speed in real-time. The proposed algorithm estimates position and orientation (6D) of objects using only RGD data. This approach struggles with occulusions , in clustered environment.

Learning-Based Grasp Planning François Hélénon et al. (2021) [4] explore learning-based grasp planning, proposing a system that learns to grasp objects by generalizing across various shapes and configurations.This approach is based on deep learning and learns from large dataset to estimate grasp points. As it requires large computation, It may suffer from resource constraints in real systems.

B. Objectives

The proposed solution has four major divisions:

1) Object Detection: The RGB images from the camera will be processed using a pre-trained YOLOv5 model, which detects and localizes objects within cluttered environments. The bounding boxes and class labels will be extracted in this step.

2) Pose Estimation: The Point Cloud Library (PCL) will utilize the depth data from the camera to estimate the 6-DOF poses of the detected objects. Pose estimation will align with the point cloud data of the detected bounding boxes from step 1, ensuring high-precision localization of objects.

3) Grasp Point Detection: A Grasp Pose Detection (GPD) algorithm will calculate suitable grasp points for the objects using both the point cloud and estimated poses.

4) Motion Planning: The detected grasp points will be fed into MoveIt!, which will plan the manipulator's motion path to grasp and move the object to the desired location.

Fig. 1. Robot Picking and Placing Objects [2]

II. Implementation

The system will be developed using ROS2, with each functionality modularized into separate nodes to ensure seamless integration and communication. The key components of the implementation are as follows:

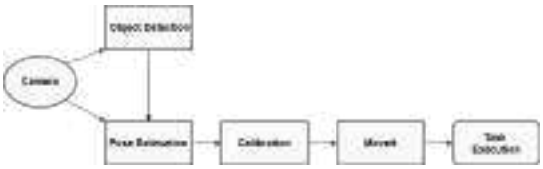


Fig. 2. Achitechture of Implementation

A. Environment Setup

The environment setup plays a pivotal role in the development and testing of the bin-picking system. By creating a virtual simulation environment in Gazebo, the project mitigates the need for physical hardware during the initial development phase.

- **Robot Model:** The simulation integrates robotic arms like MyCobot, configured using URDF and Xacro files. These define the arm's structure, kinematics, and joint properties.
- **Bin-Picking Setup:** The environment includes a bin filled with objects of varying shapes and textures. These are added as 3D models to test detection, pose estimation, and grasp planning algorithms.

B. Camera

The camera node is integral to the bin-picking system, capturing RGB and depth data for object detection, pose estimation, and grasp planning. It publishes data to topics like /camera/color/image_raw for RGB images and /camera/depth/points for point clouds. These streams are synchronized to ensure precise mapping of objects in 3D space. A depth camera, simulated in Gazebo provides the required data. The node's modular design ensures accurate and real-time data flow, forming the backbone for object detection, pose estimation, and debugging via visualization tools like RViz.Camera Files

Fig. 3. Robot in Gazebo

Fig. 4. Gazebo World Setup

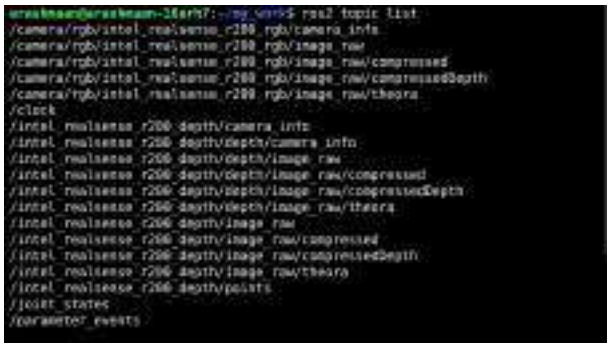


Fig. 5. Depth Camera Topics

C. Object Detection Node

The Object Detection Node is a core component of the bin-picking system, designed to detect objects in real-time using

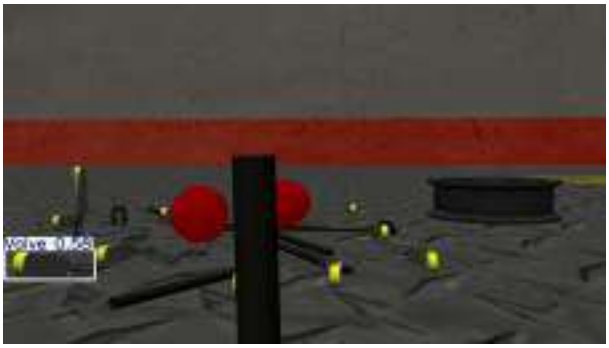


Fig. 6. Object Detection Using Yolo

an RGB camera feed. It leverages a YOLOv5 model trained on a custom dataset of industrial objects (e.g., hammers, tubes, valves) for accurate detection. The node subscribes to an input topic (e.g., /camera/image_raw) to receive RGB images, processes them through the YOLOv5 model, and publishes results as annotated images on /object_detection/image_annotated and detection data (bounding boxes, class IDs, and confidence scores) on /object_detection/detections. The detection results are essential for downstream processes like pose estimation and grasp planning, as they help match 2D detections with 3D point cloud data and guide the Grasp Pose Estimation Node in calculating optimal grasp positions. The node is tightly integrated into the ROS2 ecosystem, allowing seamless communication with other nodes, and can be launched using a dedicated ROS2 run file. Dependencies include PyTorch for model inference, OpenCV for image processing, and ROS2 libraries.Code

D. Pose Estimation Node

The Pose Estimation Node is a component of the bin-picking system, responsible for determining the precise 3D pose of detected objects using point cloud data. It processes the 2D object detections from the Object Detection Node and maps them to their corresponding 3D coordinates within the point cloud captured by a depth sensor. By leveraging the PCL (Point Cloud Library), the node filters, segments, and clusters the point cloud data to isolate objects and calculate their positions and orientations in the scene. The node subscribes to topics such as /object_detection/detections for 2D detection data and /camera/points for point cloud data and publishes the estimated 3D poses on /pose_estimation/poses. These poses include position (x, y, z) and orientation (roll, pitch, yaw) data, formatted for use by the motion planning system. The node is fully integrated with ROS2, enabling seamless communication with other nodes and facilitating tasks like grasp planning and pick-and-place operations. Users can launch the node using a dedicated ROS2 run file and rely on its robust processing pipeline for accurate and reliable pose estimation.Code

E. Grasp Point Detection Node

This node is responsible for determining the optimal grasp points for detected objects. It subscribes to the estimated

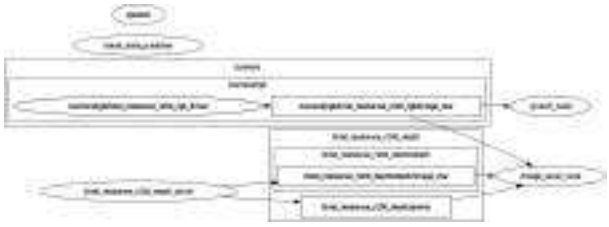


Fig. 7. rqt graph

poses from the `/pose_estimation/poses` topic and the point cloud data from the `/camera/depth/points` topic. Using the Grasp Pose Detection (GPD) algorithm, the node calculates suitable grasp points and publishes them to the `/grasping/grasp_points` topic, allowing the manipulator to execute a secure grasp.

F. MoveIt! Node

The MoveIt! node subscribes to the grasp points published by the grasp detection and pose estimation node. It uses these grasp points to generate collision-free motion plans for the robot manipulator. MoveIt! handles path planning, collision detection, and motion execution, ensuring safe and efficient robot movements for pick-and-place tasks. The final trajectory is executed based on the planned motion path. [9]

III. Results and Discussion

Our task was to bring a robot arm into Gazebo. For this, we used the Elephant Robotics *Cobot 280* with a gripper. Additionally, we experimented with Universal Robots' UR5 and UR3e arms. To accomplish this, we studied URDF and Xacro files in detail. We identified two methods to bring a robot into Gazebo and RViz:

- Using gazebo ros spawn entity.py and urdf tutorial
- Using Launch Files

The ROS2 supports two formats of launch files *filename.launch.xml* and *filename.launch.py*.We also find that typically ROS looks for launch files under launch folder. From the gained knowledge we tried to create the launch files to bring robots in gazebo and rviz. Though we are not completely successful in that till now. You can see our created files at this link. As we needed to create a package containing all urdf,meshes and launch files. We have uploaded the packages on the github. [14]

Then we needed a depth camera. For that we studied urdf, sdf files and gazebo plugin for sensors. Then we placed the files of camera package in */.gazebo/models*.The model become available in gazebo.

Then we divided work into parts one is creating a package for object detection and another is creating package for pose estimation. We have created two nodes which are subscribing to corresponding camera topics. We are still working on python script of these nodes. For object detection we have created a custom dataset and trained the yolov5 on that dataset. So far, we have learned and worked with the following concepts:

- `URDF`, `Xacro` files
- Creating a `ROS package`
- Configuring `CMakeLists.txt`
- Developing `launch` files
- Writing `ROS nodes` using Python
- Integrating a depth camera
- Using `Gazebo` plugins
- Understanding `tf` transformations
- Image Processing

IV. Plan for the Remaining Work

- Combining object detection and Pose estimation nodes and giving the output to grasp point detection
- Training GPD
- Creating object picking task using Moveit2 Constructor. [15]

V. Conclusion

Although we are not completely sucessfull in acheiving what we have planned. But will keep working. One of the major issue we found is the library compatability issues. All the work is available at Github.

VI. Contribution

Given the complexity of the project, the following work distribution is proposed to optimize the development process:

First both will work on preparation of simulation environment and adding camera to robot.

A. Teammate 1: Object Detection & Grasp Planning

- **Object Detection**:
  - Implement YOLOv5 in ROS2 to detect objects from RGB images.
  - Optimize the model for real-time detection using pre-trained weights.
  - Handle camera data input and perform testing in a simulated environment.
- **Grasp Point Detection**:
  - Implement Grasp Pose Detection (GPD) or a simple geometric grasping method.
  - Integrate pose estimation results with point clouds to compute optimal grasp points.

B. Teammate 2: Pose Estimation & Motion Planning

- **Environment Setup**:
  - Creating bin-picking environment in Gazebo
  - Setting up camera in Environment
- **Pose Estimation**:
  - Implement point cloud-based pose estimation using the Point Cloud Library (PCL).
  - Align object detection bounding boxes with point cloud data for accurate 6-DOF pose estimation.

References

[1] Wong, C.-C., Tsai, C.-Y., Chen, R.-J., Chien, S.-Y., Yang, Y.-H., Wong, S.-W. and Yeh, C.-A. (2022). Generic Development of Bin Pick-and-Place System Based on Robot Operating System.

[2] He, R., Juan Ignacio Rojas and Guan, Y. (2017). A 3D object detection and pose estimation pipeline using RGB-D images. 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO).

[3] Munoz, E., Konishi, Y., Murino, V. and A. Del Bue (2016). Fast pose estimation for texture-less objects from a single RGB image.

[4] François Hélénon, Johann Huber, Faïz Ben Amar and Stéphane Doncieux,Learning to Grasp: from Somewhere to Anywhere.(2021)

[5] Khanam, R. and Hussain, M. (2024). What is YOLOv5: A deep look into the internal features of the popular object detector.

[6] atenpas/gpd: Detect 6-DOF grasp poses in point clouds. (2019, November 2). GitHub.Github

[7] Pas, A. ten, Gualtieri, M., Saenko, K., Platt, R. (2017, June 29). Grasp Pose Detection in Point Clouds.

[8] Point Cloud Library. (n.d.). Point Cloud Library. https://pointclouds.org/

[9] MoveIt 2 Documentation — MoveIt Documentation: Humble documentation. (n.d.). Moveit.picknik.ai. MoveIt

[10] Universal Robots, "Collaborative Robots - UR5," .Univeral Robots

[11] The Robotics Back-End Creating a Node

[12] Intel Real-Sense Github

[13] Depth Camera in GazeboTutorial

[14] Project Files Uploaded on Github Github

[15] Pick and Place with MoveIt Task Constructor MoveIt Task Constructor

[16] How to Simulate a Robotic Arm in Gazebo – ROS 2

[17] Universal Robots

Designing a Lightweight NN for Crop Disease Detection

Arashdeep Singh (M23IRM003) Anvesh Shakargaye (M23EET002)

November 9, 2024

Abstract

The agriculture sector faces various challenges due to crop diseases, leading to losses in production and quality. Timely identification and precise diagnosis of diseases are essential for effective crop management. In this project, we focus on designing a lightweight Neural Network model capable of running on edge devices to assist in detecting crop diseases on-site.
Keywords: CNN, MobileNetV3, crop disease detection, edge computing,pytorch

Contents

1 Introduction

The emergence of new technologies is transforming agriculture towards a more smart and sustainable future. Machine learning (ML) is increasingly central to this transformation, enabling advancements in crop health monitoring, soil analysis, crop identification, and yield prediction. This project specifically addresses crop health monitoring through disease detection. By classifying images of crop leaves, we aim to identify disease symptoms to facilitate timely interventions. Although similar work has been conducted previously, we seek to make such techniques feasible on mobile and embedded devices, where resources are limited [3] [2].

One of our primary goals is to bring ML techniques for image classification to mobile devices. This could enable farmers to assess crop health using their smartphones. Additionally, agri-tech start-ups often use robots with embedded devices, which have limited computational resources and may lack high-speed internet connectivity. Our lightweight model can potentially offer on-field disease detection, improving accessibility and efficiency. To achieve this, we have designed a NN-based model inspired by MobileNetV3 architecture, specifically tailored for resource-constrained environments.

The dataset used for this project is the New Plant Diseases Dataset.

2 Image Classification

Image classification involves assigning images to predefined categories or classes. Convolutional Neural Networks (CNNs) are commonly employed for this task in machine learning.

2.1 CNN

Figure 1 depicts the basic architecture of a Convolutional Neural Network. Below, we provide an overview of CNN components and principles.

CNNs work through several key layers:

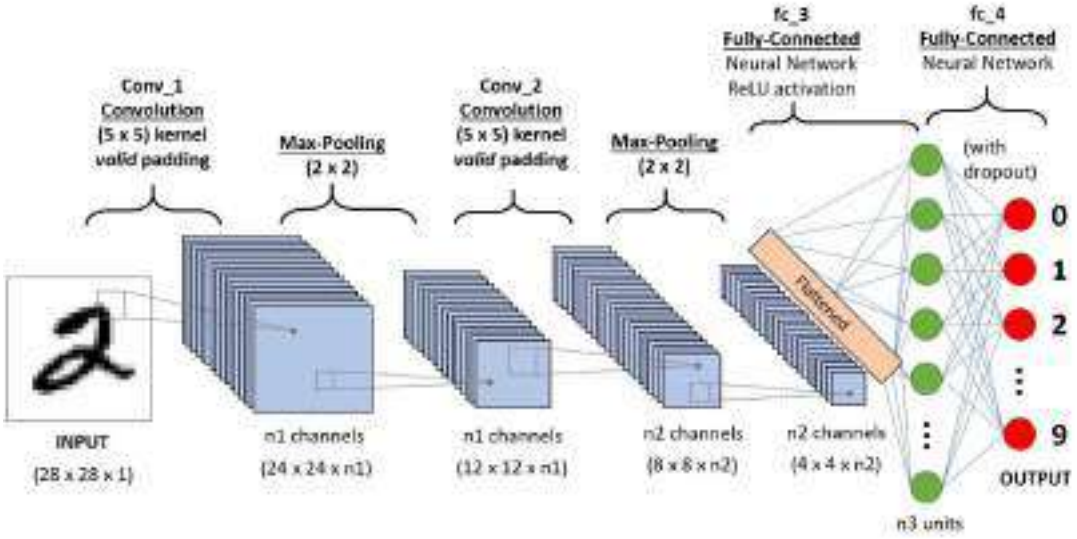


Figure 1: CNN Architecture

- **Convolution Layers:** These layers apply filters (small windows) to scan the image for specific features, such as edges or corners.
- **Pooling Layers:** Pooling layers condense information from convolution layers, reducing spatial dimensions and improving computational efficiency.
- **Flattening Layer:** This layer reshapes the condensed feature maps into a one-dimensional array, suitable for the fully connected layers.
- **Fully Connected Layers:** Acting as the network's "brain," these layers interpret the extracted features and make final predictions for each class.

CNNs learn through a process called backpropagation, adjusting filters and weights based on prediction errors. This iterative learning process enables CNNs to recognize and generalize visual patterns for various tasks, including image classification, object detection, and segmentation.[4] [1] [7]

2.2 MobileNetV3

MobileNetV3 is an efficient neural network architecture optimized for mobile and embedded devices. It combines several advanced techniques, including depthwise separable convolutions, inverted residuals, and squeeze-and-excitation modules, which reduce computational costs and enhance accuracy. MobileNetV3 also introduces the hard-swish activation function and employs a combination of manual design and neural architecture search for optimization. Available in two versions, MobileNetV3-Large and MobileNetV3-Small, it balances performance and efficiency, making it ideal for mobile image classification and object detection tasks.[6]

Key Components of MobileNetV3 include the following:

Depthwise Separable Convolutions

- **Standard Convolution:** In traditional convolutional layers, each filter is applied to all input channels, which is computationally intensive.
- **Depthwise Separable Convolution:** This technique divides convolution into two steps:
  1. **Depthwise Convolution:** Applies a single filter to each input channel individually.
  2. **Pointwise Convolution:** Uses a 1×1 convolution to combine outputs from the depthwise convolution.

This approach greatly reduces computations and parameters, improving model efficiency.

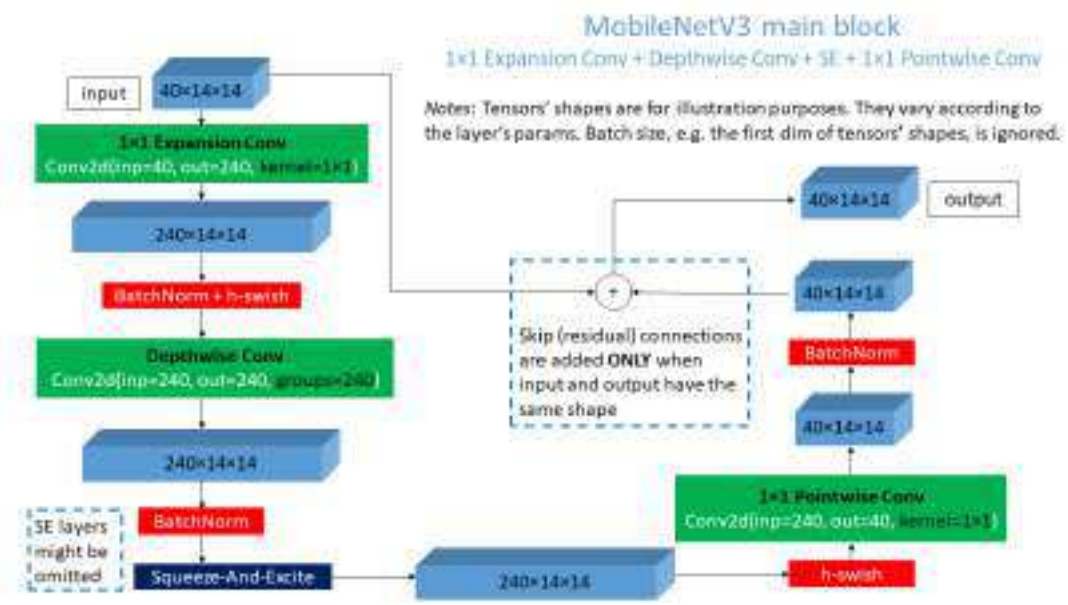


Figure 2: MobileNetV3 Architecture

Inverted Residuals and Linear Bottlenecks

- **Inverted Residuals:** Unlike traditional residual blocks, which expand channels, inverted residuals first expand channels, apply depthwise convolution, and then project back to a lower dimension.
- **Linear Bottlenecks:** To retain information during projection, linear transformations are employed, preserving essential features.

Squeeze-and-Excitation (SE) Modules

These modules enable the network to focus on critical features by recalibrating channel-wise responses, using:

- **Squeeze:** Aggregates global information via average pooling.
- **Excitation:** Learns channel-wise dependencies and scales input channels accordingly.

Efficient Last Stage

MobileNetV3's final stage replaces traditional fully connected layers with global average pooling, followed by a single fully connected layer, which reduces parameters and computations. [5]

3 Our Model(SimpleMobileNetV3Lite)

The SimpleMobileNetV3Lite model is designed to be lightweight and computationally efficient, following a streamlined version of the MobileNetV3 architecture with selective optimizations, including Hard-Swish activation and Squeeze-and-Excitation (SE) blocks. The model consists of a series of sequentially connected layers, organized as follows:

- **Initial Convolution Layer:** A 3x3 convolution with a stride of 2 and padding of 1, followed by batch normalization and the Hard-Swish activation function. This layer downsamples the input and extracts low-level features from the image.

$$\text{Conv2d}(3, 16, \text{kernel\_size} = 3, \text{stride} = 2, \text{padding} = 1)$$

- **Bottleneck Blocks:** The core of the network consists of bottleneck blocks that alternate between layers with and without Squeeze-and-Excitation (SE) blocks, using a depthwise separable convolution structure for efficient computation. Each bottleneck block has three main components:

1. **Pointwise Convolution (Expansion):** Expands the input channels by a specified expansion factor using a 1x1 convolution.
2. **Depthwise Convolution:** Applies a depthwise convolution with either a 3x3 or 5x5 kernel. Some blocks include an SE block that recalibrates the feature maps.
3. **Pointwise Convolution (Projection):** Projects the output channels back to the specified number for the next layer.

The bottleneck blocks in this model are defined as follows:

- **Block 1:** (16, 16), Expansion Factor = 1, Kernel Size = 3, **SE Block included**
- **Block 2:** (16, 24), Expansion Factor = 4, Kernel Size = 3, **No SE Block**
- **Block 3:** (24, 24), Expansion Factor = 4, Kernel Size = 3, **No SE Block**
- **Block 4:** (24, 40), Expansion Factor = 4, Kernel Size = 5, **SE Block included**
- **Block 5:** (40, 40), Expansion Factor = 4, Kernel Size = 5, **SE Block included**

- **Final Convolution Block:** A 1x1 convolutional layer with 96 output channels, followed by batch normalization and Hard-Swish activation. This block refines the high-level features extracted by the bottleneck layers.

$$\text{Conv2d}(40, 96, \text{kernel\_size} = 1)$$

- **Global Pooling and Fully Connected Layer:** The model includes a global average pooling layer to reduce the spatial dimensions, followed by a fully connected layer with a softmax output for classification.

$$\text{Global Average Pooling} \rightarrow \text{Flatten}$$
$$\text{Fully Connected Layer} : \text{Linear}(96, \text{num\_classes})$$

Design Considerations

The SimpleMobileNetV3Lite is specifically crafted to be efficient for mobile and edge devices. Key design elements include:

- **Hard-Swish Activation:** A computationally efficient activation function, which approximates the Swish activation, improving model accuracy with minimal computational cost.
- **Squeeze-and-Excitation (SE) Blocks:** These blocks are selectively applied to recalibrate feature maps in the bottleneck blocks, enhancing the model's representational power without substantial computational overhead.
- **Depthwise Separable Convolutions:** Used in the bottleneck blocks to significantly reduce the number of parameters and computations compared to standard convolutions.

This architecture maintains high accuracy while ensuring a smaller model size and reduced floating-point operations (FLOPs), making it suitable for deployment on resource-constrained devices.

4 Results and Discussion

Our Model has size of 285 KB. One Major trade-off is the training time compared to other models. It is taking 10 times more time to train on same dataset. With almost 20 times reduction in size we are getting almost same accuracy making it suitable for mobile devices. With just 62,682 parameters, it significantly reduces storage and memory demands, achieving a tiny model size of 0.2853 MB—far smaller than both the standard CNN (392.15 MB) and MobileNetV3 (6.07 MB). Inference speed is also highly competitive, with an average time per sample of 0.2898 ms, slightly faster than MobileNetV3's 0.3167 ms. This design ensures strong performance in edge contexts but at the expense of extended training time.

(a) Inference on Tomato Leaf (b) Inference on Potato Leaf (c) Inference on Apple Leaf

Figure 3: Inference Result on Test Images

Parameter	CNN	MobileNet V3	Our Model
Architecture Type	CNN	MobileNetV3	Based on MobileNetV3
Number of Layers	5 layers	13 layers	13 layers
Total Parameters	102799846	1556806	62682
Model Size (Weights)	392.15MB	6.07 MB	0.2853 MB
Inference Time per sample	0.3941 ms	0.3167 ms	0.2898 ms
Training Time	2.6 hours	1.31 hours	15.88 hours
Training Accuracy	0.90	0.9806	0.9617
Validation Accuracy	0.935	0.99	0.99
No. of epochs	25	25	25
Framework	PyTorch	PyTorch	PyTorch

Table 1: Comparison of Models

Acknowledgements We would like to acknowledge the colleagues,sources and datasets used for our project.

*More comparison Parameters
Codes Are Available at Github

References

[1] An Introduction to Convolutional Neural Networks: A Comprehensive Guide to CNNs in Deep Learning — datacamp.com. https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns. [Accessed 29-10-2024].

[2] Machine Learning Applications in Agriculture: Current Trends, Challenges, and Future Perspectives — mdpi.com. https://www.mdpi.com/2073-4395/13/12/2976. [Accessed 29-10-2024].

[3] Machine learning in agriculture domain: A state-of-art survey — sciencedirect.com. https://www.sciencedirect.com/science/article/pii/S2667318521000106. [Accessed 29-10-2024].

[4] Dharmaraj. Convolutional Neural Networks (CNN)—Architectures Explained — draj0718. https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243. [Accessed 29-10-2024].

[5] frapochetti. A visual deep-dive into the building blocks of MobileNetV3 - — francescopochetti.com. https://francescopochetti.com/a-visual-deep-dive-into-the-building-blocks-of-mobilenetv3/. [Accessed 29-10-2024].

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, M Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. URL https://doi.org/10.48550/arxiv.1704.04861.

[7] Keiron O'Shea1 and Ryan Nash. An introduction to convolutional neural networks. *arXiv:1511.08458v2 [cs.NE] 2 Dec 2015*, 2017. URL https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks.

Development of Multi-Agent Coordination and Control for Surface and Underwater Vehicles

A Project Report Submitted by

Arashdeep Singh

in partial fulfillment of the requirements for the award of the degree of

M.Tech

Indian Institute of Technology Jodhpur

Inter-disciplinary Research Division(IDRD)

April, 2025

Declaration

I hereby declare that the work presented in this Project Report titled **Development of Multi-Agent Coordination and Control for Surface and Underwater Vehicles** submitted to the Indian Institute of Technology Jodhpur in partial fulfilment of the requirements for the award of the degree of M.Tech, is a bonafide record of the research work carried out under the supervision of Dr.Jayant Kumar Mohanta. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.

Signature

Arashdeep Singh

M23IRM003

Certificate

This is to certify that the Project Report titled **Development of Multi-Agent Coordination and Control for Surface and Underwater Vehicles**, submitted by Arashdeep Singh(M23IRM003) to the Indian Institute of Technology Jodhpur for the award of the degree of M.Tech, is a bonafide record of the research work done by him under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Signature

Dr.Jayant Kumar Mohanta

Acknowledgements

I want to convey my sincere gratitude to my thesis supervisor, Dr. Jayant Kumar Mohanta, for his invaluable advice, constant encouragement, genuine concern, prompt assistance, and the creation of a great environment for conducting research. He has offered me pleasant and kind support throughout the project so that I may finish my research project. His belief in my abilities and constant encouragement helped me push through challenging moments and remain dedicated to my research goals.

I would like to thank Mr. Jay Khatri (Ph.D.) and Mr. Ravindra Kumar (Research Assistant) for their constant guidance, assistance, and suggestions on the project work.

Abstract

This study aims to create a framework for decision making and obstacle avoidance for marine robots capable of surface and underwater operations. Core elements of this framework are a multi-agent coordination system for task allocation and role switching and an obstacle-avoidance module. Utilizing priority-based task allocation, agents with flexible tasks may optimally handle mission-critical operations. Using dynamic leader-follower configurations, this system promotes cooperative behavior among several agents when necessary, therefore improving mission efficiency. This multi-agent agent system depends on autonomous systems. One important idea for autonomy is obstacle avoidance, particularly concave obstacle avoidance and passing through enclosures in complicated underwater settings. Our main emphasis was on solving the difficulties of concave obstacle avoidance and distributed self-task allocation.

Contents

List of Figures

List of Tables

List of Symbols Used

Symbol	Meaning
X	Surge force
Y	Sway force
Z	Heave force
K	Roll moment
M	Pitch moment
N	Yaw moment
$\dot{\nu}$	Time derivative of velocity vector
ν	Velocity vector
$M.$	Inertia matrix (including added mass)
$C(\nu)$	Coriolis and centripetal matrix
$D(\nu)$	Damping matrix
$g(\eta)$	Gravitational and buoyant forces
τ	Control input vector
u	Surge velocity
v	Sway velocity
w	Heave velocity
r	Yaw rate
m_{22}	Mass term for sway
m_{33}	Mass term for heave
m_{44}	Mass term for yaw
m	Mass of Vehicle
B_f	Buoyancy force
W	Weight of the vehicle
φ	Yaw angle

Symbol	Meaning
d_{33}	Damping term for heave
d_{44}	Damping term for yaw
θ	Pitch angle
ϕ	Roll angle
$\dot{x}$	Time derivative of x position (surge)
$\dot{y}$	Time derivative of y position (sway)
$\dot{z}$	Time derivative of z position (heave)
$\dot{\phi}$	Time derivative of roll angle
$\dot{\theta}$	Time derivative of pitch angle
$\dot{\varphi}$	Time derivative of yaw angle
X_r	Hydrodynamic drag in surge
Y_r	Hydrodynamic drag in sway
N_r	Hydrodynamic drag in yaw
I_z	Moment of inertia about the z axis
$\dot{u}$	Surge acceleration
$\dot{v}$	Sway acceleration
$\dot{w}$	Heave acceleration
$\dot{r}$	Yaw acceleration
$s(\cdot)$	Sine function
$c(\cdot)$	Cosine function
$t(\cdot)$	Tangent function
$se(\cdot)$	Secant function

Symbol	Meaning
ξ	State Variable
$\dot{\xi}$	Rate change of state variable with time
κ	Directional Space Vector
K	Directional Space
$\bar{\kappa}$	Mean Directional Space from all Obstacles
Γ	Distance function
w_i	Weight for Directional Space
ξ^a	Attractor Position
$C(\xi)$	Convergence Dynamics
$f(\xi)$	Initial Dynamics
$\mathbb{R}^{\mathbb{N}}$	Real Space of Dimension N
$r(\xi)$	Reference Direction
$e(\xi)$	Pesudo Tangent
$n(\xi)$	Normal
$\hat{v_i}$	Direction Space With respect to Obstacle
B	Orthonormal Matrix

Symbol	Meaning
I_{xx}	Inertia in x-axis
I_{yy}	Inertia in y-axis
I_{zz}	Inertia in z-axis
$X_{\dot{u}}$	Added Mass
$M_{\dot{q}}$	Added Mass
$N_{\dot{r}}$	Added Mass
$X_{\|u\|u}$	Hydrodynamics Drag
$M_{\|q\|q}$	Hydrodynamics Drag
$N_{\|r\|r}$	Hydrodynamics Drag
I^*_{yy}	Pitch Inertia
I^*_{zz}	Yaw Inertia

1 Introduction

As advances in technology are growing at a faster pace, humans desire to make machines capable of working without human intervention. These are known as unmanned systems. And we want unmanned systems to work for us in almost every domain, be it medical, construction, etc. One such domain is underwater operations.

In the vast, ever-changing underwater world, relying solely on one vehicle is like trying to navigate uncharted waters by yourself. A fleet of autonomous surface vehicles (ASVs) and underwater vehicles (AUVs) cooperating is used in a multi-agent system. In addition to increasing efficiency, this coordinated approach adds resilience because, in the event that one agent experiences an issue, other agents can adjust and step in to ensure mission continuity.

Complex missions require flexibility. Underwater conditions may change rapidly, most likely due to currents, unexpected marine life, or evolving mission parameters. Dynamic task allocation enables each vehicle to reassess its environment and capabilities on a continuous basis, switching roles and priorities in real-time. This smooth distribution of tasks ensures that the most suitable agent is always playing the most critical role, optimizing performance and conserving energy.

For robots, the underwater world is full of dangers like rocks and big marine life. Avoiding these dangers is essential for robot safety and mission success. Advanced algorithms help robots to safely navigate around these dangers. Which ensures progress towards the goal.

2 Literature survey

- **Sahoo et al.** highlights developments of AUVs, mentioning the work for which they are developed, their capability, and their design. Apart from this authors discussed various under water communication modems and their capability comparison. Also, the paper briefly introduces modelling, control, and path planning for AUVS. [1]

- **T. I. Fossen** This book provides detailed modeling and control techniques for marine vehicles, including ASVs and AUVs. Fossen's work is instrumental in understanding ASV/AUV motion dynamics and control. The book discusses mainly two theories, maneuvering Theory and Seakeeping Theory, for modelling dynamics of marine vehicles. [2]

- **Rui Hu et al.** focused on hybrid aerial underwater vehicle's buoyancy control. They have used a piston-based actuator to control the buoyancy of the vehicle. Further they have employed PID controller. They have created a prototype names as Nezha III. They are controlling pitch and heave based on buoyancy. They also analyzed the effect of speed of piston movement. [3]

- **Mohd Aras et al.** discuss the hull design, propulsion system, and dynamic analysis of an underwater vehicle. They have developed the actual setup. Also they have discussed the parameters effecting performance of AUV like pressure, environmental forces. [4]

- **Aditya Natu et al.** discuss the design and development of AUV named VARUNA 2.0. They have employed a PID controller. Also they have discussed about task detection and task execution. They have emolyed two batteries one for back up and discussed operating time off the vehicle. They have also discussed Computer vision applications and challenges in underwater. [5]

- **Leong Wai Lunn et al.** focused on developing AUV that autonomously navigates for a predefined task. They have employed TNMM module with camera and depth sensor. They also discuss the design and propulsion dynamics of AUVs. They also demonstrated the prototype developed based on the target acquisition test and navigation test. [6]

- **Wang et al.** discusses task allocation for homogenous and heterogenous agents. They have divided the existing algorithms into centralized and decentralized approaches. They also introduced parallel task allocation for multiple tasks. They also show simulation of SOM, Ants, GA, PSO and Auction algorithms. They showcased layered planning to improve the hierarchical planning of tasks. [7]

- **Esther Bischoff et al.** have introduced greedy-constructive heuristic which yields feasible initial solutions independent of the problem size. For task execution, they have introduced a local improvement heuristic. They have simulated the introduced method, discussing the efficiency of the method introduced on multi-task allocation. [8]

- **Y. Ma, Y. Liu, L. Zhao, and M. Zhao,** have discussed the problems faced by multi-agent control. They have discussed consensus, formation, and flocking as major divisions of a multi-

agent system. Then they discuss problems like Connectivity, Distributed estimation, Containment control, and Coverage control. [9]

- **Z. Fang et al.** have discussed problems of multi-agent reinforcement learning in uncertain marine applications. So, they explored the use of multi-agent generative adversarial imitation learning (MAGAIL) in autonomous underwater vehicles (AUVs). They have used Unmanned Underwater Vehicle (UUV) simulator with gazebo to simulate MAGIL for formation control and obstacle avoidance. They have used decentralized training with decentralized execution framework. [10]

- **M. Liu and Y. Feng** examines group consensus in mixed-order multi-agent systems, accommodating both first- and second-order agents under fixed, directed topologies. Stability and control strategies are discussed, providing valuable insights for designing task-switching and priority-based allocation strategies in complex agent architectures [11].

- **Y. Bai et al.** Bai's research on multi-agent coverage control addresses static and dynamic obstacle avoidance using Voronoi partitions. Adaptive control mechanisms are discussed, ideal for multi-agent navigation in environments with obstacles like underwater terrains. This study supports task allocation frameworks ensuring coordinated movement and obstacle avoidance [12].

- **L. Wen, J. Yan, X. Yang, Y. Liu, and Y. Gu** present a method for trajectory planning in ASVs, focusing on efficient and safe navigation in obstacle-dense environments. The proposed framework decouples the planning process into path searching and trajectory optimization, with a specific focus on energy-efficient, collision-free routes. This study is relevant for ASV applications requiring reliable path planning under dynamic constraints and fuel efficiency considerations [13].

- **Lei Cao, Guo-Ping Liu, Da-Wei Zhang** proposes a predictive control-based leader-follower consensus approach for networked multi-agent systems. The method compensates for communication delays and ensures consensus and stability in environments with dynamic topologies. The system's effectiveness is validated via experiments using air-bearing simulators [14].

- **J. Xu et al.** apply deep reinforcement learning in multi-AUV attack-defense scenarios, enabling AUVs to develop cooperative strategies under limited perception. The proposed framework achieves effective multi-agent coordination, demonstrating the suitability of reinforcement learning for decision-making in high-stakes underwater missions [15].

- **B.Xu et al.** presents a novel approach to formation control for Autonomous Underwater Vehicles (AUVs) under dynamic switching topology. The authors propose a double-layer distributed formation prediction control scheme, which includes an upper-level adaptive distributed observer and a lower-level predictive controller. This method addresses the challenges posed by external time-varying disturbances and ensures robust formation maintenance. [16]

- **Huber et al.** worked on multiple moving obstacle avoidance specially focusing on convex and star shaped obstacles. Their work is inspired from harmonic potentials. They validated their work in

grocery store pick and place task. Their work is based on modulating the original dynamics of the system. [17]

- **Huber et al.** worked on concave obstacle avoidance by modifying original non linear dynamics of the dynamical system. The main idea is rotating the initial dynamics into a tangent space which is safe to move around obstacles. The proposed method is highly reactive and effective in dynamic environments. They demonstrated proposed approach on 7-DoF robot arm around dynamic obstacles. [18]

- **Khansari et al.** discuss about dynamical systems for real time obstacle avoidance. They discussed the problem of sudden obstacles. They discussed about Harmonic Potential function. They worked on modifying the normal component of velocity field according to obstacle which will result in tangent component making robot to slide on the boundary of the obstacle. [19]

3 Problem definition and Objective

There have been a lot of ongoing developments related to marine robotics. Many problems still need more focus and alternative approaches. One such problem is the underwater search operations. Let us understand this problem in more detail. There is a surface vehicle and several underwater vehicles. As in underwater search operations, we will try to explore every possible and feasible direction. Let there be eight directions, such as NW, NE, SW, etc. Now, the question arises: How can it be effectively done? One answer is a multi-agent system with dynamic task allocation. This task allocation will help agents decide what best they can perform. Thus, we can properly utilize our resources. So the solution involves autonomus vehciles in multi- agent frame work with dynamic task assignment. For the effective working of marine vehicles for surface and underwater operations, It is necessary to have strong control systems, decision-making algorithms, communication, and hardware development. Moreover, for multi-agent systems, it becomes essential to develop a framework that will decide how they will behave in dynamic situations. Like agents are in a formation with leader follower mechanism. And suddenly, the leader gets destroyed by the enemy. Now, to complete the assigned task, there should be a new leader.

A major challenge in achieving autonomy is obstacle avoidance, especially concave obstacles and enclosures. As in the underwater scenario, there is much more possibility of gaps between objects like caves. Currently, available methods try to avoid them entirely by ignoring the gap. However, for better maneuverability and efficient operations, we want the capability of moving in and out of enclosures.

Apart from that, we have primarily underactuated underwater vehicles as we utilize fins for orientation control to get more working time. However, it introduces the problem of controlling underactuated robots.

Objectives Based on above problem statement these are my primarliy objectives :-

- Modelling of AUV,ASV:- Finding accurate dynamic models to represent both the autonomous underwater vehicle (AUV) and the autonomous surface vehicle (ASV) is essential for simulating real-world behavior, enabling effective control.
- Control of Underactuated Vehicles:- Creating control systems to track specified trajectories, prevent obstacles, and preserve stability.
- Obstacle Avoidance and Navigation: AUVs with autonomous obstacle detection and Avoidance ensures safe operation and efficient navigation. This capability is critical for uninterrupted multi-agent operations in dynamic underwater environments.
- Task Switching and Priority-Based Allocation: I wish to allow every AUV to assume duties depending on its capacity, therefore enabling dynamic task switching and reallocation depending on mission priorities. This will guarantee that high-priority tasks get attention from the most competent agents and maximize resource use.
- Simulation: Building and integrating consistent simulation soultion that will support to justify the suggested solutions. The main task will be to create a simulation environment which will have capability to simulate underwater and surface vehicles at same time.

4 Methodology

4.1 System Modeling and Control

Develop and understand mathematical models for Autonomous Surface Vehicles (ASVs) and Autonomous Underwater Vehicles (AUVs). These models incorporate hydrodynamic forces, moments, and external disturbances. Control surfaces like rudders and thrusters are modeled to reflect underactuated control and environmental interactions.

4.2 Control for AUVs

Implement control strategies for underactuated agents based on their dynamics. I will be using cascaded control with PID as outer loop and to remove non-linearity will use Feedback linearization in the inner loop.

4.3 Multi-Agent Coordination and Control Strategy

Task Allocation and Switching

- **Priority-Based Task Assignment**: Develop a task assignment framework where tasks are allocated based on each agent's capabilities and mission priority requirements, ensuring high-priority tasks are given to the most capable agents.
- **Dynamic Task Switching**: Implement a mechanism for dynamic task-switching, allowing agents to reassign tasks based on real-time conditions, such as obstacle detection, to maintain formation and mission success in uncertain environments.
- **Event-Triggered Control (ETC)**: To reduce communication load, employ ETC to trigger updates only upon significant state deviations, minimizing data exchange while maintaining system performance under limited bandwidth conditions.

4.4 Concave Obstacle Avoidance

To make the robot move through caves and archways, we need to work on concave obstacle avoidance. To make it possible, we will use the concept of rotating dynamics away from the obstacle in a pseudo-tangent direction.

5 Mathematical Modelling and Design

5.1 Modelling of Underwater Vehicle

Following the Manuvering Theory Given by T. I. Fossen [2], I have worked on kinematics and dynamics of surface and underwater vehicles.

Kinematics of Six DOF Underwater Vehicle:- It describes how the vehicle's position (x, y, z) and orientation (roll ϕ, pitch θ, yaw φ) change over time based on its control inputs like linear velocities (u, v, w) and angular velocities (p, q, r). Essentially, it captures the complex motion dynamics of the vehicle as it navigates underwater.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} c(\varphi)c(\theta) & -s(\phi)c(\phi) + c(\varphi)s(\theta)s(\phi) & s(\varphi)s(\phi) + c(\varphi)c(\phi)s(\theta) & 0 & 0 & 0 \\ s(\varphi)c(\theta) & c(\varphi)c(\phi) + s(\phi)(\theta)(\varphi) & -c(\varphi)s(\phi) + s(\theta)s(\varphi)c(\phi) & 0 & 0 & 0 \\ -s(\theta) & c(\theta)s(\varphi) & c(\theta)c(\phi) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & 0 & 0 & 0 & c(\phi) & -s(\phi) \\ 0 & 0 & 0 & 0 & s(\phi)se(\theta) & c(\phi)se(\theta) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix}$$

4-DOF Dynamic Model of an Autonomous Underwater Vehicle (AUV)

The equations of motion for an AUV in surge, sway, heave, and yaw (4-DOF) are given by:

Surge Equation:

$$m_{11}\dot{u} - m_{22}wr + m_{33}vq + d_{11}u = \tau_u \tag{1}$$

Sway Equation:

$$m_{22}\dot{v} + m_{11}wr - m_{33}uq + d_{22}v = \tau_v \tag{2}$$

Heave Equation:

$$m_{33}\dot{w} + m_{11}vq - m_{22}ur + d_{33}w = \tau_w - (W - B) \tag{3}$$

Yaw Equation:

$$m_{44}\dot{r} + m_{22}wu - m_{11}vq + d_{44}r = \tau_r \tag{4}$$

5.2 Modelling of Surface Vehicle

For the small boat with 3 degrees of freedom (surge, sway, yaw), the kinematics and kinetics can be expressed in both equation form and vector representation under these assumptions:-

- The motion in roll, pitch, and heave is ignored. This means that we ignore the dynamics associated with the motion in heave, roll, and pitch, i.e., (z = 0), (p = 0), and (q = 0).
- The vessel has homogeneous mass distribution and an xz-plane of symmetry so that ($I_{xy} = I_{yz} = 0$).
- The center of gravity (CG) and the center of buoyancy (CB) are located vertically on the z-axis.

The kinematic equations in the body-fixed frame can be written as:

$$\dot{x} = u\cos\varphi - v\sin\varphi \tag{5}$$

$$\dot{y} = u\sin\varphi + v\cos\varphi \tag{6}$$

$$\dot{\varphi} = r \tag{7}$$

Vector Representation of Kinematics

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

The **dynamic equations** in the body-fixed frame can be written as:

$$m(\dot{u} - vr) = T - X_r \tag{8}$$

$$m(\dot{v} + ur) = Y_\delta - Y_r \tag{9}$$

$$I_z\dot{r} = N_\delta - D_r \tag{10}$$

5.3 Multi-Agent Coordination and Control

I will discuss the intuitive foundation and mathematical formulation of a priority-aware, event-triggered multi-agent coordination and control algorithm. I detail how each agent's internal awareness , capability evaluation, task prioritization, and event-driven re-planning combine to yield adaptive team behavior.

Agent Awareness Each agent i maintains a local state vector:

$$\mathbf{s}_i = \begin{bmatrix} E_i \\ \mathbf{p}_i \\ \mathbf{v}_i \\ \vdots \end{bmatrix} \tag{11}$$

where:

- E_i is the remaining battery energy of agent i.
- $\mathbf{p}_i \in \mathbb{R}^n$ is the current position.
- $\mathbf{v}_i \in \mathbb{R}^n$ is the current velocity.
- Additional entries may represent sensor health, payload, or other relevant states.

This awareness enables each agent to estimate its suitability for any task.

Capability Evaluation For a task j characterized by a requirement vector $\mathbf{r}_j$, define the capability function:

$$C_i^j = f(\mathbf{s}_i, \mathbf{r}_j) = w_E \, g_E(E_i, r_j^E) + w_d \, g_d\big(\|\mathbf{p}_i - \mathbf{p}_j^*\|\big) + \cdots, \tag{12}$$

where:

- r_j^E is the energy requirement for task j.
- $\mathbf{p}_j^*$ is the task location.
- $g_E(E_i, r_j^E) = \min\{1, E_i/r_j^E\}$ normalizes energy availability.
- $g_d(\|\mathbf{p}_i - \mathbf{p}_j^*\|)$ normalizes distance.
- $w_E, w_d \geq 0$ are weights with $w_E + w_d + \cdots = 1$.

Thus, $C_i^j \in [0, 1]$ quantifies how capable agent i is of performing task j.

Task Priority and Utility Each task j has a priority $\rho_j > 0$. The utility of assigning agent i to task j is defined as:

$$U_i^j = \rho_j \, C_i^j. \tag{13}$$

The global objective is to maximize total utility:

$$\max_{\mathcal{A}} \sum_{(i,j)\in\mathcal{A}} U_i^j, \tag{14}$$

subject to assignment constraints (e.g., each agent assigned to at most one task).

Event-Triggered Decision Making Agents monitor for events $\mathcal{E}$ (e.g., $E_i < E_{\min}$, obstacle detection, task completion). Define an event indicator:

$$\delta_e(t) = \begin{cases} 1, & \text{if an event in } \mathcal{E} \text{ occurs at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

When $\delta_e(t) = 1$, the system re-computes C_i^j, U_i^j, and resolves the assignment.

Algorithm Flow

1. **Initialization:** Agents broadcast initial states $\mathbf{s}_i(0)$.
2. **Capability Computation:** Compute C_i^j for all agent-task pairs.
3. **Utility Matrix:** Form $U \in \mathbb{R}^{N \times M}$ with entries U_i^j.
4. **Assignment:**
$$\mathbf{X}^* = \arg \max_{\mathbf{X} \in \{0,1\}^{N \times M}} \text{trace}(\mathbf{X}^T U),$$
5. **Execution:** Agents execute assigned tasks.
6. **Monitoring & Events:** If $\delta_e(t) = 1$, return to Step 2.

Algorithm 1 Multi-Agent Task Allocation with Capability Sharing, Task Switching, and Formation Control

Initialize task allocator $T.A$
Initialize agents $A_1, A_2, \ldots, A_n$
Initialize mission parameters and environment
$T \leftarrow$ AllocateTasks($T.A$) {Task allocator floats tasks(may with priority) to agents}
for each agent A_i **do**
  Set initial position P_i and state S_i
  A_i.tasks $\leftarrow T$ {Receive task list, with or without priority numbers}
end for
while mission is active **do**
  for each agent A_i **do**
    $S_i \leftarrow$ GetState(A_i)
    $C_i \leftarrow$ AssessCapability(A_i)
    A_i.share $\leftarrow (A_i.\text{name}, C_i)$ {Agents share their capabilities}
    A_i.info $\leftarrow$ GatherInfo($A_{1,2,\ldots,n}$) {Each agent gathers others' capabilities}
    A_i.assigned_tasks $\leftarrow$ SelectBestTasks(A_i.info, A_i.tasks) {Assign tasks based on priority and capability score; if tasks have no priority, assign based only on capability score}
  end for
  Agents confirm selected tasks with each other, then break off communication
  if EventTriggered() **then**
    {Trigger event for reassessment} **for** each agent A_i **do**
    $C_i \leftarrow$ ReassessCapability(A_i)
    A_i.share $\leftarrow (A_i.\text{name}, C_i)$ {Share updated capabilities}
    A_i.info $\leftarrow$ GatherInfo($A_{1,2,\ldots,n}$)
    A_i.assigned_tasks $\leftarrow$ ReallocateTasks(A_i.info) {Re-allocate tasks}
    end for
    if HighPriorityAgentDestroyed() **then**
      {Task switching triggered if high-priority task agent is lost} Reshuffle tasks to reassign high-priority tasks to the most capable available agents
    end if
  end if
  $G \leftarrow$ FormGroups($A_1, A_2, \ldots, A_n$)
  for each group G_k in G **do**
    $L_k \leftarrow$ SelectLeader(G_k) {Most capable agent becomes leader}
    $L_k \leftarrow$ CoordinateActions(G_k) {Leader coordinates with followers}
  end for
  MonitorProgress()
end while
CompleteMission() =0

Algorithm 2 Capability Assessment and Task Assignment

Input: *agents* (list of agents with attributes), *task_details* (list of tasks with details)
Output: *task_assignments* (list of task-to-agent assignments)
Define information shared by each agent:
Each agent shares:

- Position (x, y, z coordinates)
- Battery level and battery capacity (normalized)
- Communication and movement capabilities
- Vehicle type (e.g., 'surface' or 'underwater')

Define a strategy for assessing capability:
Function $CalculateEnergyPriority(agents, task\_details)$
Initialize $energy\_priority \leftarrow []$
for each *agent* in *agents* **do**
  Initialize $agent\_energies \leftarrow []$
  for each *task* in *task_details* **do**
    $distance \leftarrow$ EuclideanDistance$(agent.position, task.position)$
    Append *distance* to *agent_energies*
  end for
  Normalize *agent_energies* to range [0,1] and append to *energy_priority*
end for
return *energy_priority*
Function $AssessCommunication(agent)$
$communication\_score \leftarrow w_1 \times agent.range + w_2 \times agent.bandwidth + w_3 \times (1 - agent.latency) + w_4 \times agent.reliability$
return *communication_score*
Function $AssessMovement(agent)$
$movement\_score \leftarrow u_1 \times agent.speed + u_2 \times agent.maneuverability + u_3 \times agent.terrain\_adaptability + u_4 \times agent.endurance$
return *movement_score*
Function $AssessCapability(agent)$
$normalized\_battery \leftarrow agent.battery\_level / agent.battery\_capacity$
$communication\_score \leftarrow AssessCommunication(agent)$
$movement\_score \leftarrow AssessMovement(agent)$
$overall\_score \leftarrow v_1 \times normalized\_battery + v_2 \times communication\_score + v_3 \times movement\_score$
return *overall_score*
Function $GenerateTaskAssignments(agents, task\_details)$
$energy\_priority \leftarrow CalculateEnergyPriority(agents, task\_details)$
Initialize $task\_assignments \leftarrow []$
for each $i, task$ in *task_details* **do**
  $task\_type \leftarrow task.type$
  $eligible\_agents \leftarrow []$
  for each $agent\_index, agent$ in *agents* **do**
    if agent.vehicle_type == 'surface' or (agent.vehicle_type == 'underwater' and task_type == 'underwater') **then**
      $capability\_score \leftarrow AssessCapability(agent) + energy\_priority[agent\_index][i]$
      Append $(agent, capability\_score)$ to *eligible_agents*
    end if
  end for
  Sort *eligible_agents* by *capability_score* in descending order
  if two or more agents have the same highest *capability_score* **then**
    Break ties based on:

    - Higher battery level
    - Closer Euclidean distance to task position

  end if
  Assign the agent with the highest (or tie-broken) score to *task*
  Append $(task.id, selected\_agent.id)$ to *task_assignments*

5.4 Obstacle Avoidance

Obstacle avoidance is a key component for autonomy of robots. It is responsible for several key components like Safety, Efficiency, Lifespan, and autonomous navigation. Let us discuss some keywords that are often used in research on obstacle avoidance:-

Real-Time Obstacle Avoidance: Real-time obstacle avoidance refers to a robot's ability to detect and navigate around obstacles as they appear, without pre-planning a path. This capability is crucial for robots operating in dynamic environments where obstacles can suddenly appear or move. The robot uses sensors to continuously monitor its surroundings and adjust its path instantaneously to avoid collisions.

Dynamical Systems in Obstacle Avoidance: A dynamical system in obstacle avoidance is a mathematical framework used to describe the robot's motion. It involves equations that govern the robot's behavior over time, allowing it to react to changes in its environment smoothly and efficiently. This approach ensures that the robot can avoid obstacles by continuously adjusting its trajectory based on the current state of the system.

Cartesian Space: This refers to the robot's movement in a three-dimensional space, defined by x, y, and z coordinates. Obstacle avoidance in Cartesian space involves planning the robot's path directly in this coordinate system, ensuring it doesn't collide with obstacles in its environment.

Joint Space: This involves controlling the robot's individual joints to achieve the desired end-effector position. Obstacle avoidance in joint space means planning the movements of each joint to avoid collisions, which can be more complex but allows for more precise control of the robot's posture.

Concave Obstacle: These are obstacles with inward curves or indentations. Imagine a U-shaped barrier; the inner part of the U is a concave region. Navigating around concave obstacles can be challenging because the robot might get trapped in the concave area if not properly planned.

Convex Obstacle: These are obstacles with outward curves, like a sphere or a cube. Convex obstacles are generally easier for robots to navigate around because there are no indentations where the robot can get stuck. The robot can simply move around the outer boundary of the obstacle.

There has been intense research on obstacle avoidance using artificial potential field and learning based methods. In this thesis we have mainly focused on avoiding obstacles using dynamical systems as it offers several benefits like:-

- **Robustness**: Dynamical systems can handle unexpected changes and disturbances in the environment without needing to replan the entire path.
- **Smooth Motion**: The continuous nature of dynamical systems ensures smooth and natural movements.
- **Real-Time Adaptation**: They allow for real-time adjustments, making the robot highly responsive to dynamic environments

Artifical Potential Method(APF) is widely used for obstacle avoidance. But I have done comparison with modern dynamic based methods I find it lacking the real time obstacle avoidance capbility. Below is the table of comparison of APF with dynamics based methods.

Table 5.1: Comparison of Different Obstacle avoidance Methods

Feature	APF	DMM	ROAM
Smooth Navigation	Struggles with local minima (robot can get stuck)	Ensures continuous motion by modifying velocity	Ensures smooth motion while respecting robot dynamics
Handling Dynamic Obstacles	Difficult without modifications	Can adapt to moving obstacles in real-time	Can handle dynamic obstacles better than DMM
Mathematical Stability	Can cause oscillations near obstacles	Provides stability using modulation	Ensures stability with pseudo-tangents
Implementation Complexity	Simple and easy to implement	Requires computing modulation matrices	More complex due to directional space calculations

5.4.1 Dynamic Modulation Method for Obstacle Avoidance

The **Dynamic Modulation Method** (DMM) is a reactive obstacle avoidance technique that deforms a robot's desired velocity field to generate safe and smooth navigation trajectories. The core idea is to modulate the motion vector so that it lies within the tangent space of nearby obstacles while preserving convergence to the goal.

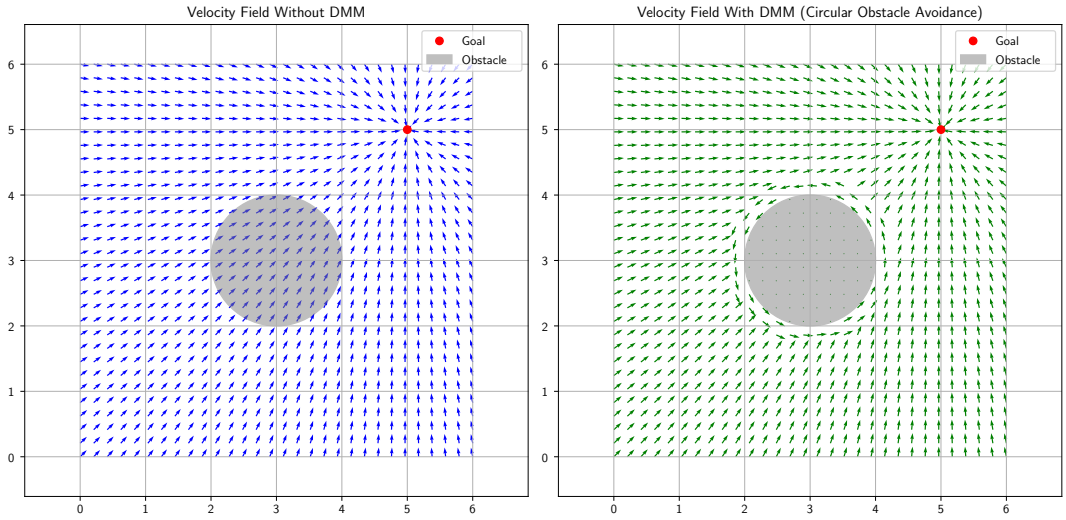


Figure 5.1: Velocity modulation via obstacle surface projection

Assumptions:

- The environment contains N smooth, compact, and convex obstacles, each represented by a differentiable implicit function $\phi_i(\mathbf{x})$, such that the obstacle boundary is $\mathcal{B}_i = \{\mathbf{x} \mid \phi_i(\mathbf{x}) = 0\}$.
- The robot has access to a desired velocity field $\mathbf{V}_d(\mathbf{x})$, e.g., generated from a goal attractor or planner.
- Obstacle avoidance occurs in a local frame assuming no global replanning is required.

Velocity Field Representation:

$$\dot{\mathbf{x}} = \mathbf{V}_d(\mathbf{x}) \in \mathbb{R}^n \tag{16}$$

In the presence of obstacles, the velocity is modulated using a local deformation:

$$\dot{\mathbf{x}}_{\text{mod}} = \mathbf{M}(\mathbf{x})\mathbf{V}_d(\mathbf{x}) \tag{17}$$

where $\mathbf{M}(\mathbf{x}) \in \mathbb{R}^{n\times n}$ is a symmetric, positive semi-definite modulation matrix constructed based on obstacle geometry.

Obstacle Surface Representation: Each obstacle i is defined by a level set:

$$\phi_i(\mathbf{x}) = 0, \quad \text{with} \quad \nabla\phi_i(\mathbf{x}) = \mathbf{n}_i(\mathbf{x}) \tag{18}$$

where $\mathbf{n}_i(\mathbf{x})$ is the normalized surface normal at point $\mathbf{x}$.

Single Obstacle Modulation Matrix: For a single obstacle, the modulation matrix is:

$$\mathbf{M}_i(\mathbf{x}) = \mathbf{I} - (1 - \lambda_i(\mathbf{x}))\mathbf{n}_i(\mathbf{x})\mathbf{n}_i(\mathbf{x})^T \tag{19}$$

where the scalar $\lambda_i(\mathbf{x}) \in [0, 1]$ controls the influence of the modulation:

$$\lambda_i(\mathbf{x}) = \text{clip}\left(\left(\frac{d_i(\mathbf{x})}{d_{s,i}}\right)^2, 0, 1\right) \tag{20}$$

with $d_i(\mathbf{x}) = \|\phi_i(\mathbf{x})\|$ and $d_{s,i}$ the safe distance threshold.

Multiple Obstacle Modulation (Sequential Composition): In environments with multiple obstacles, a sequential application of modulation is used:

$$\dot{\mathbf{x}}_{\text{mod}} = \mathbf{M}_N(\mathbf{x}) \cdots \mathbf{M}_2(\mathbf{x})\mathbf{M}_1(\mathbf{x})\mathbf{V}_d(\mathbf{x}) \tag{21}$$

Alternatively, the modulation matrices can be combined via weighted blending:

$$\mathbf{M}_{\text{blend}}(\mathbf{x}) = \sum_{i=1}^{N} w_i(\mathbf{x})\mathbf{M}_i(\mathbf{x}) \tag{22}$$

with normalized weights:

$$w_i(\mathbf{x}) = \frac{\gamma_i(\mathbf{x})^{-1}}{\sum_{j=1}^{N} \gamma_j(\mathbf{x})^{-1}}, \quad \gamma_i(\mathbf{x}) = d_i(\mathbf{x}) + \epsilon \tag{23}$$

where ϵ is a small constant to prevent division by zero.

Geometric Intuition:

- When $\mathbf{x}$ is far from all obstacles, $\lambda_i(\mathbf{x}) \to 0$ and $\mathbf{M}(\mathbf{x}) \approx \mathbf{I}$.

- When $\mathbf{x}$ is near an obstacle, $\lambda_i(\mathbf{x}) \to 1$, and the component of $\mathbf{V}_d$ in the direction of $\mathbf{n}_i$ is suppressed.
- The tangent motion dominates, leading to a natural flow around the obstacle surface.

Stability Guarantee: As shown in [19], when the original system $\dot{\mathbf{x}} = \mathbf{V}_d(\mathbf{x})$ is globally asymptotically stable at a point $\mathbf{x}^*$ and obstacles are properly modeled, the modulated system retains asymptotic convergence toward $\mathbf{x}^*$ while avoiding all obstacles.

Below is the algorithm for multiple obstacle avoidance using DMM. Here Lidar is used for obstacle detection and PID is used as controller.

Algorithm 3 DMM-Based Obstacle Avoidance with Multiple Obstacles

Current velocity $\mathbf{v}$, LiDAR data $L = \{l_i\}$, orientation quaternion q
Modified velocity $\mathbf{v}'$
function DMM_MultiObstacle_Avoidance($\mathbf{v}, L, q$)
  Extract yaw angle θ from quaternion q
  Initialize empty list of obstacle normals $\mathcal{N} = []$, distances $\mathcal{D} = []$
  for each LiDAR point $l_i \in L$ **do**
    Extract range r_i and angle α_i
    Transform to local coordinates: $\mathbf{p}_i = (r_i \cos\alpha_i, r_i \sin\alpha_i)$
    if $r_i < d_{\text{safe}}$ **then**
      Compute normal: $\mathbf{n}_i = \frac{\mathbf{p}_i}{\|\mathbf{p}_i\|}$
      Append $\mathbf{n}_i$ to $\mathcal{N}$ and r_i to $\mathcal{D}$
    end if
  end for
  if $\mathcal{N}$ is not empty **then**
    Initialize blended modulation matrix: $\mathbf{M}_{\text{blend}} \leftarrow \mathbf{0}_{2\times 2}$
    for $i = 1$ to $|\mathcal{N}|$ **do**
      Let $\mathbf{n}_i = \mathcal{N}[i]$, $d_i = \mathcal{D}[i]$
      Compute weight: $w_i = \frac{1}{d_i + \epsilon}$
      Compute $\lambda_i = \text{clip}\left(\left(\frac{d_i}{d_{\text{safe}}}\right)^2, 0, 1\right)$
      Compute modulation matrix: $\mathbf{M}_i = \mathbf{I} - (1 - \lambda_i)\mathbf{n}_i\mathbf{n}_i^\top$
      $\mathbf{M}_{\text{blend}} \leftarrow \mathbf{M}_{\text{blend}} + w_i \cdot \mathbf{M}_i$
    end for
    Normalize weights: $\mathbf{M}_{\text{blend}} \leftarrow \frac{\mathbf{M}_{\text{blend}}}{\sum w_i}$
    Compute modulated velocity: $\mathbf{v}' \leftarrow \mathbf{M}_{\text{blend}} \cdot \mathbf{v}$
  else
    $\mathbf{v}' \leftarrow \mathbf{v}$ {No nearby obstacles}
  end if
  Publish or return modified velocity $\mathbf{v}'$
end function

5.4.2 Rotational Obstacle Avoidance Method

The **Rotational Obstacle Avoidance Method (ROAM)** is a real-time, dynamic motion planning algorithm designed for navigating autonomous agents through complex environments populated with static or dynamic obstacles. Unlike traditional trajectory planners that compute a full path ahead of time, ROAM adopts a local, velocity-based strategy where the robot continuously adapts its motion based on perceived surroundings.

At the heart of ROAM lies the concept of velocity field modulation. A baseline converging dynamical system towards the goal is defined as:

$$\dot{\xi} = f(\xi), \tag{24}$$

where $\xi \in \mathbb{R}^n$ denotes the agent's position and $f(\xi)$ represents the nominal velocity field guiding the agent to the goal. In the presence of obstacles, ROAM rotates velocity field into a safe direction. This rotation ensures smooth avoidance by projecting the desired motion onto the local tangent space of obstacles while scaling the motion near boundaries to enforce safety.

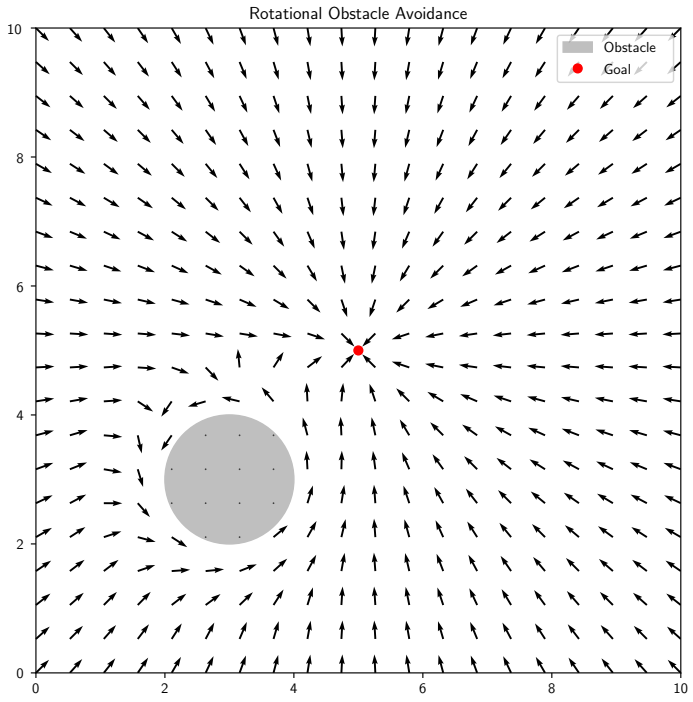


Figure 5.2: Rotational Obstacle Avoidance

Comparison with Dynamic Modulation Method (DMM)

Although ROAM and the *Dynamic Modulation Method (DMM)* share similar foundations — both modulate a nominal velocity field to achieve obstacle avoidance — they differ in design philosophy and

computational structure:

- **Geometric vs. Rotational Modulation Focus:** DMM constructs the modulation matrix by explicitly using geometric features such as the surface normal and tangent basis derived from implicit obstacle representations. ROAM, in contrast, focuses on rotating the convergence direction to avoid obstacles while preserving goal-directed behavior. Rather than modulating repulsion geometrically, ROAM modifies the vector field orientation smoothly within the tangent space.

- **Reactivity vs. Rotation-Informed Flow:** While DMM emphasizes modulation via orthogonal decomposition and scaling of eigenvalues (often leading to sharp redirection), ROAM applies smooth rotational transformations to blend between the convergence vector and an obstacle-aware pseudo-tangent direction, leading to more fluid trajectories. ROAM is not merely reactive but structurally modifies the dynamics based on obstacle proximity and orientation.

- **Modulation Design Philosophy:** DMM relies on constructing a modulation matrix using a basis formed by the normal and tangent directions, enforcing strict repulsion via eigenvalue scaling. ROAM, on the other hand, introduces a rotation angle between the convergence direction and the modulated vector, ensuring motion stays within the tangent plane and avoids obstacle penetration while allowing for graceful redirection.

- **Multi-Obstacle Handling:** DMM often includes explicit combination rules for handling multiple obstacles, usually by modulating the final velocity based on distance-weighted superposition. ROAM, due to its rotational nature, can blend rotations associated with each obstacle, resulting in a composite flow field that still respects the tangent constraints and avoids abrupt discontinuities.

In summary, DMM is highly geometric and modulation-centric, ideal for applications requiring formal guarantees and strong obstacle avoidance behavior. ROAM leverages rotational dynamics to maintain smooth and goal-consistent navigation, often producing more natural paths, especially in cluttered environments.

Below is the description of mathematical formulation for modifying the original convergence dynamics $c(\xi)$ to ensure that it remains in a safe, constrained direction. By blending the reference direction $r(\xi)$ and convergence dynamics $c(\xi)$, and solving for the correct weighting factor b, the method ensures that the final motion follows a proper geometric structure, avoiding unwanted behavior such as direct collisions with obstacles.

5.4.2.1 Directional Space and Directional Weighted Mean

The purpose of the directional space transformation is to project a global direction vector $\mathbf{v}_i \in \mathbb{R}^N$ back into the tangent space orthogonal to a reference direction $\mathbf{b} \in \mathbb{R}^N$, typically representing the obstacle normal. This transformation reduces the dimensionality of the problem from N to $N-1$, providing a compact representation of the directional deviation from the reference.

Let:

- $\mathrm{v}_i \in \mathbb{R}^N$ be a unit velocity vector representing a direction.

- $\mathbf{b} \in \mathbb{R}^N$ be the reference direction, typically the normalized obstacle normal.
- $B \in \mathbb{R}^{N \times N}$ be a basis matrix where:
  - The first column is $\mathbf{b}$
  - The remaining $N-1$ columns form an orthonormal basis of the null space orthogonal to $\mathbf{b}$
- $_i = \mathrm{B}^\top \mathbf{v}_i$ be the transformed velocity vector in the new basis.

The transformation into direction space K :

$$K = \{\kappa_i \in \mathbb{R}^{N-1} where \|\kappa_i\| < \pi\} \tag{25}$$

The magnitude of $\hat{v}_i$ is equal to angle between the original vector and reference vector.
The Transformation of initial vector in direction space is : [20]

$$\kappa_i(b) = k(\mathbf{v}_i, b) = \begin{cases} \arccos\left(\hat{v}_{i[1]}\right) \dfrac{\hat{v}_{i[2:]}}{\|\hat{v}_{i[2:]}\|}, & \text{if } \hat{v}_{i[1]} \neq 1 \\ 0, & \text{if } \hat{v}_{i[1]} = 1 \end{cases} \tag{26}$$

Where: $\hat{v}_{i[1]}$ is the angle between the reference vector and transformed vector $\hat{v}_i$
$\arccos(\hat{v}_i[1]))$ gives the angle θ between the original vector and the reference direction.
$\frac{\hat{v}_i[2:]}{\|\hat{v}_i[2:]\|}$) is a normalized component of $\hat{v}_i$ excluding the first element.
If $\hat{v}_i[1] = 1$:
The function directly returns 0, meaning no deviation from the reference direction.

The function $\kappa_i(b) = k(v_i, b)$ computes a directional weighting factor based on the transformed vector $\hat{v}_i$.

The Mean Directional Space Vector for N Obstacles is given by:

$$\bar{\kappa} = \sum_{i=1}^{N_v} w_i \kappa_i \tag{27}$$

where w_i depends upon the distance from obstacle.

5.4.2.2 Inverse Directional Space

Directional Space Transformation: As defined in Section 5.4.2.1, the directional space vector $\kappa_i \in \mathbb{R}^{N-1}$ is given by:

$$\kappa_i(\mathbf{b}) = k(\mathbf{v}_i, \mathbf{b}) = \begin{cases} \arccos\left(\hat{v}_{i[1]}\right) \dfrac{\hat{\mathbf{v}}_{i[2:]}}{\|\hat{\mathbf{v}}_{i[2:]}\|}, & \text{if } \hat{v}_{i[1]} \neq 1 \\ 0, & \text{if } \hat{v}_{i[1]} = 1 \end{cases} \tag{28}$$

Inverse Directional Space: To recover the global direction vector from directional space $\kappa_i \in \mathbb{R}^{N-1}$, we perform the inverse transformation:

1. Compute the angle

$$\theta = \|\kappa_i\| \tag{29}$$

2. Define the first element of $\hat{\mathbf{v}}_i \in \mathbb{R}^N$ as:

$$\hat{v}_{i[1]} = \cos(\theta) \tag{30}$$

3. The remaining components are given by:

$$\hat{\mathbf{v}}_{i[2:]} = \sin(\theta) \cdot \frac{\kappa_i}{\|\kappa_i\|} \tag{31}$$

4. Form the full vector:

$$\hat{\mathbf{v}}_i = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \cdot \frac{\kappa_i}{\|\kappa_i\|} \end{bmatrix} \tag{32}$$

5. Apply the inverse transformation using the basis matrix B:

$$\mathbf{v}_i = B\hat{\mathbf{v}}_i \tag{33}$$

Basis Matrix Construction: The matrix $B \in \mathbb{R}^{N \times N}$ is constructed such that:

- The first column is $\mathbf{b}$
- The second column is a unit vector $\mathbf{u}$ orthogonal to $\mathbf{b}$
- The third column is $\mathbf{b} \times \mathbf{u}$, ensuring a right-handed orthonormal basis

These columns are normalized to ensure orthogonality and unit length.

Normalization: Finally, the recovered vector $\mathbf{v}_i$ is normalized to ensure:

$$\|\mathbf{v}_i\| = 1$$

This guarantees that the direction vector remains on the unit sphere in $\mathbb{R}^N$.

5.4.2.3 Pseudo Tangent Direction

The desired pseudo tangent direction $e(\xi)$ is obtained by rotating the convergence dynamics $C(\xi)$ away from the reference direction. [18]

Desired Modification The desired pseudo-tangent vector $e(\xi)$ is obtained by rotating the original convergence dynamics $c(\xi)$ away from the reference direction $r(\xi)$ until it lies in the tangent plane. [20] $r(\xi)$ is a reference direction, possibly aligned with Obstacle's surface. Instead of moving directly along $c(\xi)$, we modify it so that it conforms to constraints in the angular space. [18]

Understanding Angular Space and Tangent Hyper-Sphere The angular space is $k(n, \cdot)$ In this space, any tangent vector is always at a fixed distance $R_e = \frac{\pi}{2}$ from the normal direction. The set of all tangent directions forms a hyper-sphere in the direction space. [20]

The key point here is that the transformation of $c(\xi)$ must be such that it respects this geometric structure. Thus, to rotate $c(\xi)$ away from $r(\xi)$, we intersect the line connecting $r(\xi)$ and $c(\xi)$ with a circle

of radius $R_e \in [\pi/2, \pi]$. [18] [20]

Constraints for Obtaining $e(\xi)$

The pseudo-tangent vector $e(\xi)$ is computed using the following conditions:

$$\text{if } \|k(-n, c)\| \geq R_e, \text{ then } e = c \qquad [18] \tag{34}$$

This means that if the magnitude of $k(-n, c)$ is already large enough (i.e., at least R_e), then no adjustment is needed, and we set $e = c$. Otherwise, we need to blend $c(\xi)$ and $r(\xi)$ using a weighted sum:

$$k(-n, e) = (1 - b)k(-n, r) + bk(-n, c) \qquad [18] \tag{35}$$

where:

- $k(-n, e)$ is the transformed direction after applying the correction.
- $(1 - b)$ and b are weights that mix the reference direction r and the original convergence direction c.
- $b \in \mathbb{R}_{>0}$ (a positive real number).

The final condition ensures that the transformation stays within the required geometric limits:

$$\|k(-n, e)\| = R_e \tag{36}$$

This means the final adjusted vector $e(\xi)$ must lie at exactly the required distance R_e from the normal.

Solving for b Since the equation 36 involves a quadratic dependence on b, the parameter b can be found by solving a quadratic equation.

Interpretation and Impact on Behavior The modification ensures that $e(\xi)$:

- **Lies in the tangent plane**: Instead of following $c(\xi)$ directly, it respects the geometric constraints.
- **Avoids obstacles**: The correction ensures that the direction of motion is never directly toward an obstacle.
- **Maintains smooth motion**: The transition between $c(\xi)$ and $e(\xi)$ is continuous, preventing abrupt changes in motion.
- **Handles special cases properly**:
  - If $c(\xi)$ and $r(\xi)$ coincide, the correction is straight forward.
  - If they are significantly different, the method finds an appropriate blend.

5.4.2.4 Rotation Towards Tangent Direction

Rotate initial dynamics $f(\xi)$ towards the peudo tangent direction. In this step, the initial dynamics $f(\xi)$

is rotated towards the pseudo tangent $e(\xi)$. This rotation operation is performed in the direction space with respect to the convergence dynamics $c(\xi)$: [20]

$$\dot{\xi} = k^{-1}\Big(c,\ \big(1-\lambda(\xi)\big)\,k\big(c,f\big)\ +\ \lambda(\xi)\,k\big(c,e\big)\Big) \qquad [18] \tag{37}$$

where k^{-1} is conversion from lower dimensional space to higher dimensional space i.e from directional space to original vector space.

$$\lambda(\xi) = \left(\frac{1}{\Gamma(\xi)}\right)^{q} \qquad [18] \tag{38}$$

$$R^{r} = \min\Big(R^{e} - \|k(-n,\,r)\|,\ \tfrac{\pi}{2}\Big) \qquad [18] \tag{39}$$

$$q = \max\Big(1,\ \tfrac{R^{r}}{\Delta k(c)}\Big)^{s} \qquad [18] \tag{40}$$

$$\Delta k(c) = \big\|\, k(-n,\,r)\ -\ k(-n,\,c)\ \big\| \qquad [18] \tag{41}$$

where the rotation weight $\lambda(\xi)$ is determined based on the inverse of the distance $\Gamma(\xi)$. This ensures the rotation has a decreasing influence as the distance increases, allowing for a smooth transition in the avoidance behavior. Additionally, the smoothing factor q is introduced, which gradually decreases to zero when the convergence dynamics vanish around the robot. This effectively cancels the rotation avoidance effect in regions where the tangent $e(\xi)$ is not defined.

5.4.2.5 Evaluation of Speed

The directional space mapping $k(\cdot)$ and its inverse mapping $k^{-1}(\cdot)$ operate on unit vectors in the original space. As a result, the algorithm modifies the *direction* of the initial dynamics, rather than its magnitude. [20] To incorporate magnitude control in a decoupled manner, we introduce an additional component using $h(\xi) : \mathbb{R}^n \to [0,1]$. [18] Formally,

$$\|\dot{\xi}\| \ =\ h(\xi)\,\|f(\xi)\| \qquad [18] \tag{42}$$

The stretching factor $h(\xi)$ is designed to slow down the motion when pointing towards an obstacle, and the effect decreases with increasing distance from the obstacle. We define:

$$h(\xi) \ =\ \min\left(1,\ \frac{\big\|\Delta k(c)\big\|}{R^{r}},\ \left(1-\frac{1}{\Gamma(\xi)}\right)^{2}\right) \qquad [18]\ [20] \tag{43}$$

5.4.2.6 Weighted Global Velocity Computation from Multiple Obstacles

When multiple obstacles influence a robot's motion, each obstacle contributes to the global velocity vector based on its proximity. Closer obstacles are more important and thus are given higher weights. This is achieved by assigning each obstacle a weight inversely proportional to its distance. The final global velocity is then computed as the weighted average of the individual velocity contributions from each

obstacle.

Let:

- N be the total number of obstacles,
- d_i be the distance to the i^{th} obstacle,
- $w_i = \frac{1}{d_i}$ be the weight assigned to the i^{th} obstacle,
- $\vec{v}_i = \begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix}$ be the velocity contribution from the i^{th} obstacle,
- $\vec{v}_{\text{global}}$ be the resulting global velocity vector.

Then the global velocity is given by:

$$\vec{v}_{\text{global}} = \begin{cases} \dfrac{\sum_{i=1}^{N} w_i \vec{v}_i}{\sum_{i=1}^{N} w_i}, & \text{if } \sum_{i=1}^{N} w_i \neq 0 \\ \displaystyle\sum_{i=1} w_i \vec{v}_i, & \text{otherwise} \end{cases} \tag{44}$$

$$\vec{v}_i = \begin{bmatrix} v_{ix} \\ v_{iy} \end{bmatrix}, \quad \vec{v}_{\text{global}} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{45}$$

This formulation ensures that obstacles closer to the robot exert a stronger influence on the robot's final movement direction.

Convergence vector $c(\xi)$ For this study $c(\xi)$ is chosen to be

$$c(\xi) = \xi - \xi^a \tag{46}$$

where ξ^a is the attractor point.

Reference vector r

$$r = p_{\text{obs}} - p_{\text{current}} = \begin{bmatrix} x_{\text{obs}} \\ y_{\text{obs}} \\ z_{\text{obs}} \end{bmatrix} - \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_{\text{obs}} - x \\ y_{\text{obs}} - y \\ z_{\text{obs}} - z \end{bmatrix} \tag{47}$$

where

p_{obs} represents position of obstacle

p_{current} represents current position of robot

Algorithm 4 Rotational Obstacle Avoidance (ROAM) Algorithm

Current position $P = (x, y)$ and orientation θ, PID velocity $v = (v_x, v_y)$ and convergence vector $c = (-e_x, -e_y, 0)$, LiDAR data L, waypoints, safe distance d_{safe}, ROAM params R_e, γ

Modified control commands (thrust and fin position)

function ROAM_AVOIDANCE(P, v, L)

$O \leftarrow$ extract_obstacles(L)

Compute convergence vector c from desired minus current position

if $O \neq \emptyset$ **then**

$n \leftarrow -\big($normal of first obstacle$\big)$

for each obstacle $o_i \in O$ **do**

$r \leftarrow \big(o_i.\text{center}_x - x,\ o_i.\text{center}_y - y,\ 0\big)$

$e \leftarrow$ COMPUTEPSEUDOTANGENT(n, c, r, R_e)

$k \leftarrow$ ROTATECONVERGENCE$\big($direction_space(c, c), direction_space(c, e), γ, R_e, direction_space(n, r), direction_space(n, c), $c\big)$

$s \leftarrow$ SPEED_SCALING$\big(R_e$, γ, direction_space(n, r), direction_space$(n, c)\big)$

$v' \leftarrow s \cdot k$

Weight v' by $1/\text{dist}(o_i, P)$ and accumulate into overall v'

end for

if $w_{\text{sum}} \neq 0$ **then**

$v' \leftarrow \left(\frac{x_{\text{sum}}}{w_{\text{sum}}}, \frac{y_{\text{sum}}}{w_{\text{sum}}}\right)$

else

$v' \leftarrow (x_{\text{sum}}, y_{\text{sum}})$

end if

$\theta' \leftarrow \arctan\big(v'_y / v'_x\big)$

Clamp and normalize v' and compute thrust/fin from (v', θ')

Publish control commands

else

Thrust $\leftarrow v_x$, fin $\leftarrow \arctan(v_y / v_x)$

Publish PID control commands

end if

end function

6 Implementation

6.1 Control of Underactuated Robot

The AUV used to verify earlier introduced mathematical concepts is underactuated. With one reversible propeller in x direction. For yaw and pitch control fins have been used.

Cascaded control offers a structured way to manage these challenges by decomposing the control problem into two nested loops. The *outer loop* regulates the vehicle's position by comparing the desired position p_d to the measured position p, and uses a PID regulator to generate a commanded velocity v_d. The *inner loop* then regulates velocity: it compares v_d to the measured velocity v and uses a second PID to produce a desired acceleration a_d. This separation simplifies tuning—each loop handles one integration order of the vehicle's kinematics—and improves robustness by limiting the bandwidth of each controller.

To handle the AUV's nonlinear dynamics in the inner loop, we employ *feedback linearization*. Given the standard rigid-body model

$$M(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + D(q,\dot{q})\,\dot{q} + g(q) \;=\; \tau \tag{48}$$

where q is the configuration vector, M the inertia matrix, C the Coriolis/centripetal terms, D the hydrodynamic damping, and g the restoring forces, we choose

$$\tau \;=\; M(q)\,a_d \;+\; C(q,\dot{q})\,\dot{q} \;+\; D(q,\dot{q})\,\dot{q} \;+\; g(q) \tag{49}$$

This exact inversion cancels all nonlinearities, yielding the linear input–output relationship $\ddot{q} = a_d$. The inner PID then effectively controls a double-integrator, vastly simplifying stability and performance analysis.

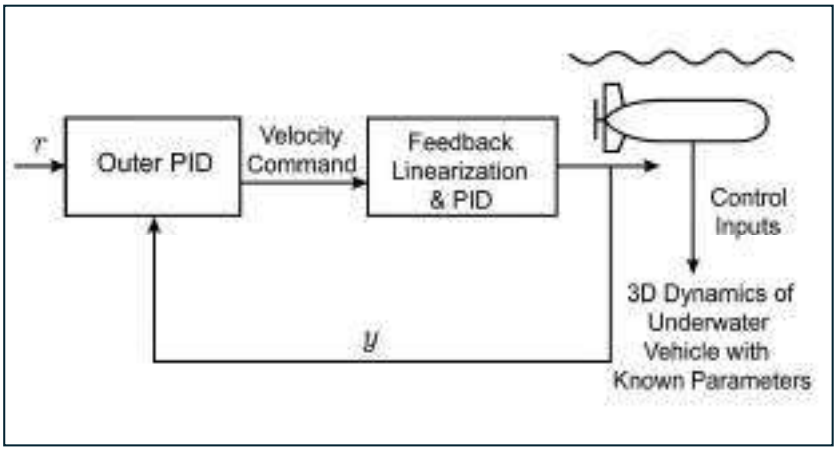


Figure 6.1: AUV State Control Flow-chart

In short, the cascaded PID structure decouples position and velocity regulation, while feedback linearization cancels the AUV's nonlinear terms. The result is a controller that is both easy to tune and capable of high-performance trajectory tracking, even in the presence of strong hydrodynamic effects.

6.2 Controller Design

AUV Parameters Physical Properties

- Mass of AUV: $m = 148.3571$ kg
- Neutral buoyancy assumed: $W = B$
- Propeller diameter: $d_p = 0.2$ m

State Variables

- u: Linear velocity in surge (x-axis)
- q: Angular velocity in pitch (y-axis)
- r: Angular velocity in yaw (z-axis)
- θ: Pitch angle
- ψ: Yaw angle

Table 6.1: AUV Parameters

AUV Parameters	
Inertia	
I_{xx}	$3.0000\ \text{kg} \cdot \text{m}^2$
I_{yy}	$41.980233\ \text{kg} \cdot \text{m}^2$
I_{zz}	$41.980233\ \text{kg} \cdot \text{m}^2$
Added Mass	
$X_{\dot{u}}$	-4.876161
$M_{\dot{q}}$	-33.46
$N_{\dot{r}}$	-33.46
Hydrodynamic Drag	
$X_{\lvert u\rvert u}$	-6.2282
$M_{\lvert q\rvert q}$	-632.698957
$N_{\lvert r\rvert r}$	-632.698957

Dynamic Equations (3 DOF)

Let the total surge, pitch, and yaw inertias be:

$$m_u = m - X_{\dot{u}} = 153.2333\ \text{kg} \tag{50}$$

$$I_{yy}^* = I_{yy} - M_{\dot{q}} = 75.440233\ \text{kg} \cdot \text{m}^2 \tag{51}$$

$$I_{zz}^* = I_{zz} - N_{\dot{r}} = 75.440233\ \text{kg} \cdot \text{m}^2 \tag{52}$$

Surge

$$m_u \dot{u} = -m_u u r + X_{|u|u}|u|u + T \tag{53}$$

Pitch

$$I_{yy}^* \dot{q} = M = M_{\text{hydro}}(\delta_p) \tag{54}$$

Yaw

$$I_{zz}^{*}\dot{r} = N = N_{\text{hydro}}(\delta_y) \tag{55}$$

Under neutral buoyancy and symmetric design, restoring moments are negligible.

Feedback Linearization

Surge (u)

$$f_u(u,r) = -m_u ur + X_{|u|u}|u|u \tag{56}$$

$$\dot{u} = \frac{1}{m_u}\left(T + f_u(u,r)\right) \tag{57}$$

Define virtual control v_u:

$$\dot{u} = v_u \quad \Rightarrow \quad T = m_u v_u - f_u(u,r)$$

Pitch (q)

$$f_q(q) = M_{|q|q}|q|q \tag{58}$$

$$\dot{q} = \frac{1}{I_{yy}^{*}}\left(\tau_{pitch} + f_q(q)\right) \tag{59}$$

Define virtual control v_q:

$$\dot{q} = v_q \quad \Rightarrow \quad \tau_{pitch} = I_{yy}^{*} v_q - f_q(q) \tag{60}$$

Yaw (r)

$$f_r(u,r) = m_u ur + N_{|r|r}|r|r \tag{61}$$

$$\dot{r} = \frac{1}{I_{zz}^{*}}\left(\tau_{yaw} - f_r(u,r)\right) \tag{62}$$

Define virtual control v_r:

$$\dot{r} = v_r \quad \Rightarrow \quad \tau_{yaw} = I_{zz}^{*} v_r + f_r(u,r) \tag{63}$$

PID Controller on Virtual Inputs

Surge PID

$$v_u = K_{p_u}(u_d - u) + K_{i_u}\int (u_d - u)\,dt + K_{d_u}(\dot{u}_d - \dot{u}) \tag{64}$$

Pitch PID

$$v_q = K_{p_q}(q_d - q) + K_{i_q}\int (q_d - q)\,dt + K_{d_q}(\dot{q}_d - \dot{q}) \tag{65}$$

Yaw PID

$$v_r = K_{p_r}(r_d - r) + K_{i_r}\int (r_d - r)\,dt + K_{d_r}(\dot{r}_d - \dot{r}) \tag{66}$$

Final Control Inputs

- Propeller Thrust:

$$T = m_u v_u + m_u ur - X_{|u|u}|u|u \tag{67}$$

- Pitch Moment via fins:

$$\tau_{pitch} = I_{yy}^{*} v_q - M_{|q|q}|q|q \tag{68}$$

- Yaw Moment via fins:

$$\tau_{yaw} = I_{zz}^{*} v_r - m_u u r - N_{|r|r}|r|r \tag{69}$$

Notes

- Restoring moments are neglected due to neutral buoyancy and symmetric configuration.
- PID gains (K_p, K_i, K_d) should be tuned based on desired tracking performance.

6.3 Obstacle Detection and Processing

I have used a lidar sensor to detect obstacles. I will explain the lidar data processing through modules namely `extract_obstacles` and `compute_obstacle_properties`. Where `extract_obstacles` is responsible for checking the presence of obstacle and distance from agent. While second function is responsible to calculate obstacle properties like normal vector, obstacle center.

6.3.1 Extracting Obstacles

The module `extract_obstacles` processes LiDAR data to identify and extract obstacles. The main steps are as follows:

1. Initialize an empty list of obstacles and a temporary list for the current obstacle.
2. Iterate through the LiDAR data points.
3. For each data point, check if the difference in distance from the previous point exceeds a given threshold.
4. If the threshold is exceeded and the current obstacle has more than two points, compute its properties and add it to the list of obstacles.
5. Clear the current obstacle list and start a new one.
6. Convert each data point to an angle-distance pair and add it to the current obstacle list.
7. After processing all data points, perform a final check on the last obstacle and add it to the list if it meets the criteria.

6.3.2 Computing Obstacle Properties

The module `compute_obstacle_properties` calculates the properties of an obstacle based on its points. The main steps are:

1. Compute the mean distance of the obstacle points.

2. Fit a line to the points using the least squares method to determine the obstacle's orientation.

3. Calculate the center of the obstacle in Cartesian coordinates.

4. Compute the normal vector to the obstacle's surface.

6.3.3 AUV state control With Obstacle Avoidance

As we are working on modulating dynamics based on obstacles. So here is the how control flow looks like with obstacle avoidance:-

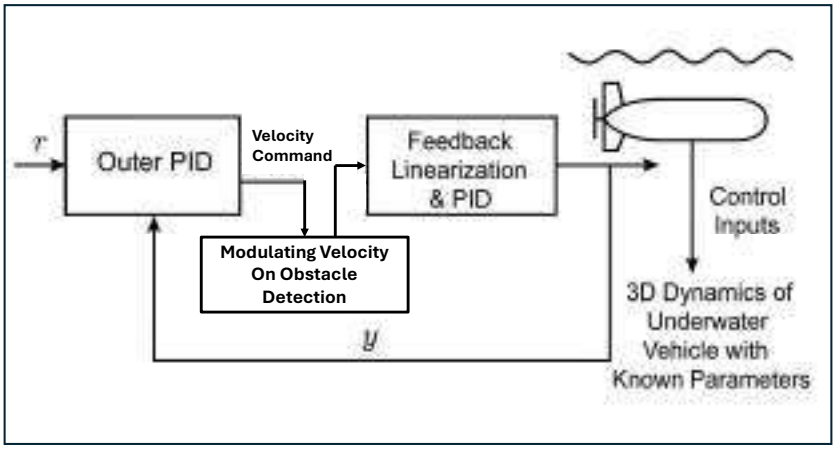


Figure 6.2: Control Flow with Obstacle Avoidance

6.4 Environment Setup

In this section, we describe the simulation environment that integrates Gazebo with ROS2 to implement a multi-agent framework with obstacle avoidance for marine robots.

Justification for Using Gazebo: Gazebo is chosen as the simulation platform due to its realistic physics, robust ROS2 integration, and extensive sensor support. Its ability to simulate complex environmental effects—such as waves, currents, and drag forces—provides a highly realistic testing ground for evaluating the performance of our autonomous systems.

Simulation World: The simulation world is designed to replicate an underwater and surface environment with carefully modeled physical properties.This includes factors like fluid dynamics and environmental disturbances to mimic real-world underwater conditions.6.1

Table 6.2: Simulation Parameters

Parameter	Value/Description
Water Density	1025 kg/m^3
Drag Coefficient	0.8
Gravity	9.81 m/s^2
Gazebo Version	Gazebo Harmonic
ROS2 Version	ROS 2 Humble

Robot Model: The robot models, such as AUVs and ASVs, are constructed using SDF/URDF. These models incorporate detailed representations of physical dimensions, mass properties, and sensor placements, ensuring accurate simulations of robot dynamics and interactions. [21]

Physics Engine: For physics simulation, we employ the Open Dynamics Engine (ODE), which provides realistic dynamics and collision handling essential for underwater simulations.

Sensor Models: The environment includes multiple sensor models (IMU, camera, LiDAR,GPS and Compass) integrated via ROS2 topics. This setup ensures comprehensive perception data for tasks such as navigation and obstacle avoidance.

Visualization Tools: The simulation setup is complemented by several visualization tools:

- ROS2 Node Graph using `rqt_graph`
- Gazebo World Screenshots
- RViz visualization of the robot and sensor data
- Velocity/Trajectory Graphs for performance analysis

ROS2 Communication: The simulation leverages ROS2 [22] for inter-node communication. The table below outlines the key nodes and their associated topics for task allocation:

Table 6.3: ROS2 Communication Overview

Node	Subscribed Topics	Published Topics
`Decision_Making`	/Task_Details /global/allocated_tasks /global/aggregated_task_allocations	/global/allocated_tasks /global/task_allocation_discussion
`Aggregator_node`	/global/task_allocation_discussion	/global/aggregated_task_allocations
`Task_publisher`	—	/Task_Details

7 Results and Analysis

7.1 Multi-Agent Task Allocation

There are 8 underwater vehicles and 1 surface vehicle. Surface vehicle is the vehcile that receives information from outer world and accordingly it floats tasks. There may be less number of tasks than the agents or it's opposite. The agents needs to discuss between themselves and decide which one task will be done by which agents. The below figure shows the flow of information between agents:-

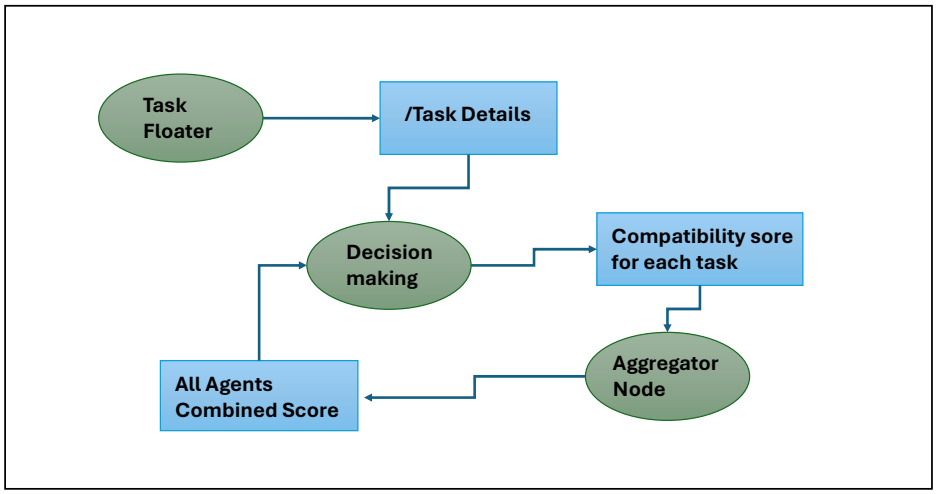


Figure 7.1: Task Allocation Flowchart

The `Task Floater` node will publish task details on the */Task_Details* topic. The `Decision Making` node of each agent will subscribe to this topic and will publish a compatibility score for each task on */global/task_allocation_discussion*. This topic will contain scores for each task from each agent, but the messages will arrive at different timestamps.

To consolidate these into a complete scorecard, the `Aggregator` node comes into play. It synchronizes the data to a common timestamp and publishes the aggregated scores to */global/aggregated_task_allocations*. Each agent's `Decision Making` node then subscribes to this topic, checks whether it is the best fit for any task, and if so, publishes the result to */global/allocated_tasks*.

Finally, the `Navigation` node of the selected agents receives the assigned task information and begins execution.

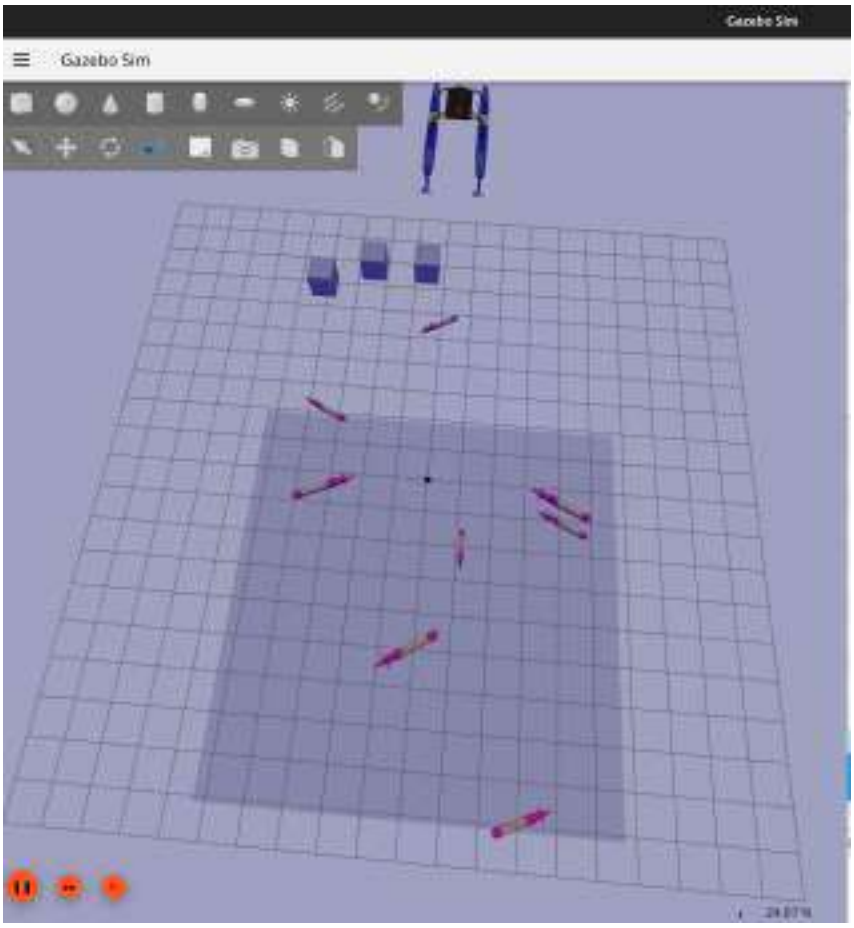


Figure 7.2: Gazebo World

Problem Definition: There are 8 agents and 4 tasks will be floated. Now agents need to communicate between themselves to decide which agent is going to do which task based on their capability.

Table 7.1: Current Position of agents

Agent Name	Position	Yaw
my_lrauv_1	(-4.99,0,-2)	2.12
my_lrauv_2	(0,-5,-2)	1.12
my_lrauv_3	(5,5,-6)	1.12
my_lrauv_4	(0,4,-2)	-1.15
my_lrauv_5	(-11,-4,-5)	-1.15
my_lrauv_6	(0,-6,-5)	1.12
my_lrauv_7	(12,0,8.5)	1.98
my_lrauv_8	(0,-2,-11)	(3.12)

Table 7.2: Task Details

Task Number	Goal Position
1	(20,10,-6)
2	(20,105,-9)
3	(100,20,-20)
4	(25,30,-8)

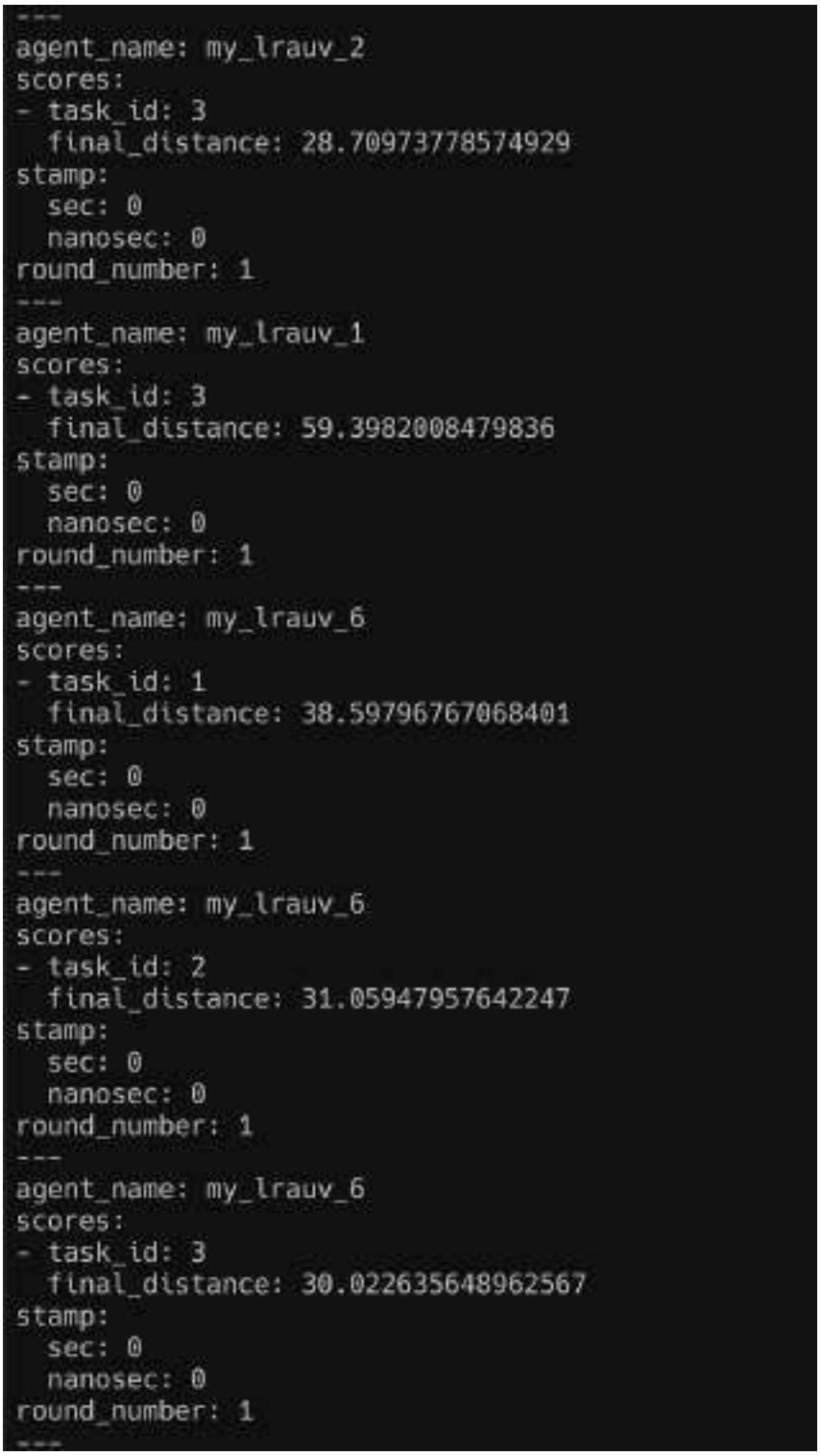


Figure 7.3: Task Allocation Discussion

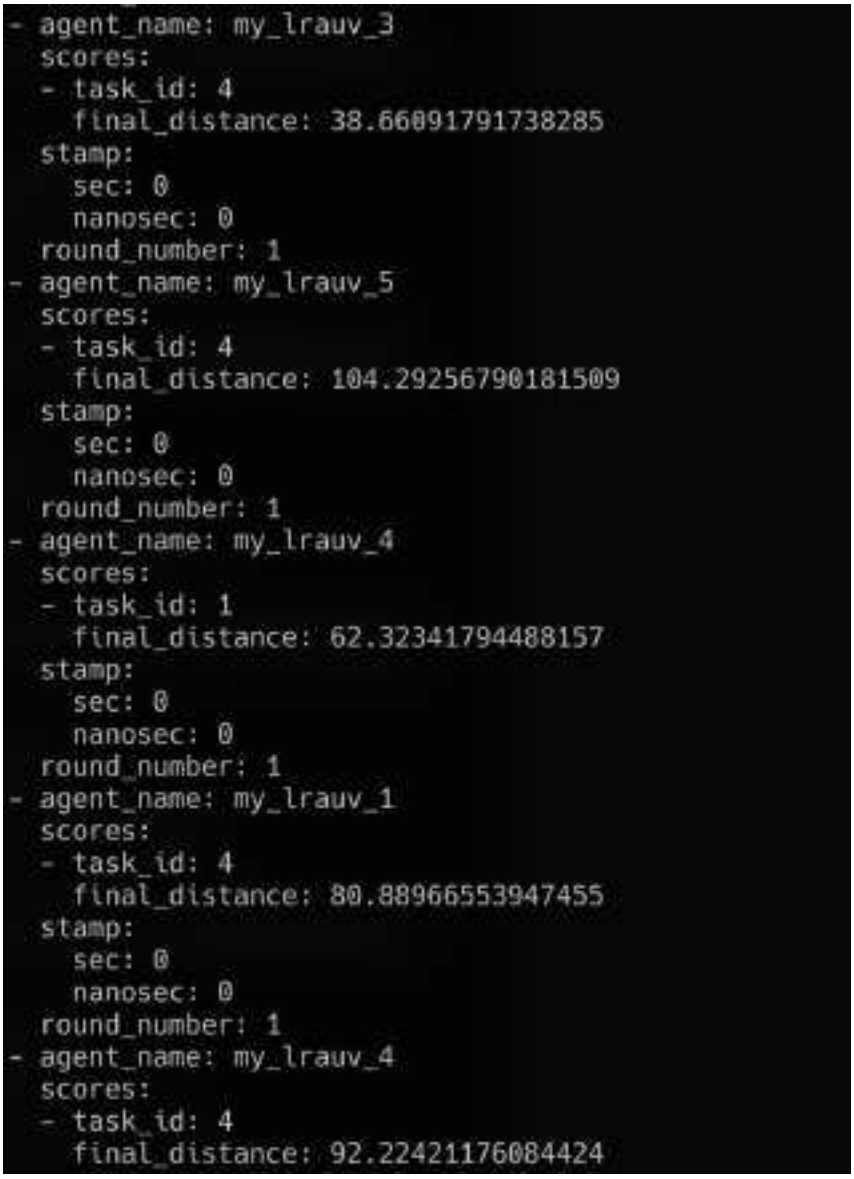


Figure 7.4: Aggregated Task Allocation

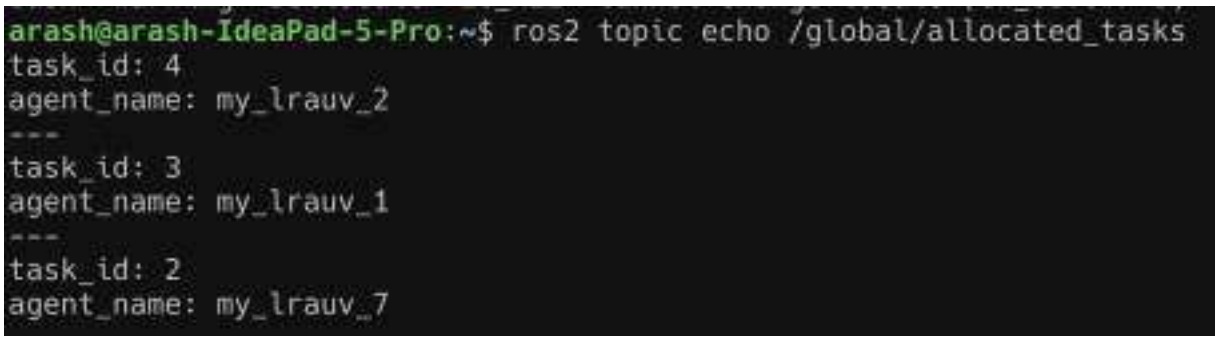


Figure 7.5: Final Decision

7.2 Dynamic Modulation Matrix

Problem Description:- I will be demonstrating cases of testing this obstacle avoidance method. A cube-shaped obstacle is used. First the AUV is given some desired goal and it's trajectory is traced. Then, the obstacle is placed in between trajectory. Then, the trajectory is traced with the presence of an obstacle.

Case 1:-

Table 7.3: Case 1-Parameters of DMM

Lambda (λ)	**Safe Distance**
1 - $\left(\frac{\text{last\_obstacle\_distance}^2}{\text{safe\_distance}^2}\right)$	15 m

Table 7.4: DMM Case 1 - Goal and Obstacle Position

Goal	Obstacle
(40,0)	(15, 0)

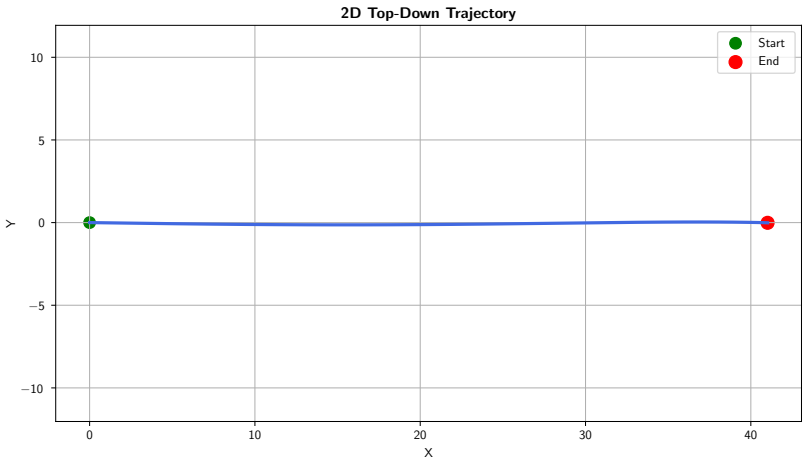


Figure 7.6: Top View Trajectory without Obstacle

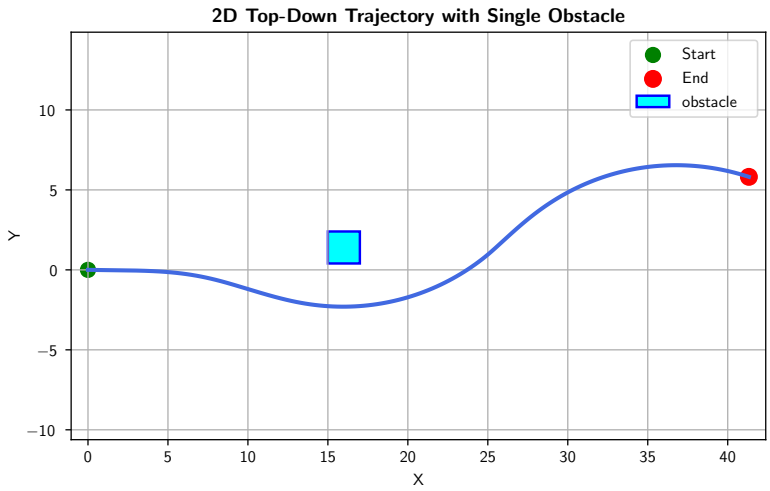


Figure 7.7: Top View Trajectory with Obstacle

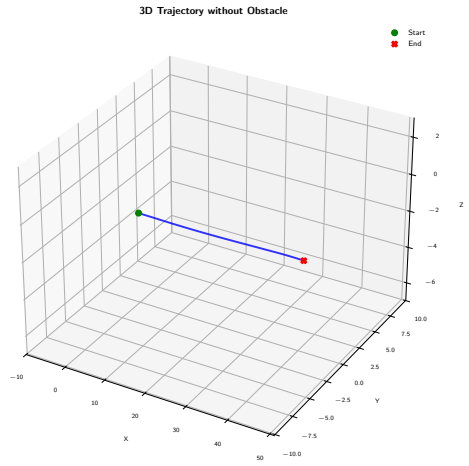


Figure 7.8: 3D View of Trajectory Without Obstacle

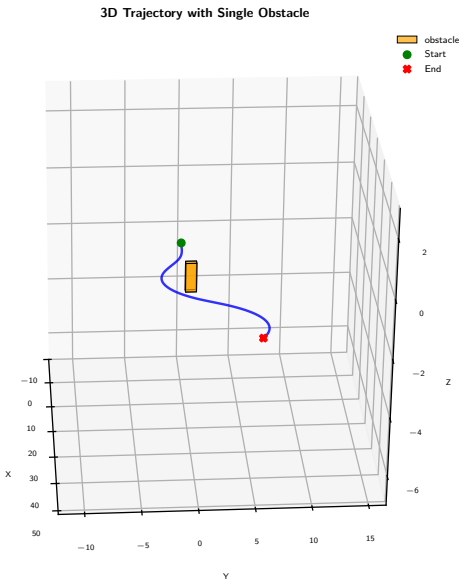


Figure 7.9: 3D view of Trajectory With Obstacle

Case 2:-

Table 7.5: DMM Case 2-Parameters of DMM

Lambda (λ)	**Safe Distance**
1 - $\left(\frac{\text{last\_obstacle\_distance}^2}{\text{safe\_distance}^2}\right)$	7 m

Table 7.6: DMM Case 2 - Goal and Obstacle Position

Goal	Obstacle
(40,0)	(15,0)

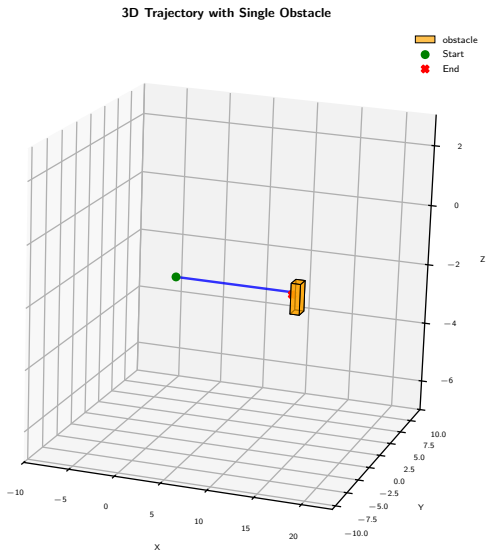


Figure 7.10: 3D view of Trajectory With Obstacle

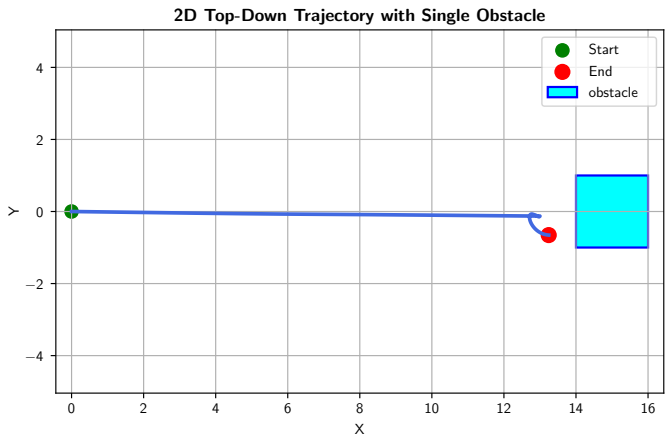


Figure 7.11: 2D view of Trajectory With Obstacle

Case 3:-

Table 7.7: Case 3-Parameters of DMM

Lambda (λ)	**Safe Distance**
1 - $\left(\frac{\text{last\_obstacle\_distance}^2}{\text{safe\_distance}^2}\right)$	15 m

Table 7.8: DMM Case 3 - Goal and Obstacle Position

Goal	Obstacle_1	Obstacle_2	Obstacle_3
(40,0)	(15,-1)	(18,1.5)	(15,3)

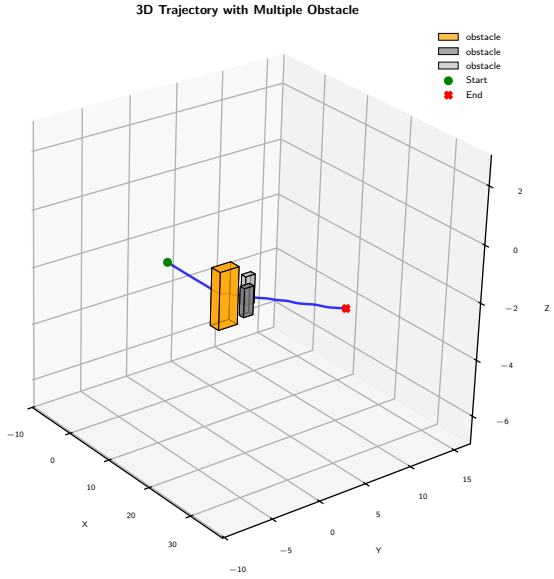


Figure 7.12: 3D view of Trajectory With Obstacles

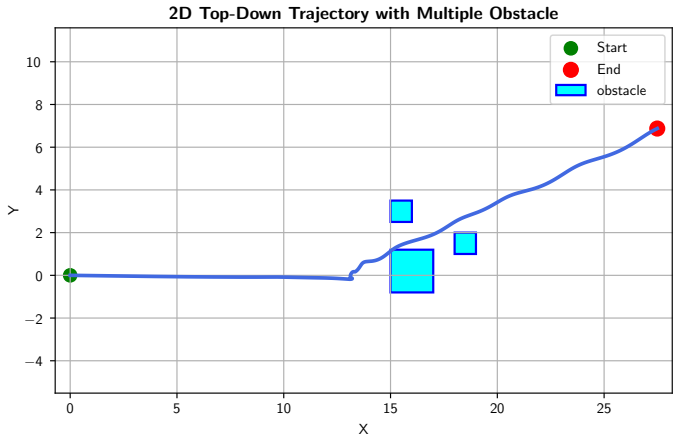


Figure 7.13: 2D view of Trajectory With Obstacles

7.3 Rotational Obstacle Avoidance Method

Problem Description: The demonstration of this method will be done in situations similar to DMM. The initial position, goal position, and obstacle placement will be the same so that both methods can be compared.

Case 1:-

Table 7.9: ROAM Case 1 - Goal and Obstacle Position

Goal	Obstacle
(40,0)	(15, 0)

Table 7.10: ROAM Parameters

Safe Distance	Gamma(γ)	R_e
4m	2	$\pi/2$

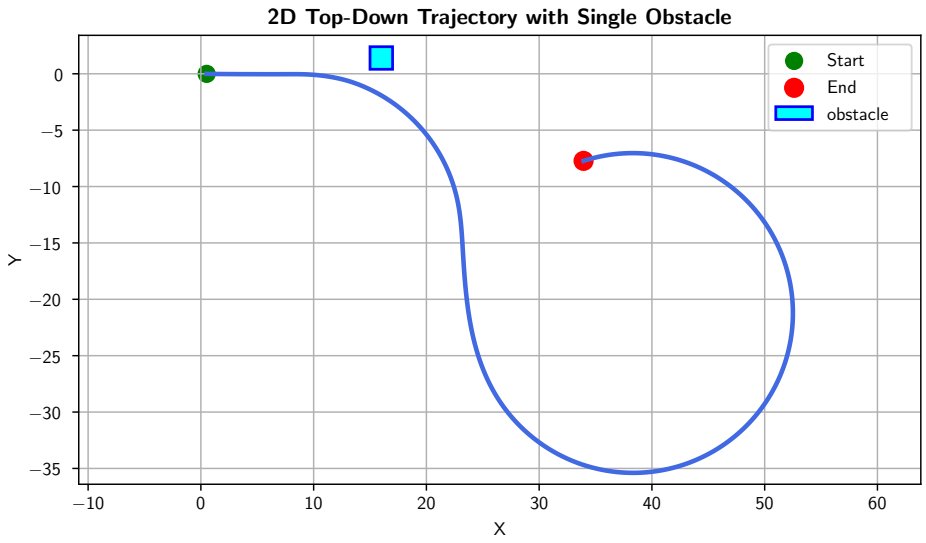


Figure 7.14: 2D View of Trajectory With Single Obstacle

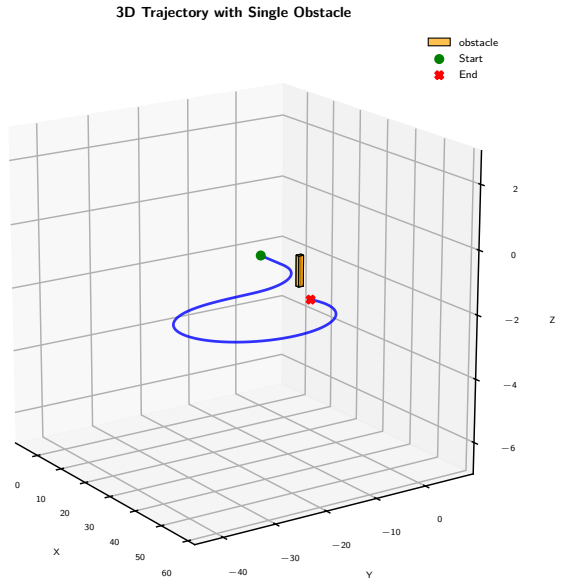


Figure 7.15: 3D View of Trajectory With Single Obstacle

Case 2:-

Table 7.11: Case 2 - Goal and Obstacle Position

Goal	Obstacle
(40,0)	(15, 0)

Table 7.12: ROAM Parameters

Safe Distance	Gamma(γ)	R_e
10m	1.5	π

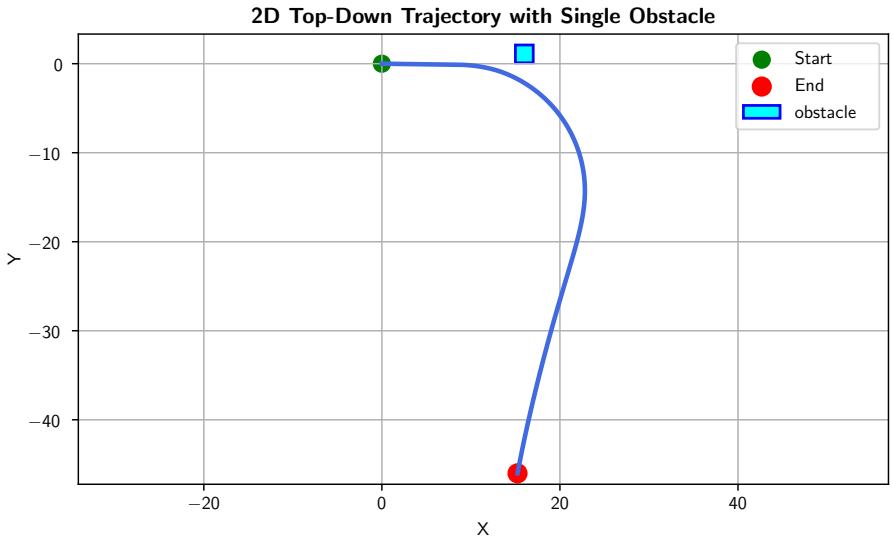


Figure 7.16: 2D View of Trajectory With Single Obstacle

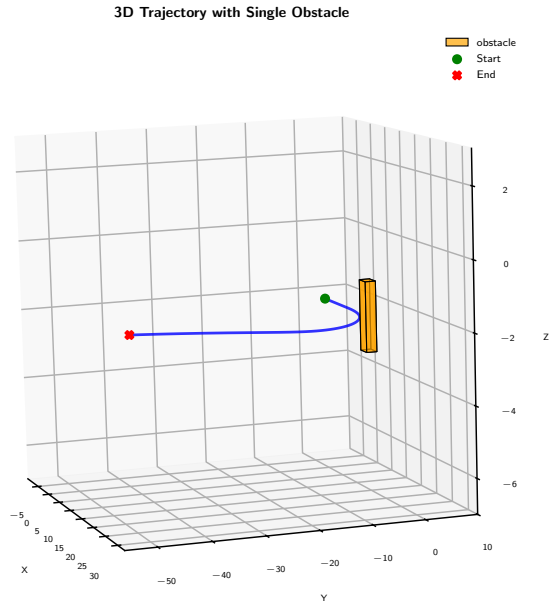


Figure 7.17: 3D View of Trajectory With Single Obstacle

Case 3:-

Table 7.13: ROAM Case 3 - Goal and Obstacle Position

Goal	Obstacle_1	Obstacle_2	Obstacle_3
(40,4)	(10, 0.8)	(13,3)	(9,5)

Table 7.14: Case 3 -ROAM Parameters

Safe Distance	Gamma(γ)	R_e
4m	2	$\pi/2$

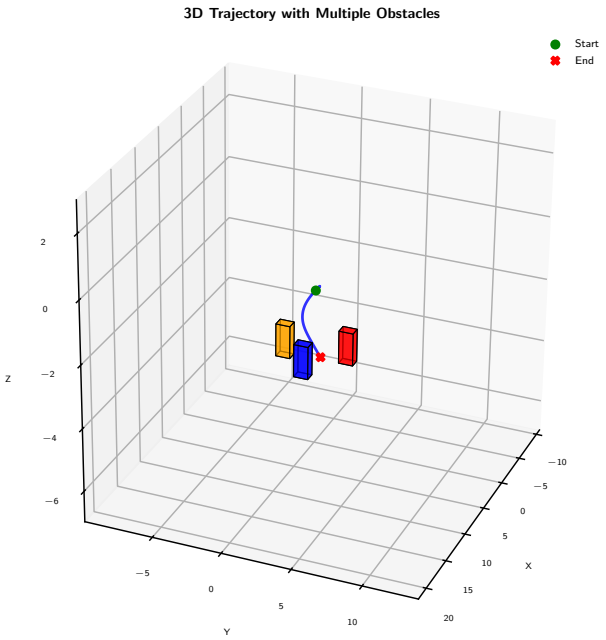


Figure 7.18: 3D View of Trajectory With Obstacles

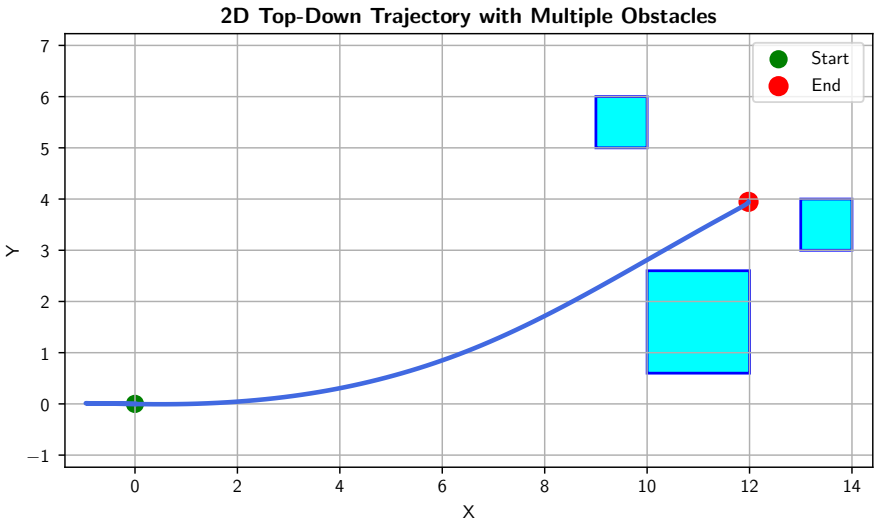


Figure 7.19: 2D View of Trajectory With Obstacles

7.4 Analysis

Comparison Between DMM and ROAM

To evaluate the performance of the obstacle avoidance methods, both **Dynamic Modulation Method (DMM)** and **Rotational Obstacle Avoidance Method (ROAM)** were tested in simulation under various obstacle configurations using **ROS2** and **Gazebo**. The focus was on assessing their effectiveness in terms of *goal convergence*, *response sharpness*, and *ability to navigate around single and multiple obstacles*.

1.Single Obstacle Scenario DMM demonstrated smooth and stable convergence toward the goal with minimal deviation in trajectory. Its modulation-based strategy allowed the agent to gently curve around the obstacle while continuously progressing toward the target.

ROAM, in contrast, produced sharper and faster reactions. The rotational strategy made the agent aggressively avoid the obstacle. While this improved speed and reactivity, it occasionally resulted in longer path lengths due to wider turns.

2.Multi-Obstacle Scenario DMM struggled in environments with closely spaced or multiple obstacles. The modulation matrix often failed to account for combined obstacle influences, causing agents to get trapped.

ROAM excelled in multi-obstacle environments. Its ability to rotate around complex shapes and dynamically adjust its trajectory allowed it to maintain motion continuity and avoid deadlocks. The method successfully navigated agents through narrow passages with better clearance and adaptability.

Comparison Summary

Criterion	DMM	ROAM
Goal Convergence	High	Moderate
Obstacle Avoidance	Effective (single obstacle)	Effective (all cases)
Sharpness/Responsiveness	Smooth but slower	Sharp and fast
Performance (Multi-obstacle)	Poor	Excellent
Trajectory Smoothness	High	Moderate

Table 7.15: Performance comparison between DMM and ROAM

While DMM is advantageous for scenarios where smooth, direct convergence to the goal is critical, ROAM proves to be more versatile and robust in cluttered environments, thanks to its quick reaction and adaptability.

8 Summary and Future plan of work

This thesis presents the design, development, and simulation of a decentralized multi-agent system that performs self task allocation and real-time obstacle avoidance in dynamic environments. The framework is built on the Robot Operating System 2 (ROS2) and simulated using the Gazebo environment, providing a realistic and modular setup for robot modeling, sensing, and control.

A key contribution is the implementation of a capability-based self task allocation strategy, where each agent autonomously evaluates its own suitability for available tasks and selects the most appropriate one without relying on centralized coordination. To ensure safe and efficient navigation in obstacle-rich environments, the system integrates two obstacle avoidance strategies: the Dynamic Modulation Method (DMM), which modulates velocity fields to smoothly deflect trajectories around obstacles, and the Rotational Obstacle Avoidance Method (ROAM), which enables agents to generate rotational motion around obstacles to maintain safe clearance and path continuity.

The system is demonstrated in simulation using Surface Vehicles and Underwater Vehicles, showcasing their ability to allocate tasks, avoid collisions, and navigate cooperatively. The use of ROS2 and Gazebo enables high-fidelity simulation, real-time interaction, and future adaptability for real-world deployment. This work contributes to scalable and resilient solutions in the domains of underwater exploration, search-and-rescue, and multi-agent robotic operations.

Future Plan of Work

- Full body obstacle avoidance for tail avoidance of AUVs
- Testing ROAM Algorithm on Dynamic Obstacles

References

[1] M. S. Mohd Aras, H. Kasdirin, M. Jamaluddin, and M. Basar, "Design and development of an autonomous underwater vehicle (auv-fkeutem)," *Proceedings of MUCEET2009 Malaysian Technical Universities Conference on Engineering and Technology, MUCEET2009, MS Garden, Kuantan, Pahang, Malaysia*, 01 2009.

[2] *Models for Ships, Offshore Structures and Underwater Vehicles*. John Wiley Sons, Ltd, 2011, ch. 7, pp. 133–186. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119994138.ch7

[3] R. Hu, D. Lu, C. Xiong, C. Lyu, H. Zhou, Y. Jin, T. Wei, C. Yu, Z. Zeng, and L. Lian, "Modeling, characterization and control of a piston-driven buoyancy system for a hybrid aerial underwater vehicle," *Applied Ocean Research*, vol. 120, p. 102925, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014111872100393X

[4] A. Sahoo, S. K. Dwivedy, and P. Robi, "Advancements in the field of autonomous underwater vehicle," *Ocean Engineering*, vol. 181, pp. 145–160, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801819301623

[5] A. Natu, V. Garg, P. Gaur, U. Biswas, D. Bansal, N. Biswas, M. Ranjan, M. Goyal, R. Gupta, S. Parashar, R. Bansal, C. Kumar, R. Kumar, A. Kumar, A. Tyagi, A. Verdhan, D. Auv, Dhruv, P. Raj, and S. Sain, "Design and development of an autonomous underwater vehicle varuna 2.0," 08 2020.

[6] L. W. L. Alexander, M. M. A. Roslan, K. Isa, H. A. Kadir, and R. Ambar, "Design and development of an autonomous underwater vehicle (auv) with target acquisition mission module," in *2018 IEEE 8th International Conference on Underwater System Technology: Theory and Applications (USYS)*, 2018, pp. 1–6.

[7] C. Wang, D. Mei, Y. Wang, X. Yu, W. Sun, D. Wang, and J. Chen, "Task allocation for multi-auv system: A review," *Ocean Engineering*, vol. 266, p. 112911, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801822021941

[8] E. Bischoff, F. Meyer, J. Inga, and S. Hohmann, "Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, pp. 3949–3956.

[9] Y. Ma, Y. Liu, L. Zhao, and M. Zhao, "A review on cooperative control problems of multi-agent systems," in *2022 41st Chinese Control Conference (CCC)*, 2022, pp. 4831–4836.

[10] Z. Fang, D. Jiang, J. Huang, C. Cheng, Q. Sha, B. He, and G. Li, "Autonomous underwater vehicle formation control and obstacle avoidance using multi-agent generative

adversarial imitation learning," *Ocean Engineering*, vol. 262, p. 112182, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801822014986

[11] M. Liu and Y. Feng, "Group consensus of mixed-order multi-agent systems with fixed and directed interactive topology," *IEEE Access*, vol. 7, pp. 179 712–179 719, 2019.

[12] Y. Bai, Y. Wang, M. Svinin, E. Magid, and R. Sun, "Adaptive multi-agent coverage control with obstacle avoidance," *IEEE Control Systems Letters*, vol. 6, pp. 944–949, 2022.

[13] L. Wen, J. Yan, X. Yang, Y. Liu, and Y. Gu, "Collision-free trajectory planning for autonomous surface vehicle," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2020, pp. 1098–1105.

[14] L. Cao, G.-P. Liu, and D.-W. Zhang, "A leader-follower consensus control of networked multi-agent systems under communication delays and switching topologies," in *2022 41st Chinese Control Conference (CCC)*, 2022, pp. 4378–4383.

[15] J. Xu, F. Huang, D. Wu, Y. Cui, Z. Yan, and K. Zhang, "Deep reinforcement learning based multi-auvs cooperative decision-making for attack–defense confrontation missions," *Ocean Engineering*, vol. 239, p. 109794, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801821011562

[16] B. Xu, Z. Wang, and H. Shen, "Distributed predictive formation control for autonomous underwater vehicles under dynamic switching topology," *Ocean Engineering*, vol. 262, p. 112240, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801822015505

[17] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1462–1469, Apr. 2019.

[18] L. Huber, J.-J. Slotine, and A. Billard, "Avoidance of concave obstacles through rotation of nonlinear dynamics," *IEEE Trans. Robot.*, vol. 40, pp. 1983–2002, 2024.

[19] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Auton. Robots*, vol. 32, no. 4, pp. 433–454, May 2012.

[20] L. Huber, "Exact obstacle avoidance for robots in complex and dynamic environments using local modulation," PhD thesis, EPFL, Lausanne, Switzerland, April 2024.

[21] accurrent. (April) Mbari tethys lrauv. Open Robotics. [Online]. Available: https://fuel.gazebosim.org/1.0/accurrent/models/MBARITethysLRAUV

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

Indian Institute of Technology Jodhpur

CSL 7360 COMPUTER VISION

Project Report

UNDERWATER PIPELINE DETECTION

Submitted By

Arashdeep Singh (M23IRM003)

Chaitanya Patil (M23IRM004)

Harshit Dhanorkar (M23IRM005)

Kranti Parkash (M23IRM006)

Contents

Introduction

We have chosen a research paper (A Classical Computer Vision Pipeline for Underwater Detection of Long, Flexible, and Highly Deformable Curvilinear Objects) based on which we have developed this project. Our purpose of this project is to build a robust and dynamic pipeline for underwater images of pipelines and power lines. As underwater images are of not so good quality and photos vary in nature largely. If we have to build an algorithm that takes underwater images as input and gives judgment on pipelines or power lines. Then, we want the images to be as good so that the algorithm can easily detect objects with higher accuracy. That's why, before sending images to the algorithm, we need to make changes to images so that the algorithm can detect certain types of features to give output. As we are dealing with long objects, so we want these to be visible as edges. We are not choosing a deep learning approach here. Our analysis is based on Classical Computer Vision Techniques.

What is the purpose of this detection.

Focusing on detecting long objects in underwater imagery may seem specific, but it has practical applications.

1. **Navigation and Path Planning**:
    - **Autonomous Underwater Vehicles (AUVs)**: AUVs rely on edge detection to navigate and avoid obstacles. By identifying edges (such as rocks, reefs, or underwater structures), AUVs can plan collision-free paths.
    - **Search and Rescue**: Edge detection assists in locating submerged objects (e.g., lost equipment, wreckage, or missing persons) during search and rescue operations.
2. **Scientific Research:**
    - Biological Studies: Identifying edges of marine organisms (fish, corals, etc.) aids in biodiversity assessments and ecological research.
    - Geological Surveys: Detecting underwater geological features (fault lines, ridges, etc.) contributes to geological studies.

3.Search and Recovery Operations:

- Locating lost objects or wreckage underwater is challenging.
- Detecting long submerged objects aids search teams, divers, and autonomous vehicles.

- After a shipwreck, identifying a sunken vessel's elongated parts helps in recovery efforts.

4.Underwater Archaeology:

- Ancient structures and artifacts lie hidden beneath the waves.
- Detecting elongated features assists archaeologists in mapping and preserving submerged heritage.
- : Locating a ship's mast or anchor provides insights into historical voyages.

5. Security and Defense:

a. Monitoring underwater threats is vital for national security.

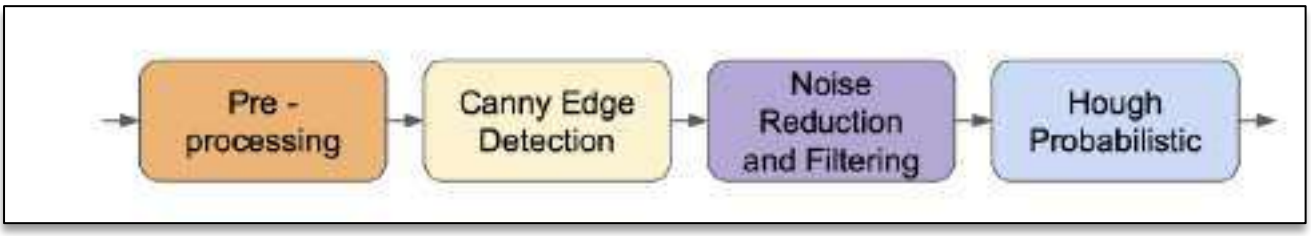
b. Detecting long objects (e.g., submarines, torpedoes) enhances surveillance.

c. **Example**: Identifying a hidden submarine's periscope prevents surprise attacks.

Research Paper

Vision-based algorithms have been widely used for detection of underwater objects that are long, flexible, and highly deformable - like cables, pipelines, ropes, wires, and long strands of fishing net. Most algorithms use a deep learning or a combination of deep learning and classical computer vision approach to detect these specific objects. This paper presents a pure classical computer vision approach that aims to solve this detection problem by combining several state-of-the art computer vision techniques. The implemented algorithm is tested using real world data collected using a AUV.

Proposed Pipeline



Pre-processing

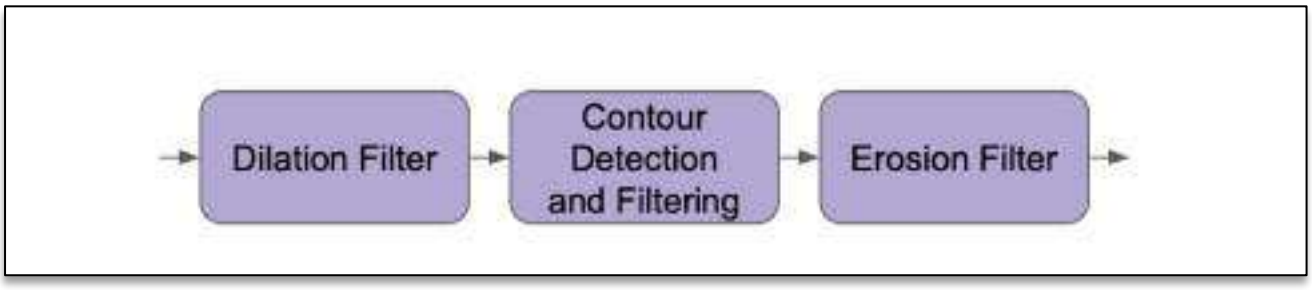
The first step of the pipeline is the pre-processing step. The input image is converted from a RGB image space to a LAB image space in order to enhance contrasts. Then, a gaussian blur filter with a fixed kernel is applied to the image in order to silence noise and particles in the image. This is useful for the future steps. The first step of the pipeline is the pre-processing step. The input image is converted from a RGB image space to a LAB image space in order to enhance contrasts. Then, a gaussian blur filter with a fixed kernel is applied to the image in order to silence noise and particles in the image. This is useful for the future steps.

Canny Edge Detector

The next step in the pipeline is the canny edge detection step. The canny edge detection algorithm is a multi-scaled multi-staged edge detection algorithm proposed by John F. Canny in 1986 that uses a threshold approach to determine edge points [11] [12]. A fixed lower threshold and upper threshold is set with an option to use an adaptive threshold. The canny edge detection step returns a black and white image with black pixels representing non-edges and white pixels representing edges.

Noise Reduction and Filtering

The next step of the pipeline is to reduce and filter out noise generated from the canny edge detection step. The canny edge detection filter is extremely sensitive to noise as canny edge detection detects all edges in the image and noise particles, especially in the underwater domain, contain sharp edges. A sub-pipeline is proposed and implemented in order to conduct noise reduction and filtering. The sub-pipeline is shown in Figure 6.



This step requires the use of morphological filters. Morphological filters are binary image operations that can change the shape of the underlying binary objects [13]. A combination of dilation and erosion morphological filters is used in the proposed algorithm. A dilation morphological filter expands the binary image while an erosion morphological filter shrinks the binary image.

A dilation morphological filter is first used with a kernel size of 3 × 3 with 1 iteration. The dilation filter expands the binary image such that stronger and larger edge detections become very large, and at the same time, the size of the smaller noise edge detections also become comparatively large. This step is also responsible in merging smaller segments of edges. A contour detection algorithm is then applied to the dilated binary image, which outlines the line edge segments. The contours are then filtered by hierarchy such that smaller contours inside larger contours are filtered out and all contours smaller than a minimum contour area threshold are filtered out. The purpose of a contour detection and filtering step is to outline the contours and remove contours with smaller areas as these contours (edge segments) are usually additional noise particles. A erosion filter with a kernel size of 2 × 2 with 1 iteration is then applied to the image with contours such that the detected contour area sizes are normalized.

Hough Probabilistic

The final step of the pipeline is to apply a hough probabilistic filter. A hough transform is a technique in computer vision that is used to detect shapes and lines by mapping points in images into a parameter space where patterns of points are identified as peaks [14]. Although standard hough transform is an efficient technique for detecting image features, the complexity and memory requirements will highly

increase with the dimension and number of edge points. Probabilistic hough transform aims to solve this issue by using random sampling of the edge points in the input image instead of using all edge points [15]. This step takes the output black and white image from the Noise Reduction and Filtering step and identifies probabilistic hough lines with with a minimum line length of 50 and maximum line gap of 50. The parameters can also be updated in the code. An additional step to merge the detected hough lines is also implemented. Here, the distance between each hough lines is computed. If the minimum distance between two lines is smaller than a minimum distance threshold, the lines are joined with an additional line (merged). The final output from the algorithm is show in Figure 1

Here is the result of all stages of processing.

Fig. Actual Image 1

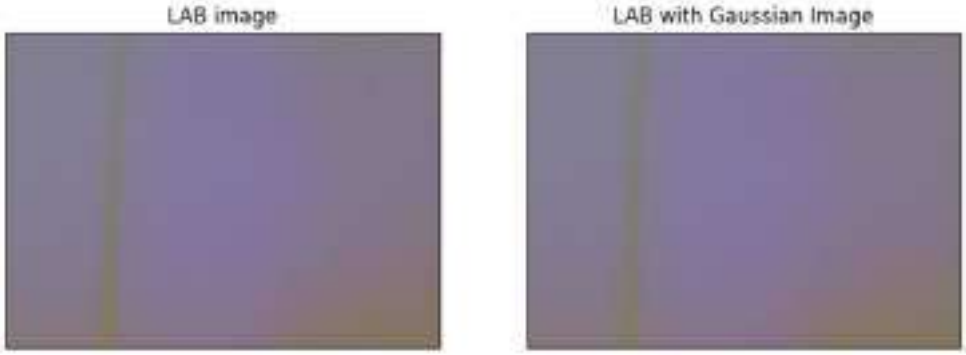


Fig. LAB image and after applying gaussian

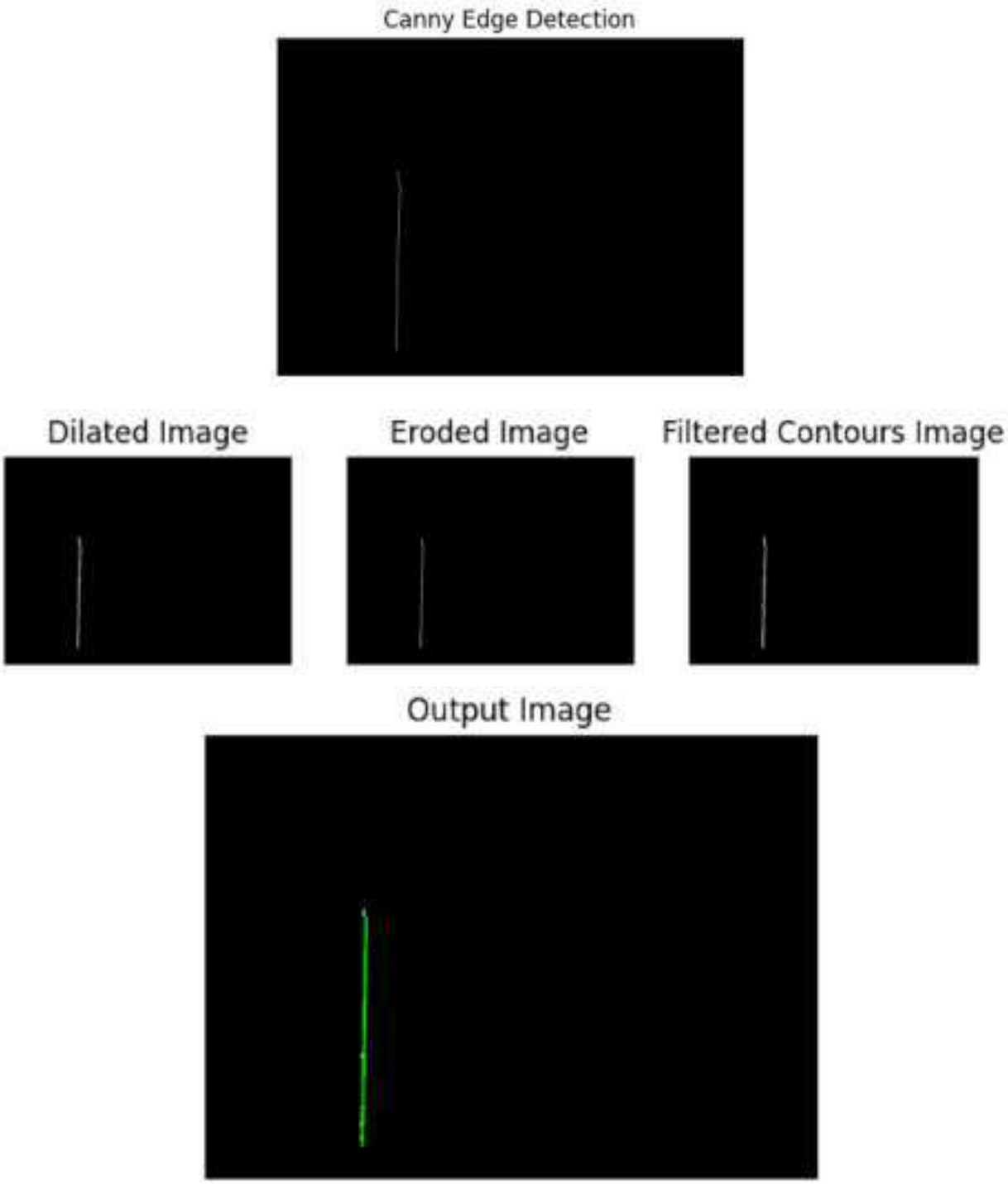


Fig. Various process on the image 1 and the result

Fig. Actual Image 2

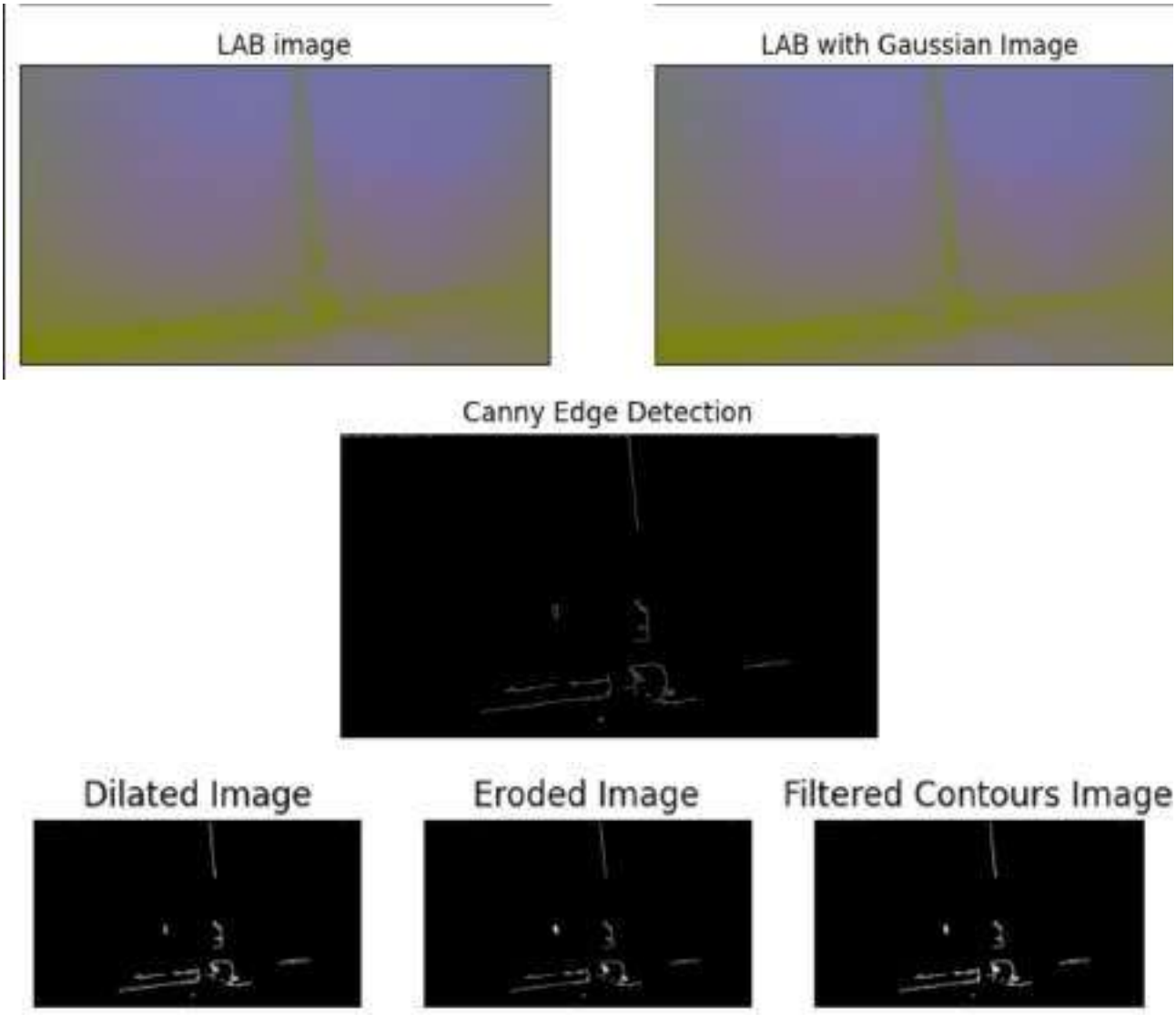


Fig. Various Process on the image 2

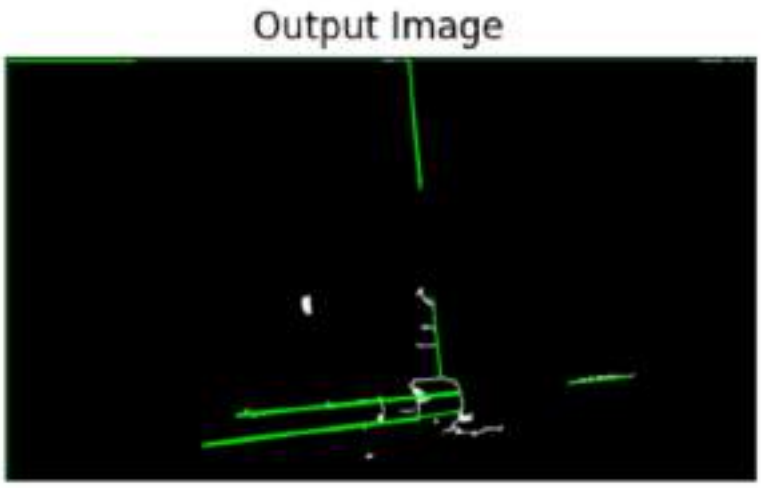
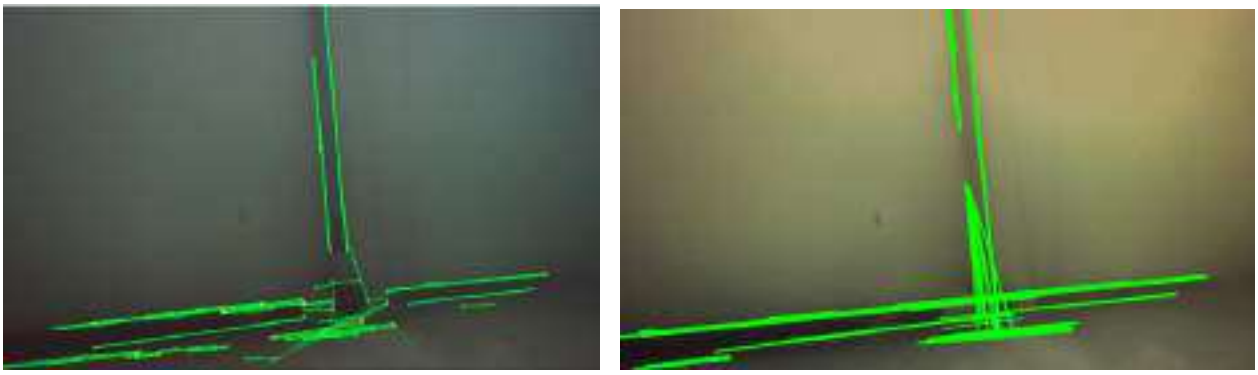


Fig. Result for the Image 2

Comparison of their and our result

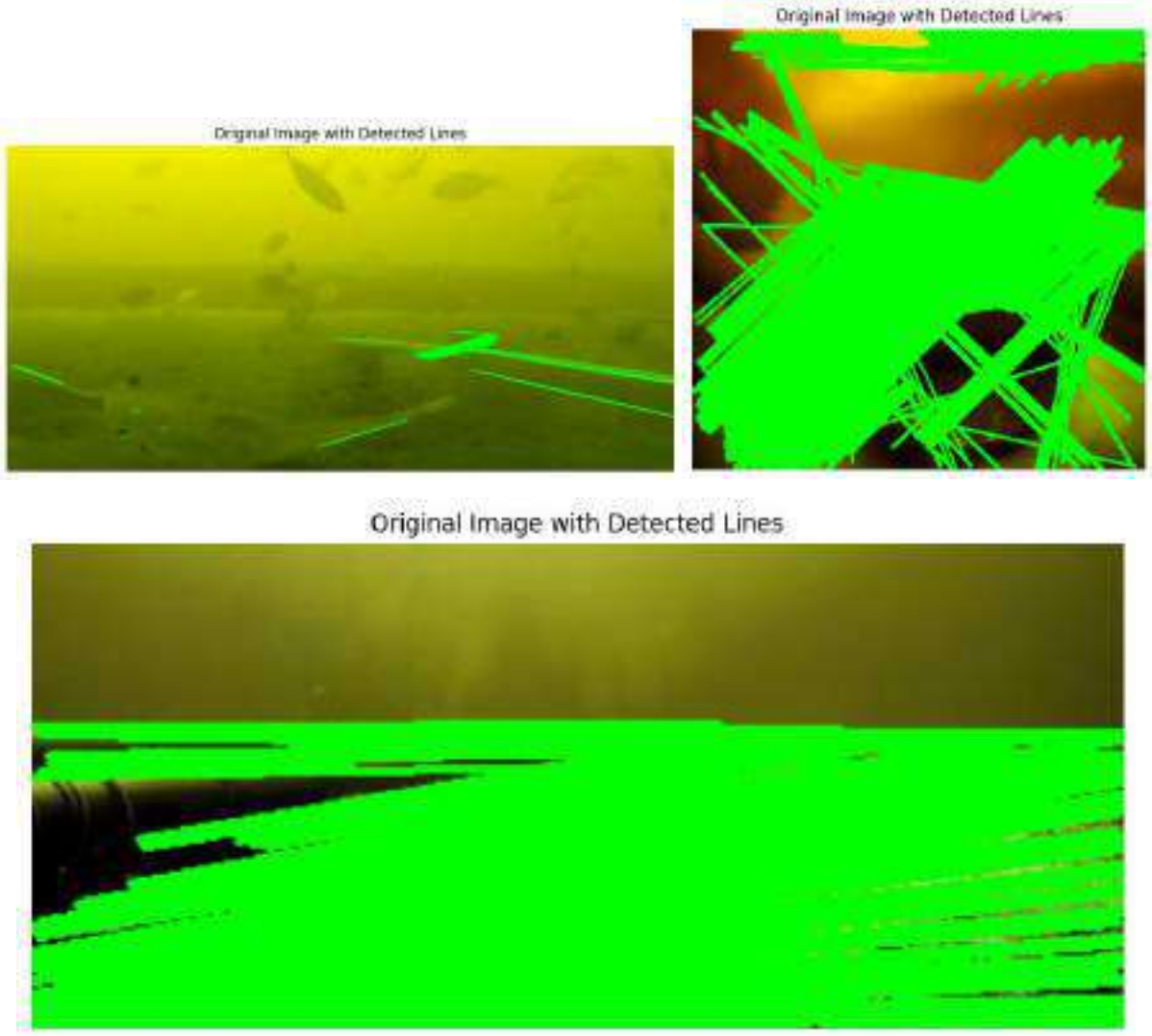


(a) (b)

Fig. Their work (a) and our work (b).

Comment on Pipeline's performance.

First see the output on other images: -



- Not applicable across all images.
- Demonstrates effectiveness primarily on selective images.

Here are the results for 16 different kinds of images.

Good	Bad
4	12

Developing New Pipeline

- Adding Histogram Equalization to already developed pipeline.

**little about hist

Using LAB+Gaussian+Canny Edge Detector + Morphological+ Probabilistic Hough+ Histogram Equalization

Result with these

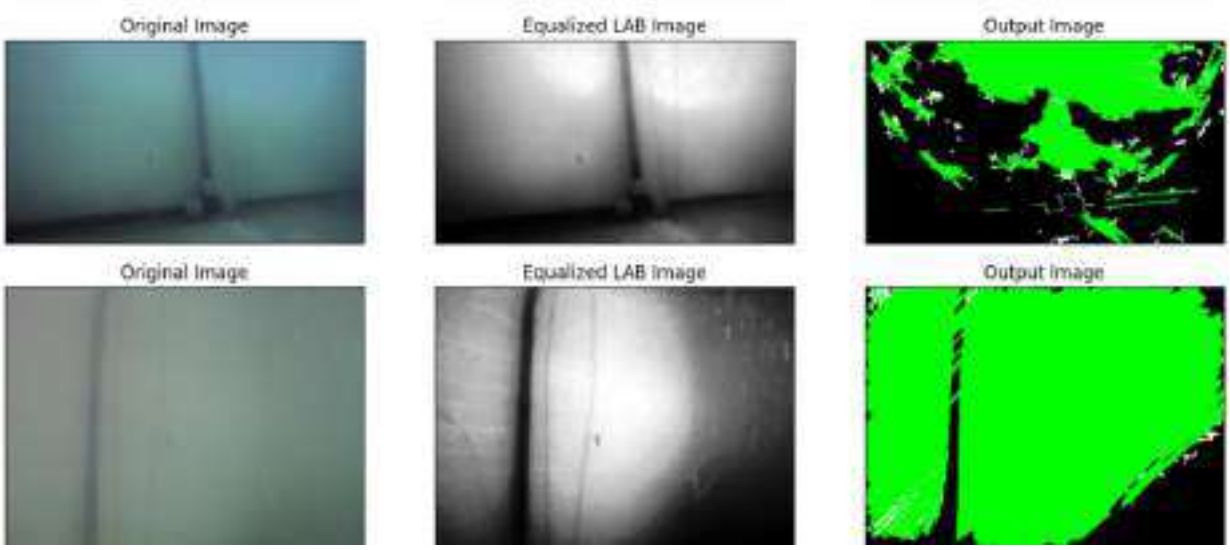


Fig. Image process and result

Suggestions to improve the Pipeline.

- **Suggestion 1. ADAPTIVE THRESHOLD FOR EDGE DETECTOR**

Adaptive Thresholding: Instead of using fixed thresholds for edge detection, you can use adaptive thresholding techniques that automatically adjust the thresholds based on local image properties. Adaptive thresholding methods like Adaptive Gaussian Thresholding (`cv2.adaptiveThreshold`) can be helpful in handling variations in lighting conditions.

Result with Adaptive Threshold

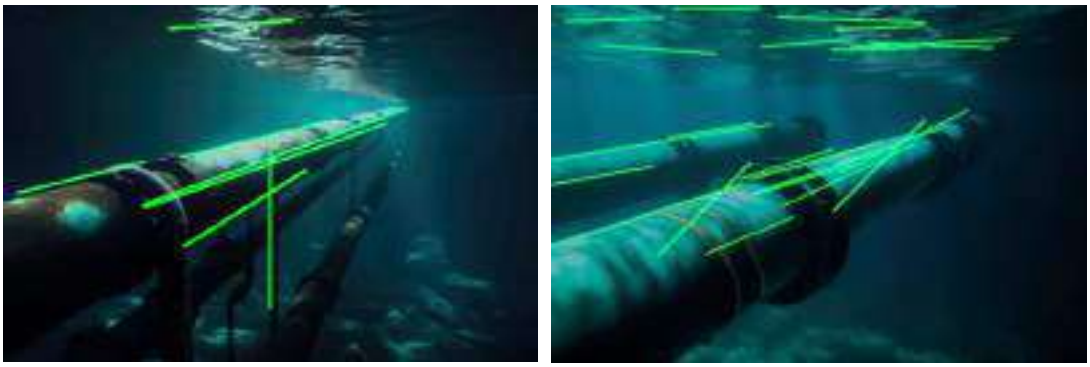


Fig. Results using adaptive Threshold

It is working very well. But can we make it a little better?

- **Suggestion 2:**

Using Gaussian, Adaptive, Morphological, Hough

Result: Not Worked Just Same Image as Original

- **Suggestion 3:**

Using Gaussian, Canny, Gaussian, Hough

Results:

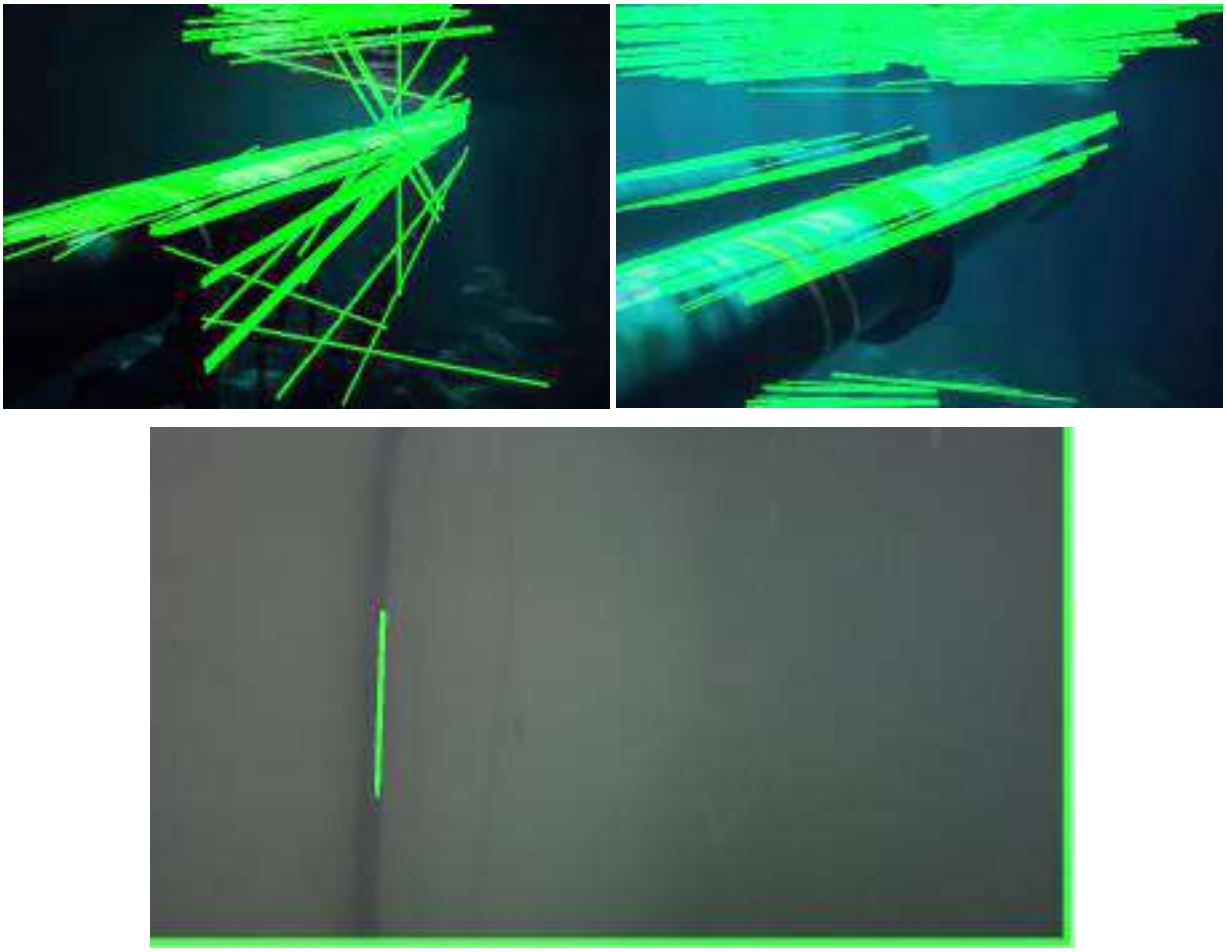


Fig. Results on different images

Too strong false detection.

Suggestion 4:

Using White Balancing, Contrast Limited, Adaptive Histogram Equalization, Gaussian, Canny, Hough Transformation

Result.

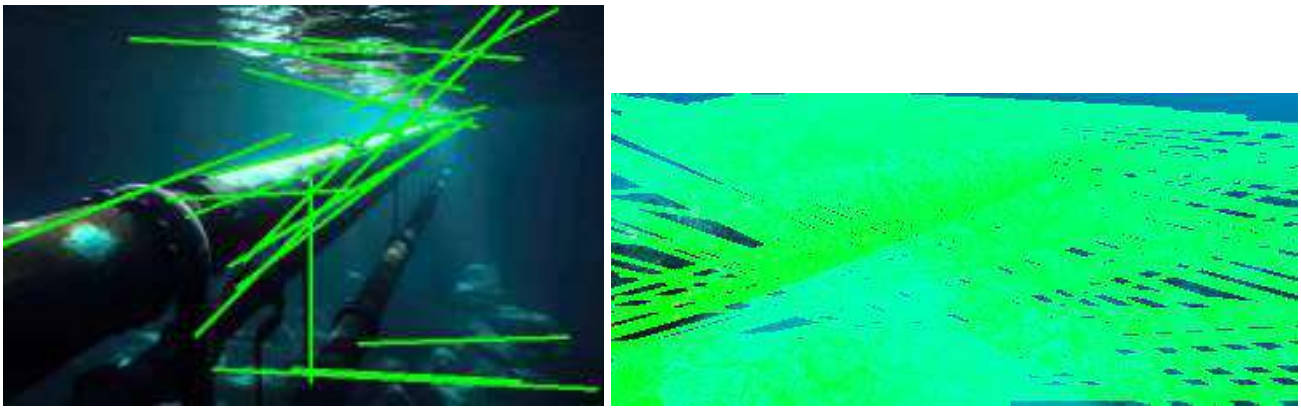


Fig. Results on different images

Good for some type of images only.

➢ Suggestion 5:

Addition of Region of Interest

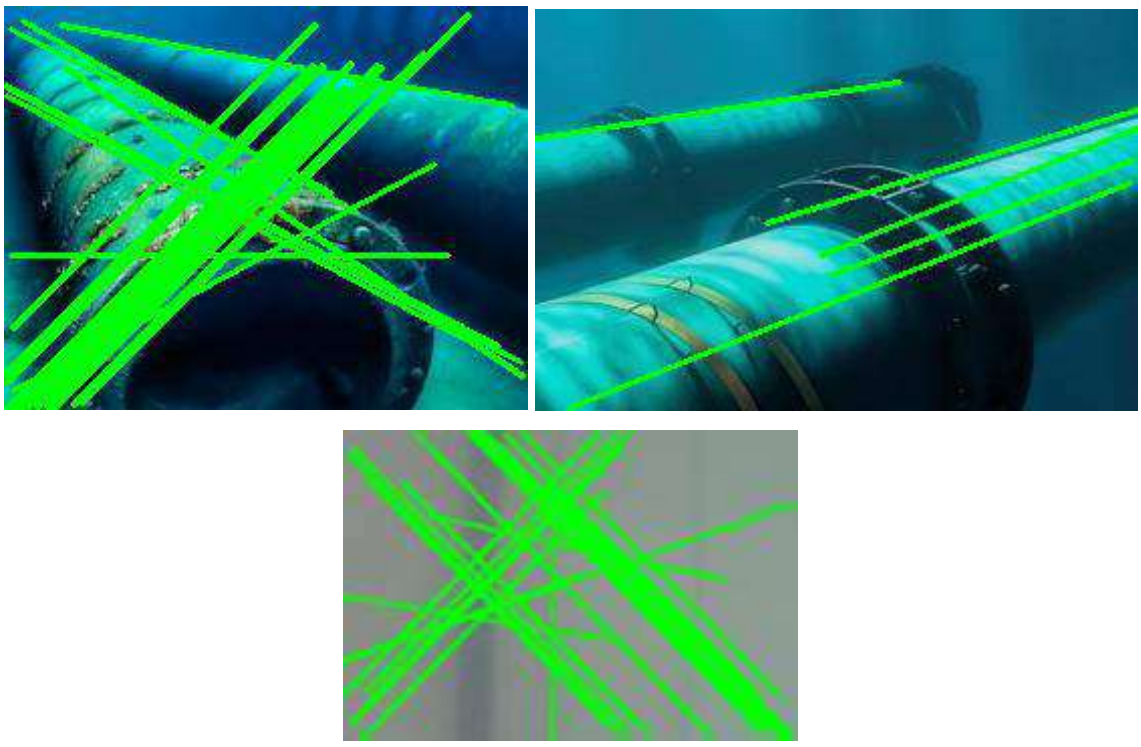


Fig. Results on different images

Too many false detections.

- **Suggestion 6**:

Using White Balancing, Contrast Limited, Adaptive Histogram Equalization, Gaussain, Adaptive, Adaptive ROI, Canny, Hough.

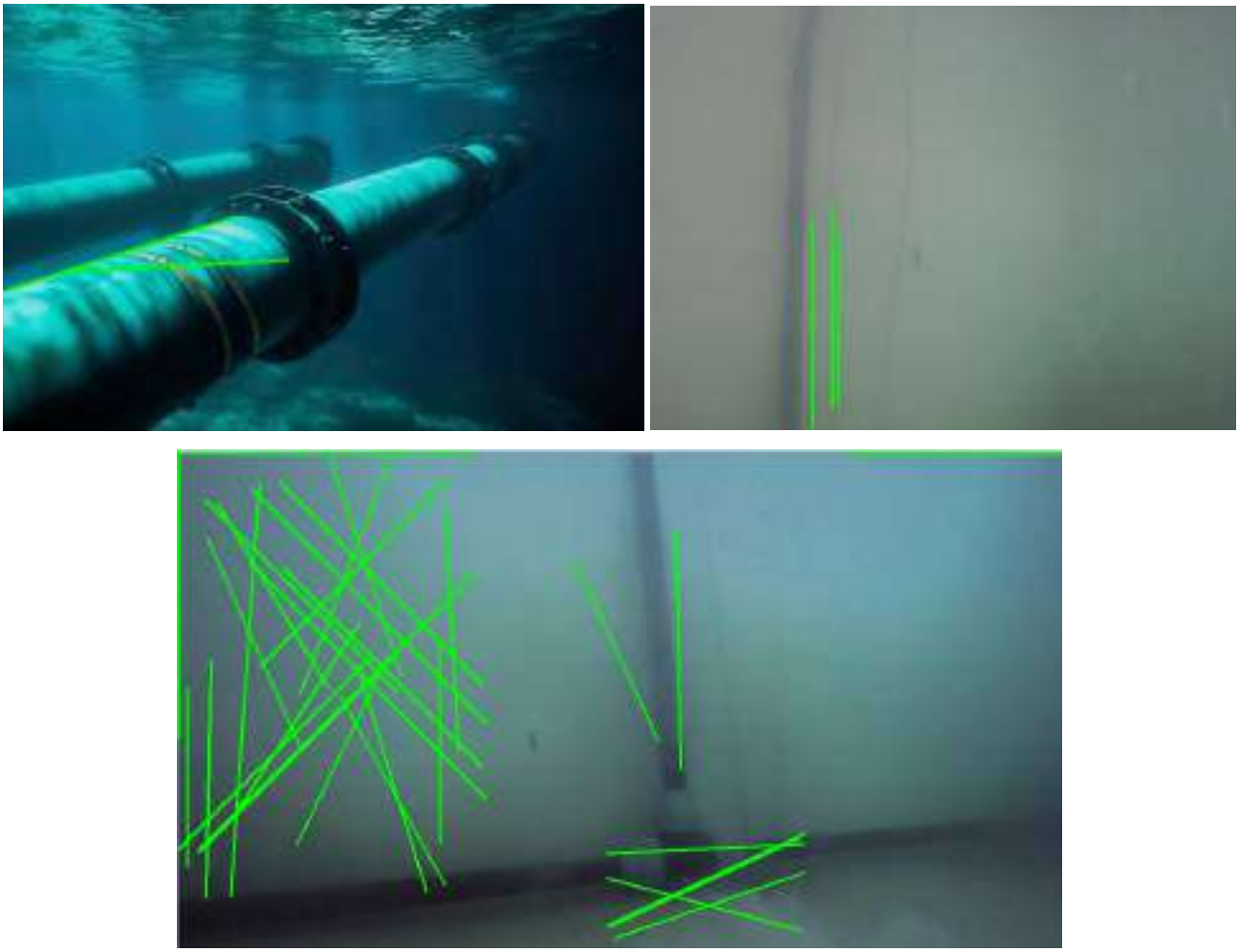
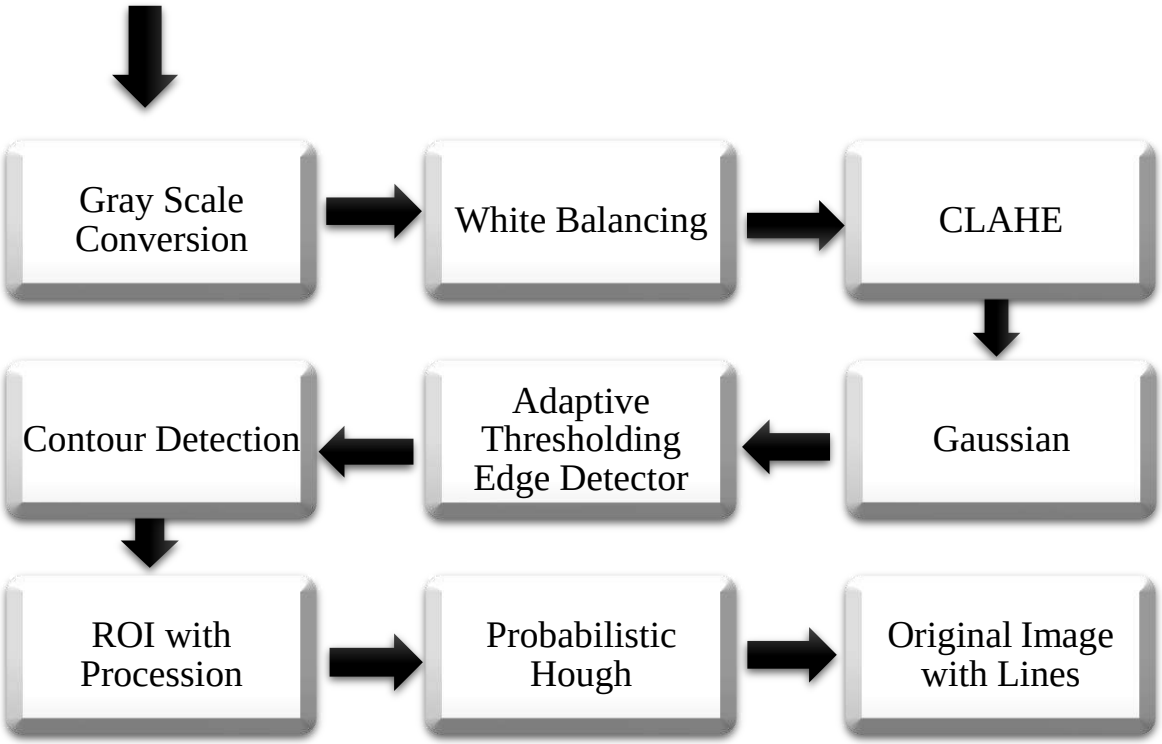


Fig. Results on different Images

Some false detections, otherwise fine.



FINAL PIPELINE

Here is a short explanation and reasoning for usage of all used techniques: -

1. **White Balancing**:

White balancing is a process that adjusts the colors in an image to make white objects appear truly white, regardless of the lighting conditions. It aims to remove color casts caused by different light sources. **Use in Underwater Pipeline Detection**: Underwater environments often suffer from color distortion due to varying water conditions and light penetration. White balancing helps correct these color shifts, making it easier to identify pipelines and other structures.

2. **Contrast-Limited Adaptive Histogram Equalization (CLAHE)**:

CLAHE is an enhancement technique that improves local contrast in an image. It divides the image into small tiles, computes histograms for each tile, and then redistributes pixel intensities based on these histograms while limiting amplification. **Use in Underwater Pipeline Detection**: CLAHE enhances details in underwater images, making pipelines more distinguishable. By adapting to local variations in lighting and contrast, it helps reveal subtle features that might otherwise be obscured.

3. **Gaussian Filtering**:

Gaussian filtering is a type of image smoothing technique that uses a Gaussian function as a kernel to blur an image. It reduces noise and sharpens edges.

Use in Underwater Pipeline Detection: Gaussian filtering helps suppress noise caused by water turbidity and sensor limitations. By reducing noise, it enhances the visibility of pipelines and minimizes false detections.

4. **Adaptive Threshold for Edge Detection**:

The edge detector identifies edges in an image by detecting significant intensity changes. Adaptive thresholding adjusts the threshold dynamically based on local image characteristics.

Use in Underwater Pipeline Detection: Underwater scenes often have varying lighting conditions and noise levels. Adaptive thresholding ensures that edges are detected consistently across different regions, improving the accuracy of pipeline edge detection.

5. **Probabilistic Hough Transform**:

The Hough Transform detects lines or curves in an image. The probabilistic variant focuses on identifying line segments by sampling only a subset of edge points.

Use in Underwater Pipeline Detection: Probabilistic Hough Transform can robustly detect pipeline segments even when they are partially occluded or broken. It’s useful for reconstructing fragmented pipelines from noisy data.

6. **Adaptive Region of Interest (ROI)**:

Adaptive ROI defines specific areas of interest within an image based on relevant features or dynamic criteria.

Use in Underwater Pipeline Detection: By adaptively selecting ROIs, we can focus computational resources on critical regions where pipelines are likely to be present. This improves efficiency and reduces false positives.

Explanation:

Explanation of all the techniques used in report under the perspective of underwater images: -

1. LAB

The **LAB color space** is a three-dimensional color model used in image processing and computer vision. Let me break it down for you: [16]

1. **L (Lightness)**: The L channel represents the lightness or brightness of a color. It ranges from 0 (black) to 100 (white). Essentially, it captures how much light is reflected by an object.
2. **A (Green-Red)**: The A channel represents the color along the green-red axis. Negative A values correspond to greenish colors, while positive values correspond to reddish colors. Zero represents neutral gray.
3. **B (Blue-Yellow)**: The B channel represents the color along the blue-yellow axis. Negative B values correspond to bluish colors, while positive values correspond to yellowish colors. Again, zero represents neutral gray.

How for underwater pipeline detection LAB image will be useful?

Color Invariance: The LAB color space is more robust to changes in lighting conditions and shadows compared to RGB. In underwater environments, lighting conditions can vary significantly due to water turbidity, depth, and time of day. By working in LAB, you can achieve better color invariance for pipeline detection algorithms.

Contrast Enhancement: LAB separates the luminance (L) channel from the chromatic channels (A and B). The L channel represents the lightness, while the A and B channels represent color information. By enhancing the contrast in the A and B channels, you can highlight color variations associated with pipelines against the background.

Pipeline Color Discrimination: LAB allows you to distinguish between different colors more effectively. For instance, if pipelines have a specific color (e.g., yellow or orange), you can create a

mask based on the A and B channels to isolate those colors. This can aid in segmenting the pipelines from the surrounding environment.

2. Gaussian Blur Filter

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss).[17]

Gaussian blur filters play a crucial role in enhancing underwater image quality and aiding in pipeline detection. Let me explain how:

1. Noise Reduction: Underwater images often suffer from noise due to scattering, absorption, and other environmental factors. Gaussian blurring helps reduce noise by smoothing out pixel values. By averaging neighboring pixels, it suppresses random fluctuations and enhances the overall image quality.[18]
2. Edge Preservation: While reducing noise, Gaussian blurring maintains the edges and structures in the image. Unlike some other filters that might blur edges, Gaussian blur strikes a balance between noise reduction and edge preservation. This is essential for detecting meaningful features in the image. [19]
3. Preprocessing for Detection Algorithms: When detecting pipelines in underwater scenes, you want to focus on the actual pipeline structures rather than being influenced by noise or irrelevant details. Gaussian blurring pre-processes the image, making it easier for subsequent detection algorithms (such as edge detection or segmentation) to work effectively. [20]
4. Enhancing Contrast: By smoothing out variations in pixel intensities, Gaussian blur can enhance contrast. This is particularly useful for distinguishing pipelines from the background. The improved contrast helps algorithms identify pipeline edges more accurately.
5. Reducing Artifacts: Underwater images may contain artifacts caused by light scattering or sensor limitations. Gaussian blurring helps mitigate these artifacts, resulting in cleaner images for subsequent analysis.

3. Canny Edge Detector and Adaptive Edge Detector

Canny Edge Detector [21]

The Canny edge detector is a popular edge detection operator that uses a multi-stage algorithm to identify edges in images. Here are the key steps involved in the Canny edge detection process:

Noise Reduction:

Since edge detection relies on derivatives, it's sensitive to image noise. To mitigate this, we apply Gaussian blur to smooth the image.

The Gaussian kernel size determines the blurring effect. Smaller kernels result in less visible blur.

Gradient Calculation:

Compute the gradient of the image using edge detection operators.

Highlight intensity changes in both horizontal (x) and vertical (y) directions.

Sobel filters are commonly used for gradient calculation.

Non-Maximum Suppression:

Thin out potential edges by keeping only local maxima in the gradient magnitude.

Suppress non-maximum pixels to obtain thin edge lines.

Double Thresholding:

Apply two thresholds: a high threshold and a low threshold.

Pixels with gradient magnitude above the high threshold are considered strong edges.

Pixels between the high and low thresholds are considered weak edges.

Adaptive Edge Detection

Adaptive Canny edge detection aims to automatically adjust the threshold values based on the image content.

Otsu's Thresholding:

Compute Otsu's threshold for the grayscale image.

Use this threshold as the higher threshold for the Canny algorithm.

Preprocessing:

Before applying edge detection, smooth the image (e.g., Gaussian smoothing).

Convert the image to grayscale.

Adaptive edge detection adjusts thresholds dynamically, making it more robust across different images.

4. Dilation and Erosion [22]

Dilation and Erosion are basic morphological processing operations used in image processing. They produce contrasting results when applied to either gray-scale or binary images.

Dilation:

- It is the reverse process with regions growing out from their boundaries.
- It increases the size of the objects.
- It fills the holes and broken areas.
- It connects the areas that are separated by space smaller than the structuring element.
- It increases the brightness of the objects.
- It is used prior in the Closing operation and later in the Opening operation.
- It is XOR of A and B.

Purpose in Underwater Pipeline Detection:

- **Connecting Broken Parts**: Underwater pipelines may have gaps or missing segments due to occlusions or partial visibility. Dilation can bridge these gaps by expanding the pipeline regions, ensuring better continuity.
- **Enhancing Object Features**: Dilation increases the size of detected features. For pipelines, this means enhancing the thickness of the pipeline edges, making them more prominent.
- **Filling Holes**: If there are small gaps or holes within the pipeline structure, dilation can fill them, resulting in a more complete representation of the pipeline.

Erosion:

- It involves the removal of pixels at the edges of the region.
- It decreases the size of the objects.
- It removes the small anomalies.
- It reduces the brightness of the bright objects.
- It removes the objects smaller than the structuring element.
- It is used later in the Closing operation and prior in the Opening operation.
- It is the dual of dilation.

Purpose in Underwater Pipeline Detection:

- **Noise Reduction**: Underwater images often contain noise due to water turbidity, lighting variations, and sensor limitations. Erosion helps remove small noise particles by reducing the size of bright regions (white areas) in the image.
- **Object Segmentation**: When pipelines are partially covered by sediment or other debris, erosion can separate the pipeline from the surrounding clutter. It helps isolate the pipeline structure by eroding away unwanted pixels.
- **Boundary Smoothing**: Erosion smooths the edges of objects, making them more regular. In the context of pipelines, this can help improve the accuracy of subsequent edge detection or segmentation algorithms.

5. Hough Transform and Probabilistic Hough Transform [23][24]

Hough Transform:

The Hough Transform is a technique used to detect lines or other parametric shapes in an image. Specifically, it's commonly applied to identify straight lines.

Here's how it works:

For each edge pixel (usually obtained from an edge detection algorithm like Canny), the Hough Transform converts the pixel's coordinates from Cartesian space (x, y) to a parameter space (usually represented as ρ and θ).

In the parameter space, each edge pixel contributes to a sinusoidal curve (a line in polar coordinates).

Accumulating these curves results in peaks that correspond to the parameters of the detected lines.

Usefulness for Underwater Pipeline Detection:

The Hough Transform can robustly detect straight pipelines even when they are partially occluded or broken.

It works well for detecting long, continuous pipelines by identifying their centrelines.

Researchers have applied the Hough Transform to extract pipeline contours and improve overall pipeline inspection [25].

Probabilistic Hough Transform:

The Probabilistic Hough Transform is an optimization of the standard Hough Transform.

Unlike the standard version, which considers all edge points, the probabilistic variant randomly selects a subset of edge points.

Here's why it's useful:

Efficiency: By using only a subset of points, it significantly reduces computation time.

Line Segments: Instead of detecting entire lines, the probabilistic version identifies line segments (start and end points).

Robustness: It's less sensitive to noise and outliers.

Application in Underwater Pipeline Detection:

The Probabilistic Hough Transform can efficiently find line segments representing pipeline boundaries.

It's particularly useful for detecting discontinuous or fragmented pipelines.

Researchers have adapted this method to enhance transmission line detection in complex backgrounds, making it applicable to underwater pipelines as well [26].

Results With New Pipeline:

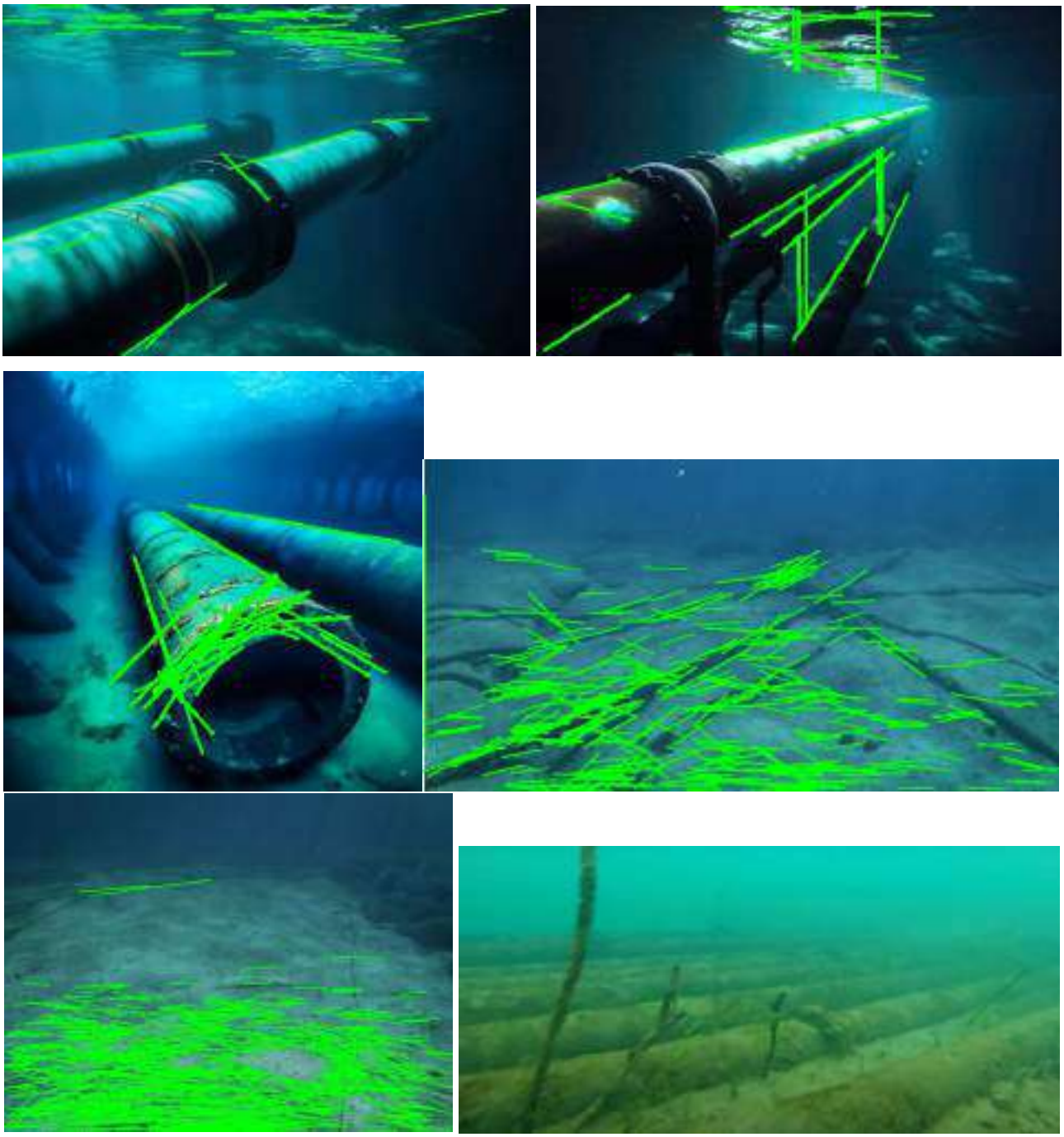


Fig. Results with newly Designed Pipeline

RESULT AND COMPARISON WITH ORIGINAL ONE

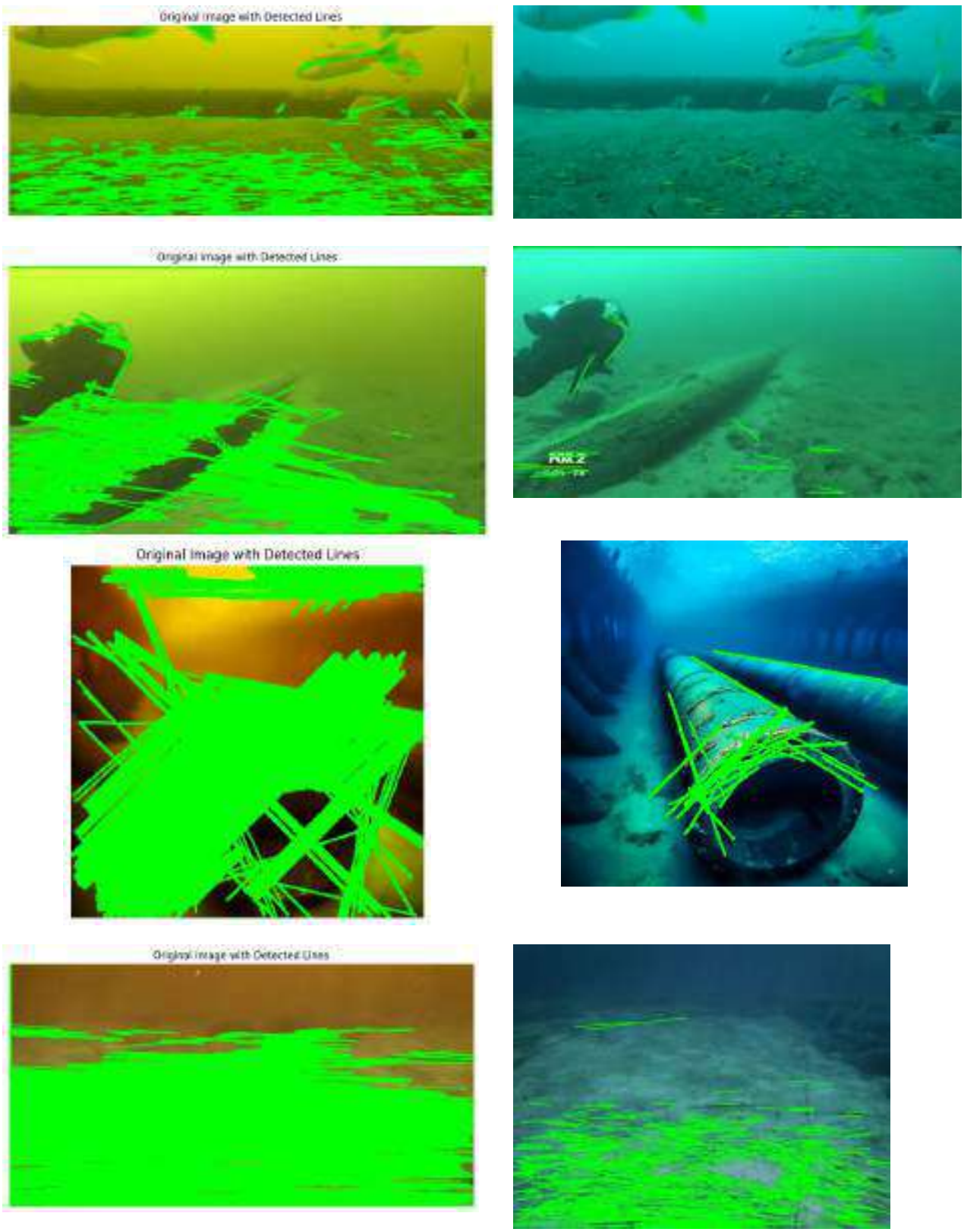


Fig. Comparison of both Pipelines working on various images

- As we can see the Pipeline proposed by the research paper works well on some images and give too much of false detection.
- The newly designed pipeline works quite better and false detection is less in comparison with the research paper's pipeline.

Conclusion:

The proposed research pipeline has been implemented and tested. However, it was found to be ineffective on all images. In response, a new pipeline was designed and tested, demonstrating superior performance. This pipeline relies solely on classical computer vision techniques and does not necessitate pre-labelled dense datasets or powerful processors for training, nor does it require additional navigation modules such as dead reckoning. Through the implementation of the proposed algorithm and visual inspection of the results, it can be concluded that objects in an underwater environment can indeed be detected. This achievement is made possible through the integration of various computer vision algorithms and filters, including grayscale conversion, white balancing, CLAHE, contour detection, adaptive thresholding, Gaussian filtering, ROI selection, and probabilistic Hough transform.

For the More details please go to Github or Drive.

References

[1] Prajwal Bhattarai, Szymon Krupi´nski, Vikram Unnithan, Nicola Secciani, Matteo Franchi, Leonardo Zacchini, Alessandro Ridolfi, and Francesco Maurelli. "A deep learning approach for underwater bubble detection." In IEEE/MTS Oceans 2021, 2021.

[2] Lorik Mucolli, Szymon Krupinski, Francesco Maurelli, Syed Atif Mehdi, and Suleman Mazhar. "Detecting cracks in underwater concrete structures: an unsupervised learning approach based on local feature clustering." In OCEANS 2019 MTS/IEEE SEATTLE, pages 1–8, 2019.

[3] Francesco Maurelli, Szymon Krupi´nski, Xianbo Xiang, and Yvan Petillot. "AUV localisation: a review of passive and active techniques." International Journal of Intelligent Robotics and Applications, page 246–269.

[4] F. Maurelli, A. Mallios, D. Ribas, P. Ridao, and Y. Petillot. "Particle filter based AUV localization using imaging sonar." In IFAC Proceedings Volumes (IFAC-PapersOnline), volume 42, pages 52–57, 2009.

[5] Arjuna Balasuriya and Tamaki Ura. "Vision-based underwater cable detection and following using auvs." In OCEANS'02 MTS/IEEE, volume 3, pages 1582–1587. IEEE, 2002.

[6] Mehdi Fatan, Mohammad Reza Daliri, and Alireza Mohammad Shahri. "Underwater cable detection in the images using edge classification based on texture information." Measurement, 91:309–317, 2016.

[7] Javier Antich and Alberto Ortiz. "Underwater cable tracking by visual feedback." In Iberian Conference on Pattern Recognition and Image Analysis, pages 53–61. Springer, 2003.

[8] Rangachar Kasturi and Octavia I Camps. "Wire detection algorithms for navigation." 2002.

[9] Guangjian Yan, Chaoyang Li, Guoqing Zhou, Wuming Zhang, and Xiaowen Li. "Automatic extraction of power lines from aerial images." IEEE Geoscience and remote sensing letters, 4(3):387–391, 2007.

[10] Joshua Candamo and Dmitry Goldgof. "Wire detection in low-altitude, urban, and low-quality video frames." In 2008 19th International Conference on Pattern Recognition, pages 1–4. IEEE, 2008.

[11] John Canny. "A computational approach to edge detection." IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI- 8(6):679–698, 1986.

[12] Zhao Xu, Xu Baojie, and Wu Guoxin. "Canny edge detection based on open cv." In 2017 13th IEEE international conference on electronic measurement & instruments (ICEMI), pages 53–56. IEEE, 2017.

[13] Richard Szeliski. "Computer vision: algorithms and applications." Springer Science & Business Media, 2010.

[14] James R Bergen and Haim Shvaytser. "A probabilistic algorithm for computing hough transforms." Journal of algorithms, 12(4):639–656, 1991.

[15] Iain Macdonald. "Probabilistic hough transform." Notes, pages 1–4, 2016.

[16] RGB to LAB Color Space Conversion: Formulas, Insights, and Applications | by Alakh Sharma | Mar, 2024 | Medium

[18] https://www.ijcoe.org/article_189415_a4f504dfd47a6e46a961ff57ea095f4c.pdf

[19] https://towardsai.net/p/machine-learning/gaussian-blurring-a-gentle-introduction

[20] http://www.guolab.org/Papers/2017/ICMA2017-243.pdf

[21] Canny Edge Detection Step by Step in Python — Computer Vision | by Sofiane Sahir | Towards Data Science

[22] Difference between Dilation and Erosion - GeeksforGeeks

[23] Standard and probabilistic Hough transform | dummerAugust

[24] macdonald.pdf (ed.ac.uk)

[25] Subsea Pipeline Inspection Based on Contrast Enhancement Module | SpringerLink

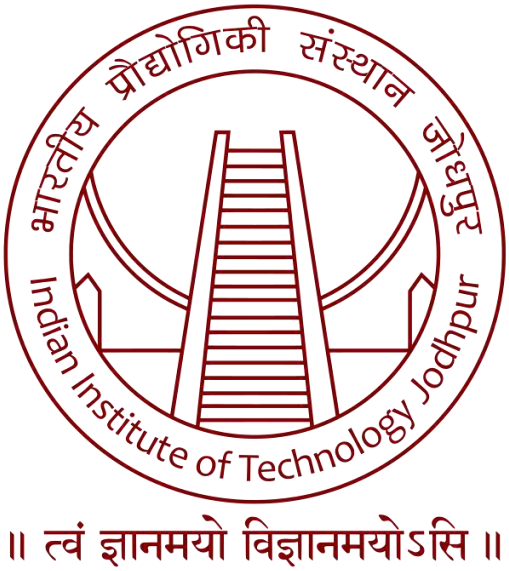
[26] 2402.02761.pdf (arxiv.org)

GitHub - lzyuan168/multimodal_biometric_authentication

[27] N. M. S. Pradhan and F. Maurelli, "A Classical Computer Vision Pipeline for Underwater Detection of Long, Flexible, and Highly Deformable Curvilinear Objects," OCEANS 2022, Hampton Roads, Hampton Roads, VA, USA, 2022, pp. 1-5, doi: 10.1109/OCEANS47191.2022.9977280.

EEL 7150: Embedded System Design

Project Report



On-board Word Recognition and Sensor Monitoring System for Relay Control

Submitted By

Arashdeep Singh (M23IRM003)

Chaitanya Patil (M23IRM004)

Parag Chourasia (M23IRM010)

Contents:-

Introduction

Problem Statement

In this project, we are trying to implement a word recognition system to control appliances.The main purpose of the project is to use a low-power microcontroller with more computational power and capability to run machine learning models, which will help to control many devices. The major benefit of this project will be that a low power device can control high power devices. So that they donot have to continuously look for user commands. Even a small device can help them to work smarter.

Here is the main segemenrts on which our project relies.:-

- **Intelligent Voice Control**: Integrate a speech recognition engine optimized for microcontrollers, allowing users to control appliances through spoken commands.
- **Sensor-Driven Automation**: Equip the system with relevant sensors (temperature, light, etc.) to gather real-time environmental data. This data can be used independently to trigger actions (e.g., light turns on with motion) or combined with voice commands for more sophisticated control.
- **Relay Power Management**: Utilize relays to bridge the gap between the microcontroller's low-power signals and the higher power requirements of appliances. Ensure chosen relays are properly sized for safe operation.

Benefits:-

- **Low-Power Efficiency**: The project capitalizes on a low-power microcontroller, promoting energy-conscious smart home automation.
- **Scalability and Versatility**: This system can be adapted to control various appliances by incorporating different sensors and tailoring the machine learning model for specific use cases.
- **Enhanced User Experience**: Voice control and sensor-driven automation offer a convenient and intuitive way to manage your home environment.

Device chosen :-

Arduino Nano 33 BLE Sense Rev 2

The Arduino Nano 33 BLE Sense Rev2 combines a tiny form factor, different environment sensors and the possibility to run AI using TinyML and TensorFlow™ Lite. Whether you are looking at creating your first embedded ML application or you want to use Bluetooth® Low Energy to connect your project to your phone, the Nano 33 BLE Sense Rev2 will make that journey easy.

Fig Arduino Nano 33 BLE Sense Rev 2

Why we use this device?

1. Low-Power Efficiency:
    - The Nano 33 BLE Sense is designed to be power-efficient, making it suitable for battery-powered or energy-conscious applications. It features a low-power ARM Cortex-M4 processor and power management capabilities that help minimize energy consumption.
2. Computational Power:
    - Despite its compact size, the Nano 33 BLE Sense packs a punch in terms of computational power. It is equipped with a powerful ARM Cortex-M4 processor running at 64 MHz, providing sufficient processing power to execute machine learning models for speech recognition and sensor data processing.
3. Sensor Integration:
    - The Nano 33 BLE Sense comes with built-in sensors that are relevant to your project, including an accelerometer, gyroscope, temperature sensor, humidity sensor, pressure sensor, and microphone. These sensors enable the device to gather real-time environmental data necessary for sensor-driven automation.
4. Bluetooth Connectivity:
    - As indicated by its name, the Nano 33 BLE Sense features Bluetooth Low Energy (BLE) connectivity. This allows the device to communicate wirelessly with other BLE-enabled devices, such as smartphones or smart home hubs, expanding its capabilities and enabling remote control and monitoring of appliances.

5. Machine Learning Support:

    - The Nano 33 BLE Sense supports machine learning applications through the integration of the TensorFlow Lite for Microcontrollers library. This allows you to deploy machine learning models directly onto the device, enabling intelligent voice control and other AI-driven functionalities without relying on external processing.

6. Relay Power Management:

    - While the Nano 33 BLE Sense itself does not include relay modules, it can interface with external relay modules or transistors to control appliances with higher power requirements. Its GPIO pins can be used to send control signals to relays, allowing safe operation and management of appliances.

We have decided to divide our project in four segments. So it becomes easy, and all of us can work on different parts simultaneously.

Work Flow:-

1. Data Collection and Formatting
2. Model Training Testing
3. Model Deployment
4. Hardware Integration

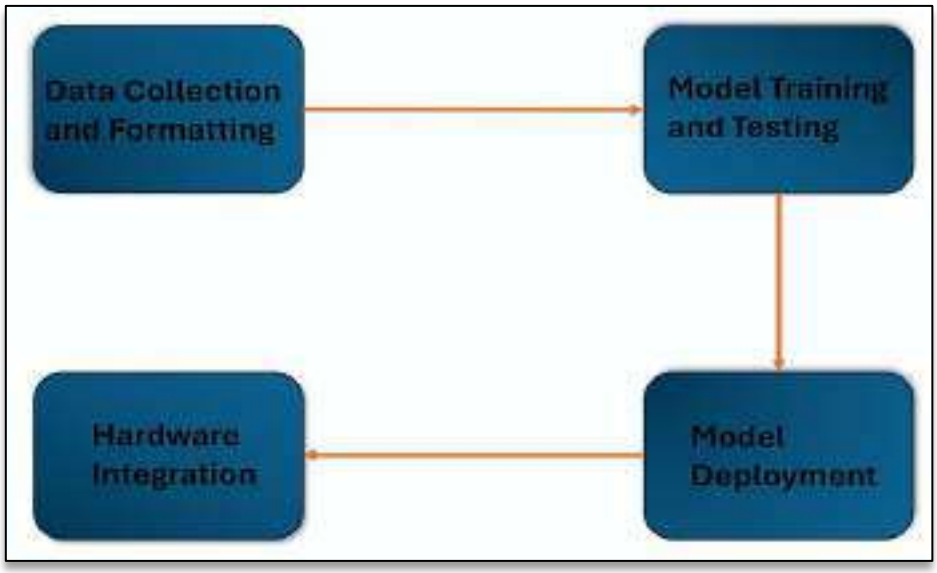


Fig Workflow of the Project

1. Data Collection and Formatting

A. Data Collection

- Voice samples from our classmates for the following keywords.:-
    - Light On, Light Off, Temperature, Pressure, Jalao(जालाओ),Bujao(बुजाओ) is collected.

- Simply phone recorder is used to collect samples. Here, participants were asked to speak keywords in different ways so that a robust model can be prepared by training on various forms of same command.

B. DataSet Making

We have used audacity software in this process. First, we have the collected samples in 1-second frames, for each time. The file format after this we got is .wav. After that we have got roughly 100 files for each keyword.

- Adding Noise
    - To make model robust so that it can sustain noisy situations we have mixed noise with collected samples so that it can work more robustly.
- Multiplexing dataset
    - As to get good performance from model it is advisable to train model on large data and data should be repeated. Going by this logic we have multiplexed it by 15. So we have got 1500 files for each segmentated audio file.
- We have used a Python tool to implement this thinking. The python code is attached with report.
    - Using noiseandmultiplexing.py

Here are the steps of how it works:

1. It takes a list of target words, directories containing audio samples of those words, a directory containing background noise files, and other options as input.

2. It creates a number of subdirectories in the output directory, one for each target word and one for unknown words.

3. It mixes audio samples from the background noise directory with audio samples from the target words directories. The mixed audio samples are then saved in the corresponding subdirectory in the output directory.

4. It also creates a subdirectory for unknown words and mixes audio samples from the background noise directory with audio samples from unknown words directories.

The reason for doing this is to create a dataset of audio samples that are more representative of the real world, where spoken words are often heard in the presence of background noise. This can help to improve the accuracy of keyword spotting models.

2. Model Training Testing

A. Model Making

The model takes in one second's worth of data at a time. It outputs four probability scores, one for each of the chosen classes, predicting how likely it is that the data represents one of them. However, the model does not take in raw audio sample data. Instead, it works with log power spectrums.

Used signal processing to extract feature.

Window size means duration of audio file which is taken as 1 sec.

Frequency of each audio file is 16000Hz.

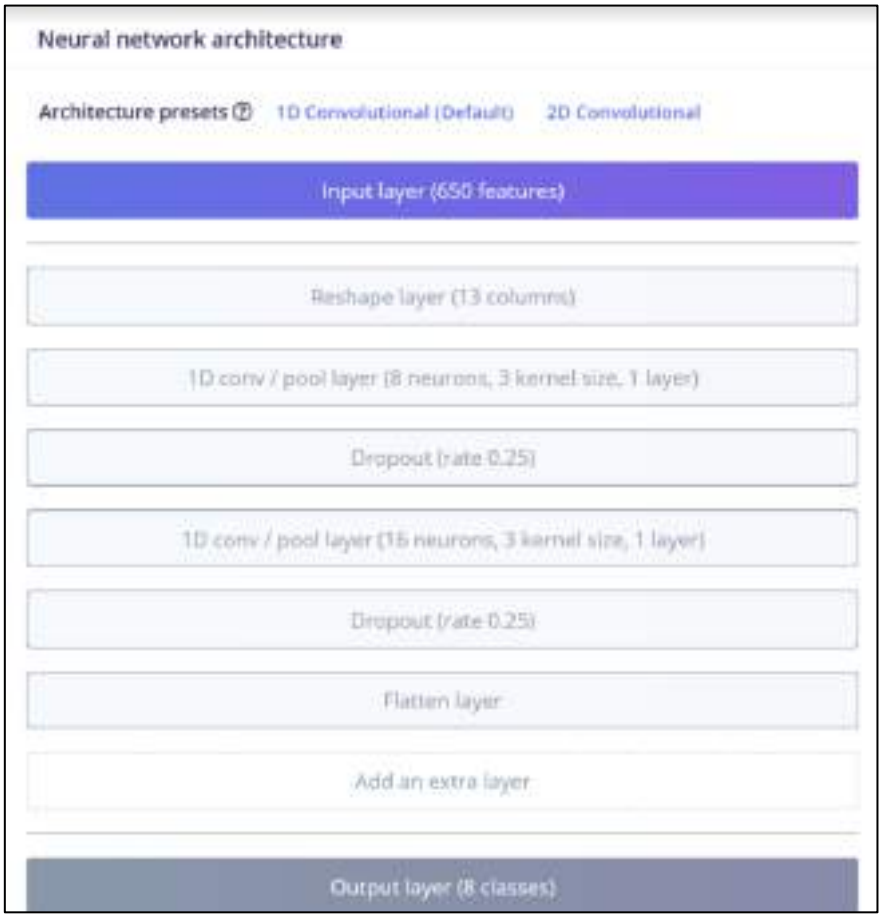


Fig Neural Network Architecture

The layers of model are :-

- **Reshape Layer:**
  - Reshapes the input data into a suitable format for the convolutional layers.
- **Conv1D Layer (with 8 filters):**
  - Applies convolution operation with 8 filters and a kernel size of 3.
- **MaxPooling1D Layer**:
  - Performs max pooling operation with a pool size of 2 and strides of 2.
- **Dropout Layer**:
  - Applies dropout regularization to prevent overfitting (dropout rate of 0.25).
- **Conv1D Layer (with 16 filters):**
  - Another convolutional layer with 16 filters and a kernel size of 3.
- **MaxPooling1D Layer:**
  - Another max pooling operation with the same parameters as before.
- **Dropout Layer**:
  - Another dropout layer with the same dropout rate.
- **Flatten Layer**:
  - Flattens the output of the previous layers into a one-dimensional vector.
- **Dense Layer (Output Layer):**
  - Fully connected layer with 'classes' number of neurons, using softmax activation for multi-class classification.

So, the network architecture has a total of 8 layers:

1) Conv1D layers (2)
2) MaxPooling1D layers (2)
3) Dropout layers (2)
4) Flatten layer (1)
5) Dense (Output) layer (1)

The number of neurons in each layer is as follows:

- Input Layer: Depends on the length of the input data, determined by input_length.
- Conv1D Layer 1: 8 filters
- MaxPooling1D Layer 1: No neurons, as it performs pooling.
- Dropout Layer 1: No neurons, as it performs regularization.
- Conv1D Layer 2: 16 filters
- MaxPooling1D Layer 2: No neurons, as it performs pooling.
- Dropout Layer 2: No neurons, as it performs regularization.
- Flatten Layer: Flattens the output of the previous layer.
- Dense (Output) Layer: 'classes' number of neurons, which is the number of output classes for classification.

How model will work?

It takes audio samples and converts them to MFCC files. So when this model is deployed it converts the given audio input to MFCC. Now, input converted MFCC files will be compared with already MFCC files with which the model is trained. Here is very simplified flow describing working of model:-

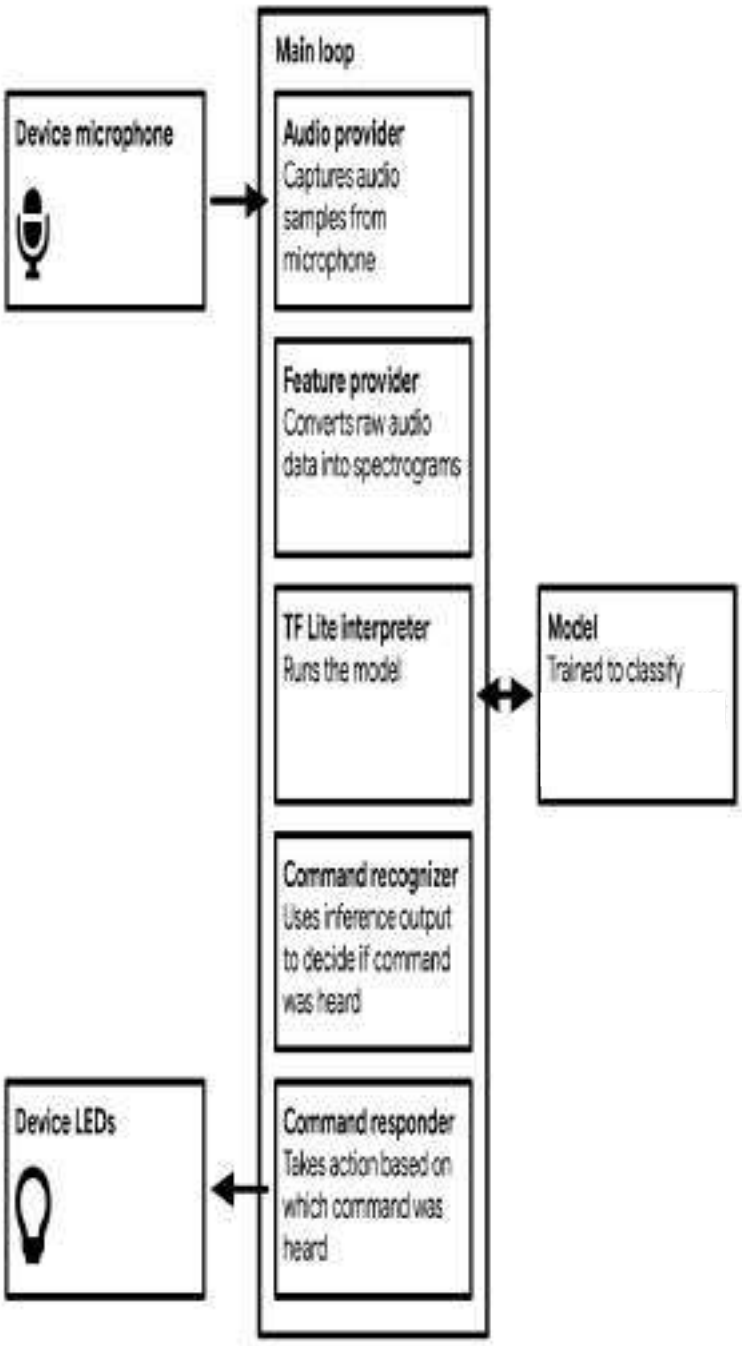


Fig Block Diagram of the Audio Recognition

How audio is converted to MFCC files?

MFCC (Mel-Frequency Cepstral Coefficients) is a feature extraction technique widely used in speech and audio processing. It captures the spectral characteristics of sound in a way that is well-suited for various machine-learning tasks, such as speech recognition and music analysis. In simpler terms, MFCCs are a set of coefficients that capture the shape of the power spectrum of a sound signal. MFCC stands for it is an acronym for **Mel Frequency Cepstral Co-efficients** which are the coefficients that collectively make up an MFC. **MFC** is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. This is similar to JPG format for images.

Here is a step-by-step breakdown of the MFCC technique :

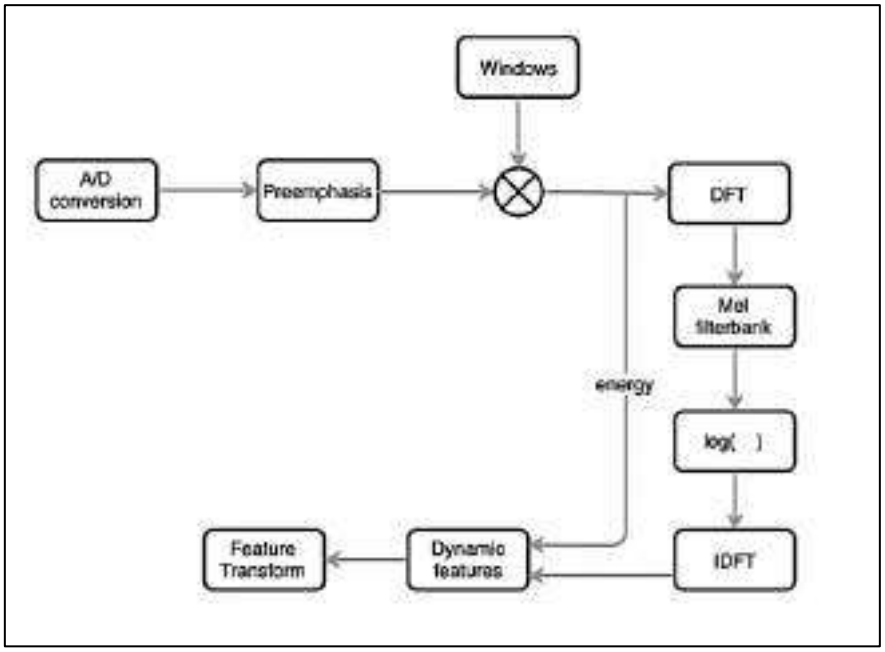


Fig Flowchart of the MFCC technique

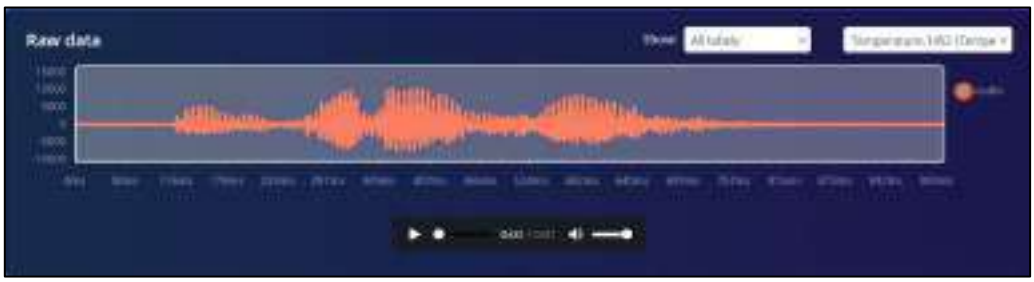


Fig Raw Audio file

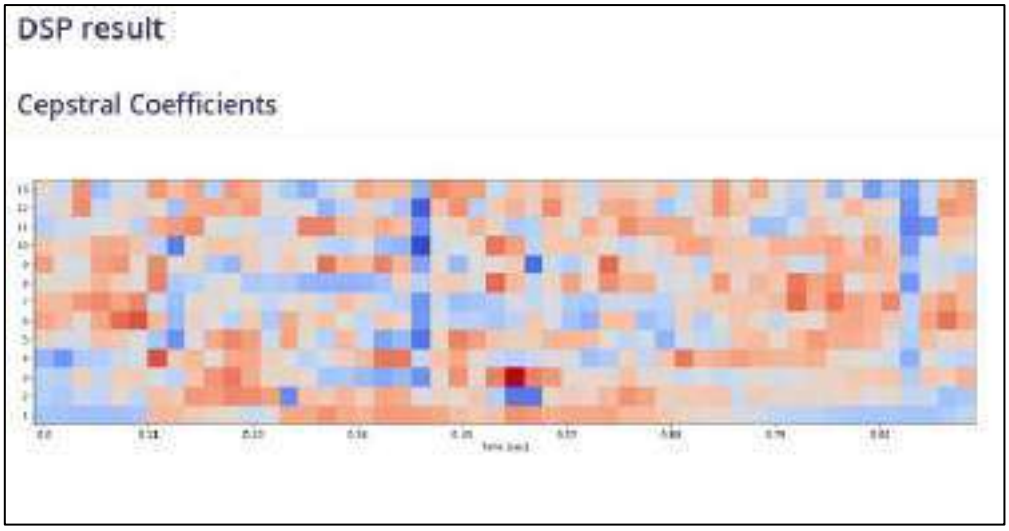


Fig MFCC Final Output

B. Model Training

Here basically collected data set is fed to created model. 80% of all keyword data will be fed at this stage.

The model adjusts its internal parameters to minimize the difference between its predictions and the actual values (the labels of the training data). Classification of keyword file using supervised learning. Edge impulse platform with tensor flow lite is used for this purpose.

A code for model training is provided in this link.

This code is a Python script designed to train a convolutional neural network (CNN) using TensorFlow and Keras. It begins by importing the necessary libraries, including TensorFlow and modules from Keras for building and training neural networks. Key constants such as the number of epochs, learning rate, batch size, and a flag for ensuring deterministic behavior during training are defined. Following this, the script processes the training and validation datasets, shuffling them if non-deterministic behavior is allowed and batching them according to the specified batch size. The model architecture is then defined using a Sequential model from Keras, comprising convolutional layers with max pooling and dropout for feature extraction, followed by a dense layer with softmax activation for classification. An Adam optimizer is employed for training, and a custom callback, likely for logging training progress, is appended. The model is compiled with categorical cross-entropy loss and trained using the fit() method, with validation data provided for monitoring performance. Additionally, there's a flag to enable or disable per-channel quantization for the model, potentially impacting memory usage and accuracy. Overall, this script encapsulates the process of building, training, and evaluating a CNN for classification tasks using TensorFlow and Keras.

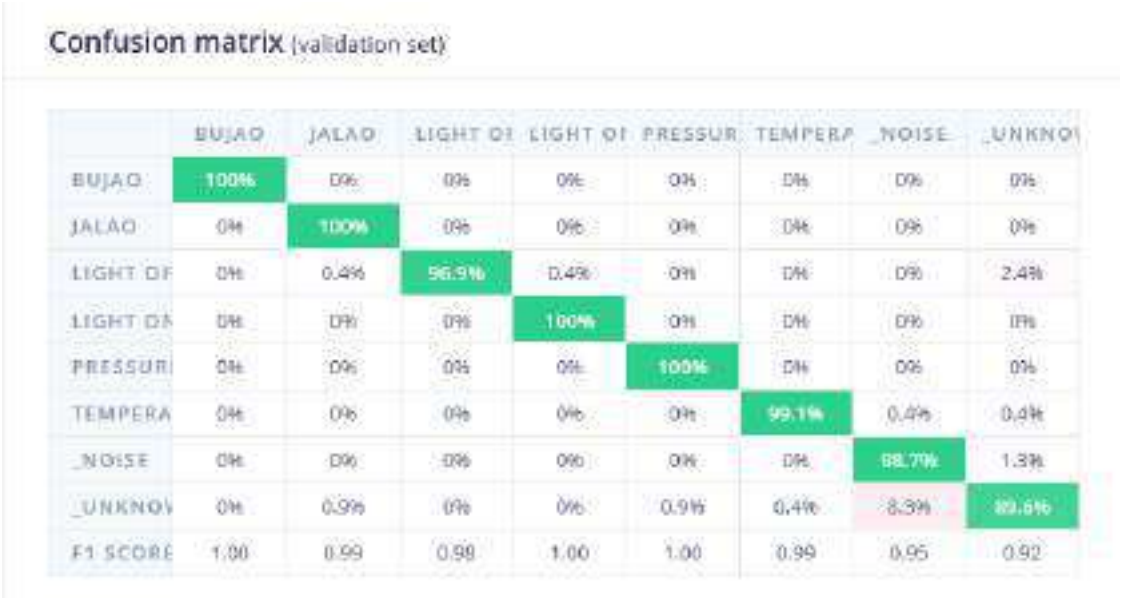


Fig Confusion Matrix for Validation Set

A confusion matrix is a performance measurement tool for machine learning classification algorithms. It is a table that is used to evaluate the performance of a classification model by summarizing the number of correct and incorrect classifications for each class.

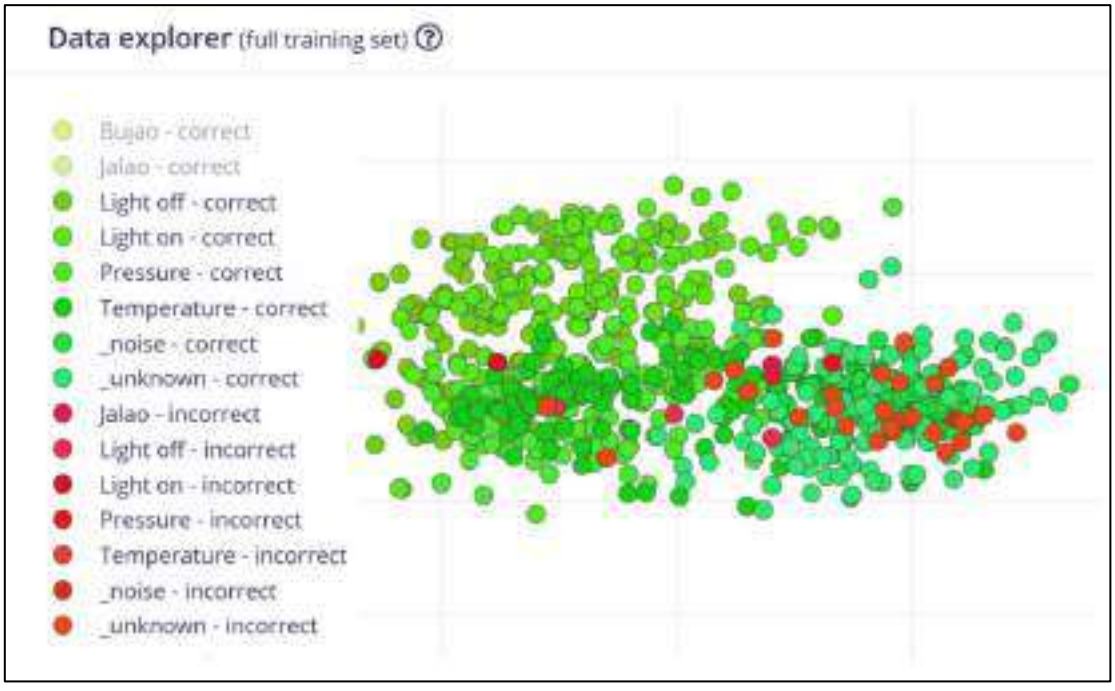


Fig Point chart showing the data of Full Training Set.

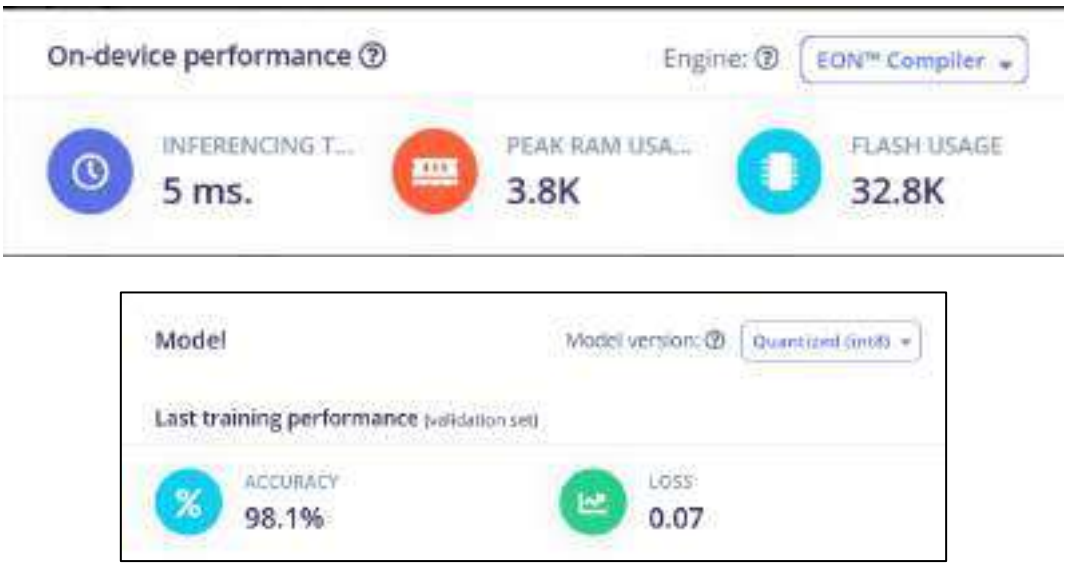


Fig Model Accuracy and On-device Performance

C. Model Testing

Rest 20% keyword data will be fed to check accuracy and result as giving a test to created system.This data is not used during training, so it provides a measure of how well the model can generalize to new, unseen data.

Fig Confusion matrix showing Overall Accuracy

3. Model Deployment

To deploy the model on an Arduino Nano 33 BLE Sense board, you'll need to convert it into a format compatible with the board's hardware constraints. This may involve optimizing the model architecture and parameters for efficient execution on the microcontroller.

Inside the microcontroller, the deployed model will process incoming audio samples captured by the onboard microphone.

Word detection involves continuously monitoring the audio input for specific keywords ("Light On," "Light Off," etc.). When a match is found, the microcontroller can trigger relevant actions, such as turning on a light or adjusting temperature.

A typical model deployment needs to consider these things:-

- Model Conversion
- Library Setup
- Initialization
- Audio Input
- Preprocessing
- Model Inference
- Wake Word Detection
- Action Trigger
- Edge impulse takes care of many things automatically.

Probabilty based Thresholding is used to identify the given input. There is possibility that a given input may have matches with a few trained keywords. Then this thresholding helps to decide which one is dominating. The value of threshold varies according to the environment. In silent environment it should be kept high. We have chosen 0.7 as threshold. If match is more than 0.7 then result will be declared that user has given this command. We have also created a category as unknown. If the user says something that the model does not understand, then it will be kept in the category of the unknown in the keywords.

Fig Serial Monitor of Arduino IDE showing the showing the word and its probability

4. Hardware Integration

- SENSOR INTERFACING

Microphone

MP34DT05-A (MEMS audio sensor omnidirectional digital microphone)

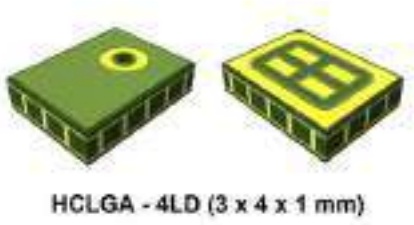


Fig Microphone Sensor

The MP34DT05-A is an ultra-compact, low-power, omnidirectional, digital MEMS microphone built with a capacitive sensing element and an IC interface. The sensing element, capable of detecting acoustic waves, is manufactured using a specialized silicon micromachining process dedicated to producing audio sensors. The IC interface is manufactured using a CMOS process that allows designing a dedicated circuit able to provide a digital signal externally in PDM format.
The sensing element shall mean the acoustic sensor consisting of a conductive movable plate and a fixed plate placed in a tiny silicon chip. This sensor transduces the sound pressure into the changes of coupled capacity between those two plates.

Pulse Density Modulation (PDM) is a form of modulation used to represent an analog signal with a binary signal. In a PDM signal, specific amplitude values are not encoded into codewords of pulses of different weight as they would be in pulse-code modulation (PCM); rather, the relative density of the pulses corresponds to the analog signal's amplitude. The output of a 1-bit DAC is the same as the PDM encoding of the signal.

In a PDM bitstream, a 1 corresponds to a pulse of positive polarity (+A), and a 0 corresponds to a pulse of negative polarity (-A). A run consisting of all 1s would correspond to the maximum (positive) amplitude value, all 0s would correspond to the minimum (negative) amplitude value, and alternating 1s and 0s would correspond to a zero amplitude value. The continuous amplitude waveform is recovered by low-pass filtering the bipolar PDM bitstream.

Temperature and Humidity Sensor

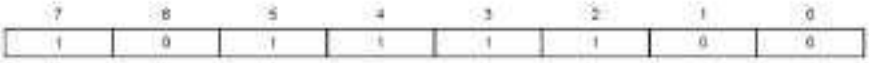
HTS221 (Capacitive digital sensor for relative humidity and temperature). The HTS221 is an ultra-compact sensor for relative humidity and temperature. It includes a sensing element and a mixed signal ASIC to provide the measurement information through digital serial interfaces.

The sensing element consists of a polymer dielectric planar capacitor structure capable of detecting relative humidity variations and is manufactured using a dedicated ST process.

Fig Temperature and Humidity Sensor.

Register which are being used for reading.

- **WHO_AM_I (0Fh)**
  Device Identification



This register holds a value which holds the value of device address. Before starting anything it is confirmed that it is the right device.

```
_wire->begin();
if (i2cRead(HTS221_WHO_AM_I_REG) != 0xbc) {
        end();
```

HTS221_CTRL1_REG

One shot mode is enabled using this by configuring bit 0 and 1. The ODR1 and ODR0 bits permit changes to the output data rates of humidity and temperature samples.The default value corresponds to a “one-shot” configuration for both humidity and temperature output. As per datasheet bot ODR bits need to be set to zero to enable one-shot mode.

One-shot mode is triggered and a new acquisition starts when it is required. Enabling this mode is possible only if the device was previously in power-down mode. Once the acquisition is completed and the output registers updated, the device automatically enters in power-down mode. ONE_SHOT bit self-clears itself.

HTS221_CTRL2_REG

One shot mode is enabled using bit 0.

HTS221_STATUS_REG

Bit 0 tells about the data availability of temperature. Bit 1 tells about the data availability of humidity. Whenever a new sample is available, these bits are set to high by the hardware. After reading values these bits are cleared.

HTS221_TEMP_OUT_L

It holds output data LSB of temperature measured. It contains direct digital value automatically converted by ADC.

HTS221_TEMP_OUT_H

It holds temperaturte data measured MSB.

```
INT16_T TOUT = i2cRead16(HTS221_TEMP_OUT_L_REG);
```

Final Temperature RTeading.

```
float reading = (tout * _hts221TemperatureSlope + _hts221TemperatureZero)
```

HTS221_HUMIDITY_OUT_L

It holds output data LSB of Humidity measured. It contains direct digital value automatically converted by ADC.

HTS221_HUMIDITY_OUT_H

It holds humidity data MSB.

Hout from output register

```
int16_t hout = i2cRead16(HTS221_HUMIDITY_OUT_L_REG);
```

```
float reading =hout * _hts221HumiditySlope + _hts221HumidityZero)
```

Pressure Sensor

Arduino nano 33 ble sense has few inbuilt sensors. It has world's smallest pressure sensor.

It has LPS22HB(MEMS nano pressure sensor: 260-1260 hPa absolute digital output barometer) by STMicroelectronics.

Fig Pressure Sensor

We are not discussing much about working of the sensor, rather we are focusing on register and other parts related to our course.

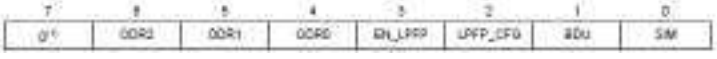
Important registers we were used in our application.

- **WHO_AM_I(0Fh)**
  This register stores the address of the sensor. If I want to use the sensor and sensor is integrated in device. Then First I will find the address of the sensor from the microcontroller datasheet. And before doing anything I will compare that value with the value stored in WHO_AM_I register.
  Here is the code justifying this statement:-

```
{
_wire->begin();
if (i2cRead(LPS22HB_WHO_AM_I_REG) != 0xb1) {
end();
return 0;
}
_initialized = true;
return 1;
}
```

  *0Fh is the address of the register in the sensor.

  CTRL_REG1(10h)
  This register helps us to select output data rate, Low-pass configuration,SPI Serial Interface Mode selection.



Here, we will control three bits, 4,5,6 to select the output data rate. In this case we have used one shot mode. According to the datasheet, the sensor needs to set all three bits to zero to enable one-shot mode.
One Shot Mode,
If the ONE_SHOT bit in CTRL_REG2 (11h) is set to '1', One-shot mode is triggered and a new acquisition starts when it is required. Enabling this mode is possible only if the device was previously in power-down mode (ODR bits set to '000'). Once the acquisition is completed and the output registers updated, the device automatically enters in power-down mode. ONE_SHOT bit self-clears itself.

CTRL_REG2(11h)
Bit 0 of this register represents one_shot mode by default.

Output Value:-
As pressure is analog but we are measuring it digital. So we need to convert analog quantity to digital. But this sensor directly gives us digital value. We do not have to care about ADC conversion. It gives us 24-bit output. As it has 8-bit registers only. It uses 3 registers to display output. Here is a detailed explanation of these registers.

- **PRESS_OUT_XL(28h)**
  It contains the low part of the pressure output value.
- **PRESS_OUT_L(29h)**
  It contains the mid-part of the pressure output value.
- **PRESS_OUT_H(2Ah)**
  It contains the high part of the pressure output value.

```
float reading = (i2cRead(LPS22HB_PRESS_OUT_XL_REG) |
 (i2cRead(LPS22HB_PRESS_OUT_L_REG) << 8) |
        (i2cRead(LPS22HB_PRESS_OUT_H_REG) << 16)) / 40960.0;
if (units == MILLIBAR) { // 1 kPa = 10 millibar
return reading * 10;
} else if (units == PSI) {  // 1 kPa = 0.145038 PSI
 return reading * 0.145038;
} else {
return reading;
}
}
```

How to get value from these registers:-

Pressure Value (LSB) = PRESS_OUT_H (2Ah) & PRESS_OUT_L (29h) & PRESS_OUT_XL (28h)
= 3FF58Dh = 4191629 LSB (decimal signed)

$$\text{Pressure (hPa)} = \frac{\text{Pressure Value (LSB)}}{\text{Scaling Factor}} = \frac{4191629\ \text{LSB}}{4096\ \text{LSB/hPa}} = 1023.3\text{hPa}$$

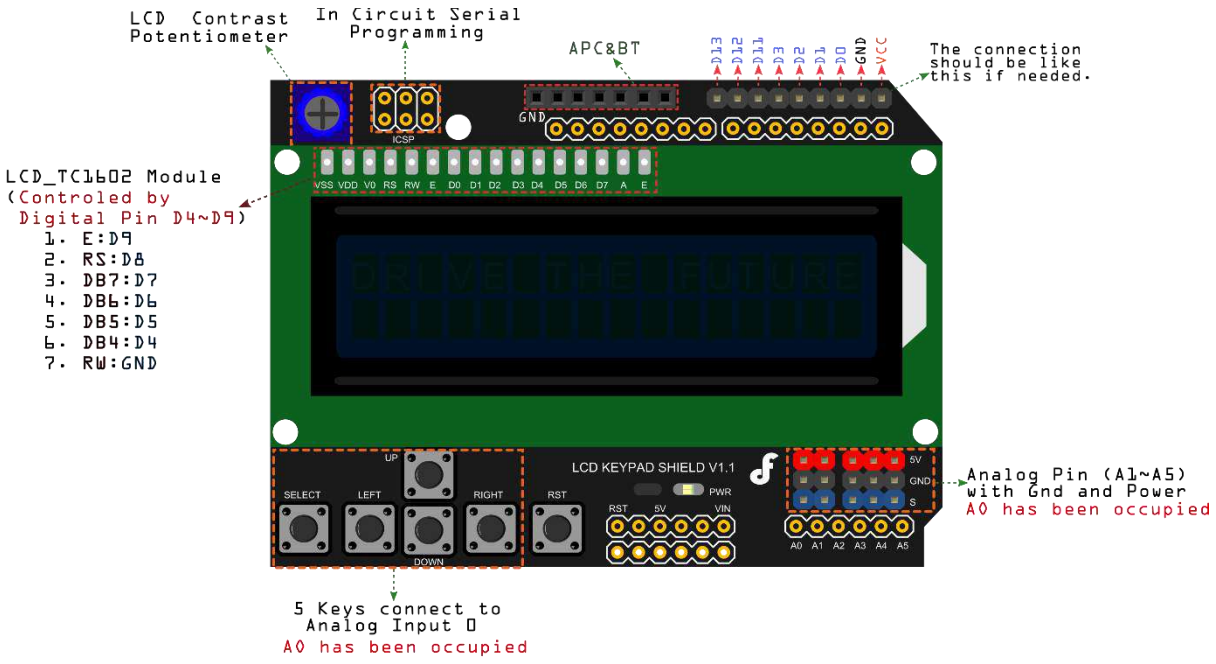
DISPLAY

LCD Keypad Shield

This is a very popular LCD Keypad shield for Arduino. It includes a 2x16 LCD display and 6 momentary push buttons. Pins 4, 5, 6, 7, 8, 9 and 10 are used to interface with the LCD. The LCD shield supports contrast adjustment and backlit on/off functions. It also expands analog pins for easy analog sensor reading and display.

Specification

- Operating Voltage:5V
- 5 Push buttons to supply a custom menu control panel
- RST button for resetting Arduino program
- Integrate a potentiometer for adjusting the backlight
- Expanded available I/O pins
- Expanded Analog Pinout with standard DFRobot configuration for fast sensor extension
- Dimension: 80 x 58 mm



Instruction for D4 To D10 and Analog Pin 0

Pin	Function	Instruction
Digital 4(D4)	D4~D7 are used as DB4~DB7	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the LCD.
Digital 5(D5)		
Digital 6(D6)		
Digital 7(D7)		
Digital 8(D8)	RS	Choose Data or Signal Display
Digital 9(D9)	Enable	Starts data read/write
Digital 10(D10)	LCD Backlight Control	
Analog 0(A0)	Button select	Select, up, right, down and left

Function Explanation

- LiquidCrystal(rs, enable, d4, d5, d6, d7)
    - Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters. for example:
        - LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

- lcd.begin(cols, rows)
    - Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. begin() needs to be called before any other LCD library commands. for example:
        - lcd.begin(16, 2);

- lcd.setCursor(col,row)
    - Set the location at which subsequent text written to the LCD will be displayed. for example:
        - lcd.setCursor(0,0);

- lcd.print(data)
    - Prints text to the LCD.for example:
        - lcd.print("hello, world!");

- lcd.write(data)
    - Write a character to the LCD

RELAY

- This is a LOW Level 5V 2-channel relay interface board, and each channel needs a 15 -20mA driver current.
- It can be used to control various appliances and equipment with large current.
- It is equipped with high-current relays that work under AC 250V 10A or DC 30V 10A.
- It has a standard interface that can be controlled directly by microcontroller.

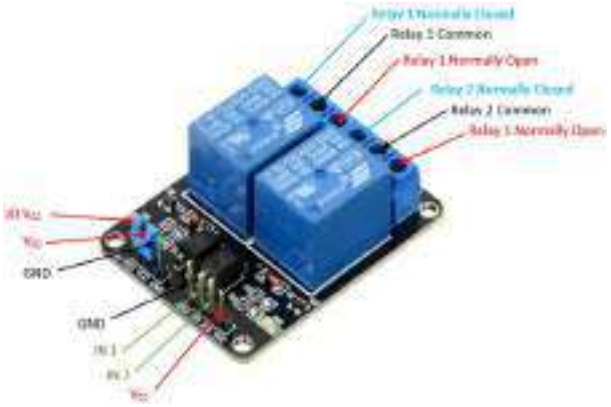


Fig Relay Module

Feature:

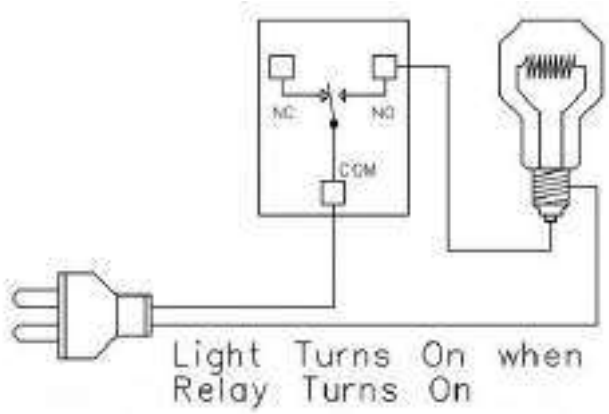
- Relay Maximum output: DC 30V/10A, AC 250V/10A .
- 2 Channel Relay Module with Optocoupler LOW Level Triger expansion board, which is compatible with Arduino.
- Standard interface that can be controlled directly by microcontroller (8051, AVR, *PIC, DSP, ARM, ARM, MSP430, TTL logic) .
- Relay of high quality loose music relays SPDT (Single Pole Double Throw) . A common terminal, a normally open, one normally closed terminal optocoupler isolation, good anti-jamming.
- SPDT

Input:

- VCC : Connected to positive supply voltage (supply power according to relay voltage) ; input for directly powering the relay coils .
- JD-VCC : Input for the isolated power supply for relay coils . (VCC AND JD-VCC are shorted)
- GND : Connected to negative supply voltage ; input ground reference
- IN1: Signal triggering terminal 1 of relay module
- IN2: Signal triggering terminal 2 of relay module

Output:

Each submodular of the relay has one NC(nomalclose), one NO(nomalopen) and one COM(Common). So there are 2 NC, 2 NO and 2 COM of the channel relay in total. NC stands for the normal close port contact and the state without power; No stands for the normal open port contact and the state with power. COM means the common port. You can choose NC port or NO port according to whether power or not.




```
{
    relay_SetStatus(ON, OFF);//turn on RELAY_1
    delay(2000);//delay 2s
    relay_SetStatus(OFF, ON);//turn on RELAY_2
    delay(2000);//delay 2s
}
```

Wiring Diagram

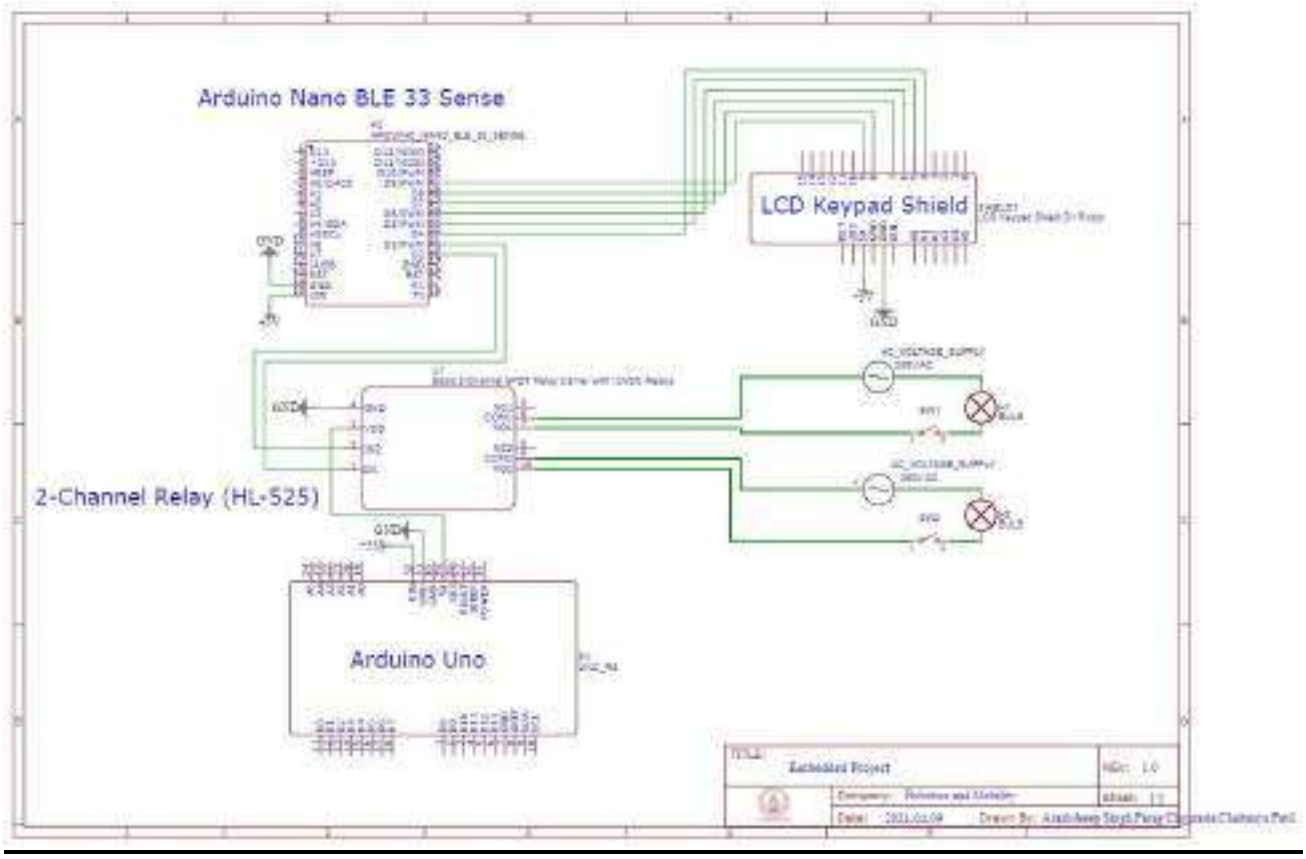


Fig Wiring Diagram

Main Script

We have attached the code as separate files. Here are explaining main segments which are responsible for model deployment.

- Initialising the libraries of Sensors and LCD

```
// Library Setup
#include <PDM.h>
#include <speech_recognition_2_inferencing.h>
#include <Arduino_LPS22HB.h>
#include <Arduino_HS300x.h>
#include <LiquidCrystal.h>

// Initialization
void setup() {
```

```
  Serial.begin(115200);
  HS300x.begin();
  BARO.begin();
  pinMode(relay1, OUTPUT);
  pinMode(relay2, OUTPUT);
  lcd.begin(16, 2);
  // Other initializations...
}

// Audio Input (part of Initialization)
static bool microphone_inference_start(uint32_t n_samples) {
  // PDM initialization...
}

// Model Inference
void loop() {
  bool m = microphone_inference_record();
  signal_t signal;
  signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
  signal.get_data = &microphone_audio_signal_get_data;
  ei_impulse_result_t result = {0};
  EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
  // Actions based on classification results...
}
```

- Condition of if-else for showing the result, by relay and LCD Module

```
  if(result.classification[0].value > thresh){ // ...../// "BUJHAO"
    //digitalWrite(led_red,LOW);   //  BLE Sense is ON when the pin  is LOW
    digitalWrite(relay1,HIGH);  // Turning off the relay
    lcd.print("BUJHAO  ");
    delay(500);
    lcd.clear();

  }else if(result.classification[1].value > thresh){ // ...../// "JALAO"
          //  BLE Sense is ON when the pin  is LOW
    // Turning on the relay
    digitalWrite(relay1,LOW);
```

```
    lcd.print("JALAO  ");
    delay(500);
    lcd.clear();

  }else if(result.classification[2].value > thresh){ // ....../// "LIGHT OFF"
    digitalWrite(relay2,HIGH); // Turning off the relay
```

Hardware Demonstration

We attaching a link of video demonstrating our work. Here we are attaching pictures to give you more idea about project.

https://drive.google.com/drive/folders/1zmc6pTd-Wkw5olhbmeGIsCwXqpTlRqMD?usp=sharing

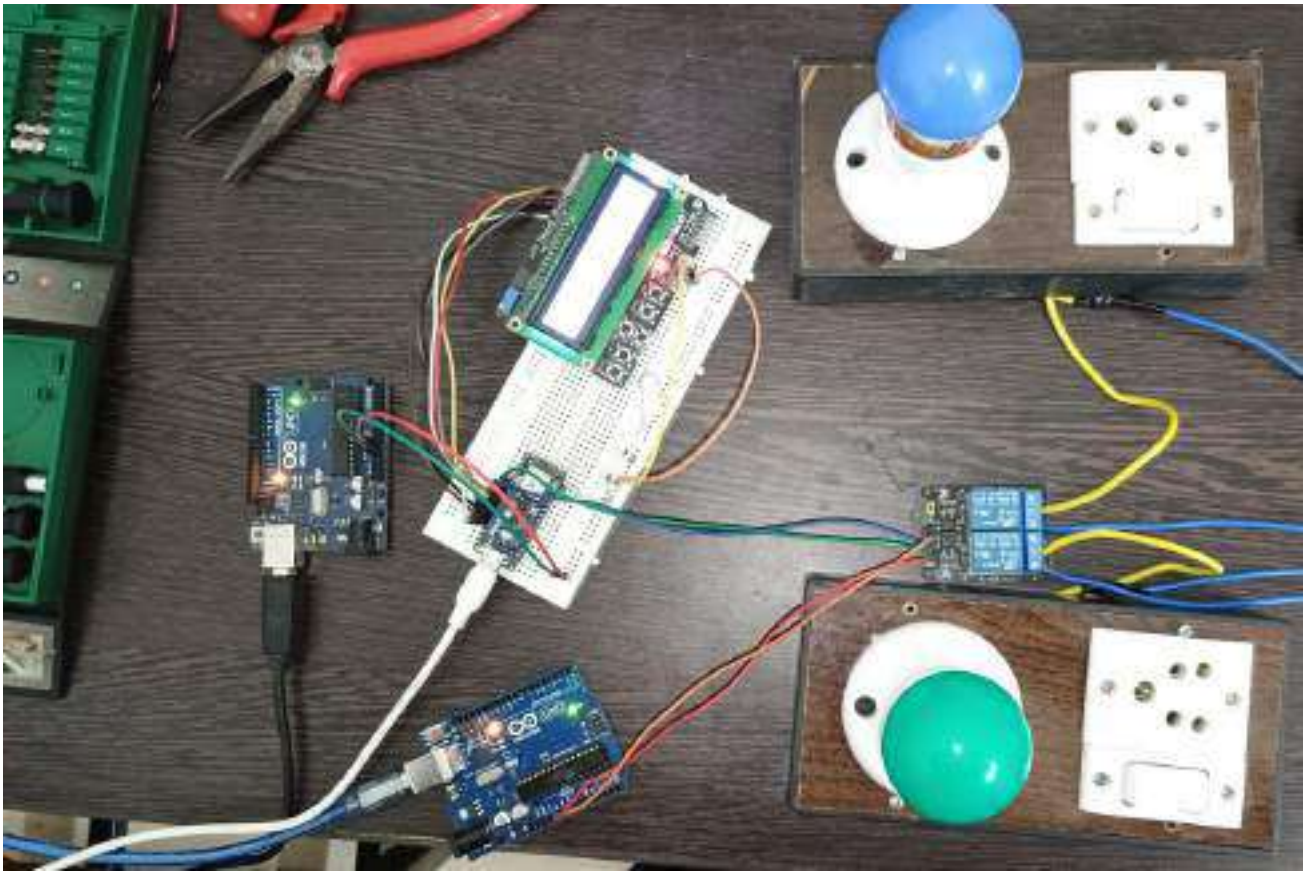


Fig Actual Circuit

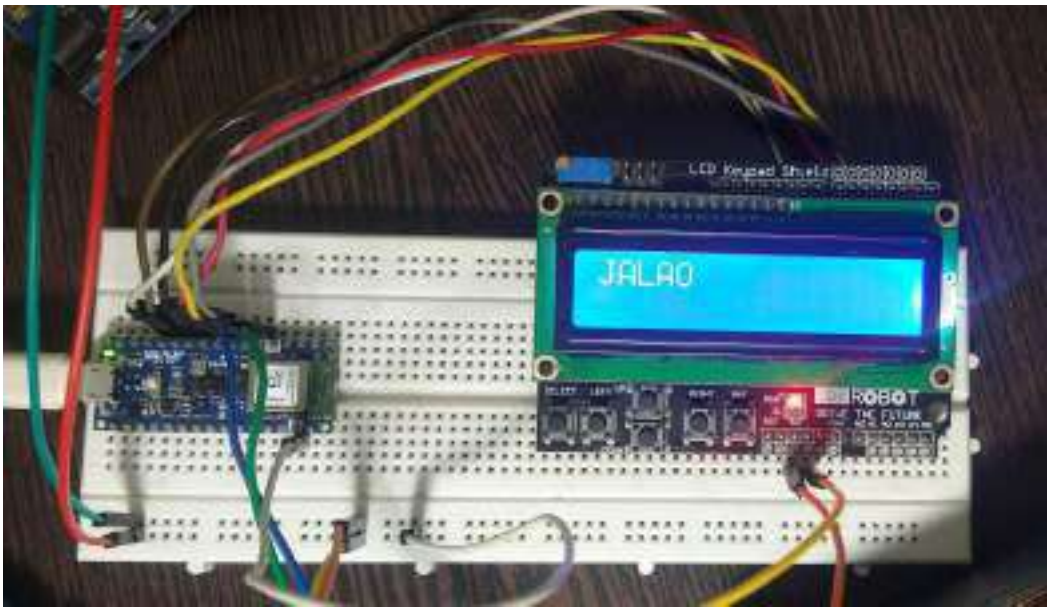


Fig Screen Displaying the result for keyword ‘JALAO’

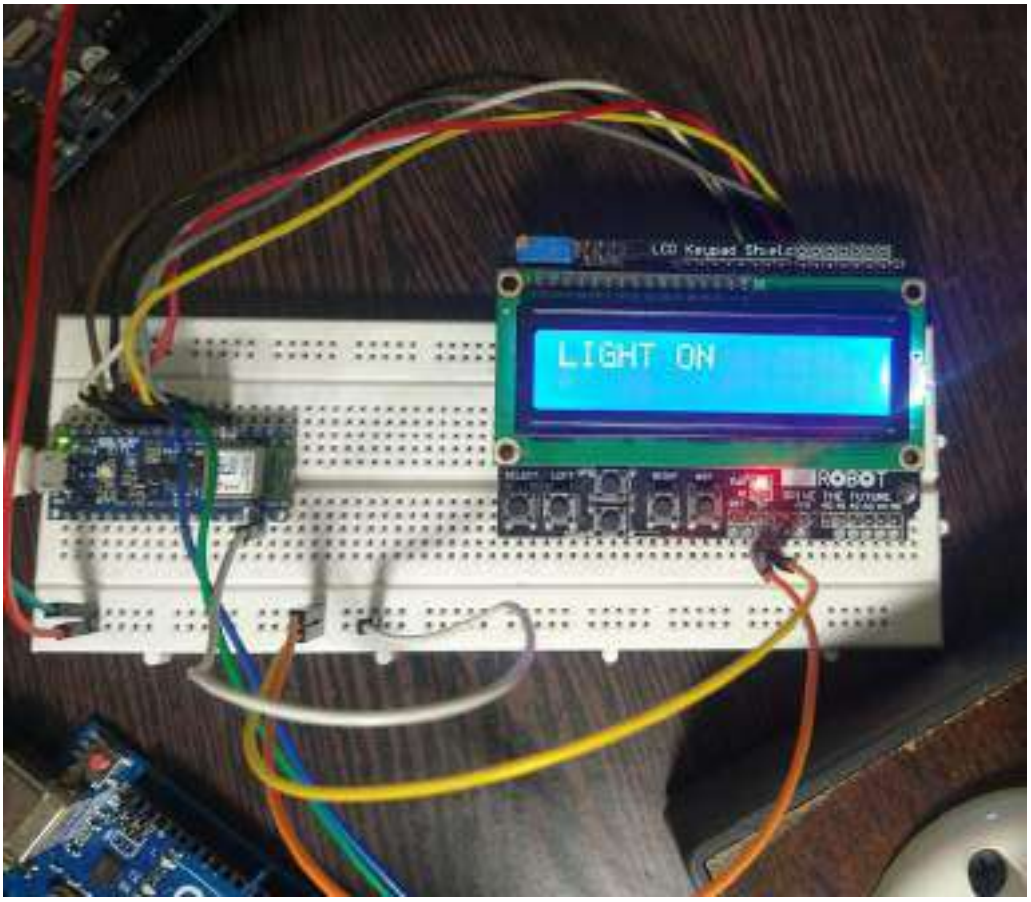


Fig Screen Displaying the result for keyword ‘LIGHT ON’

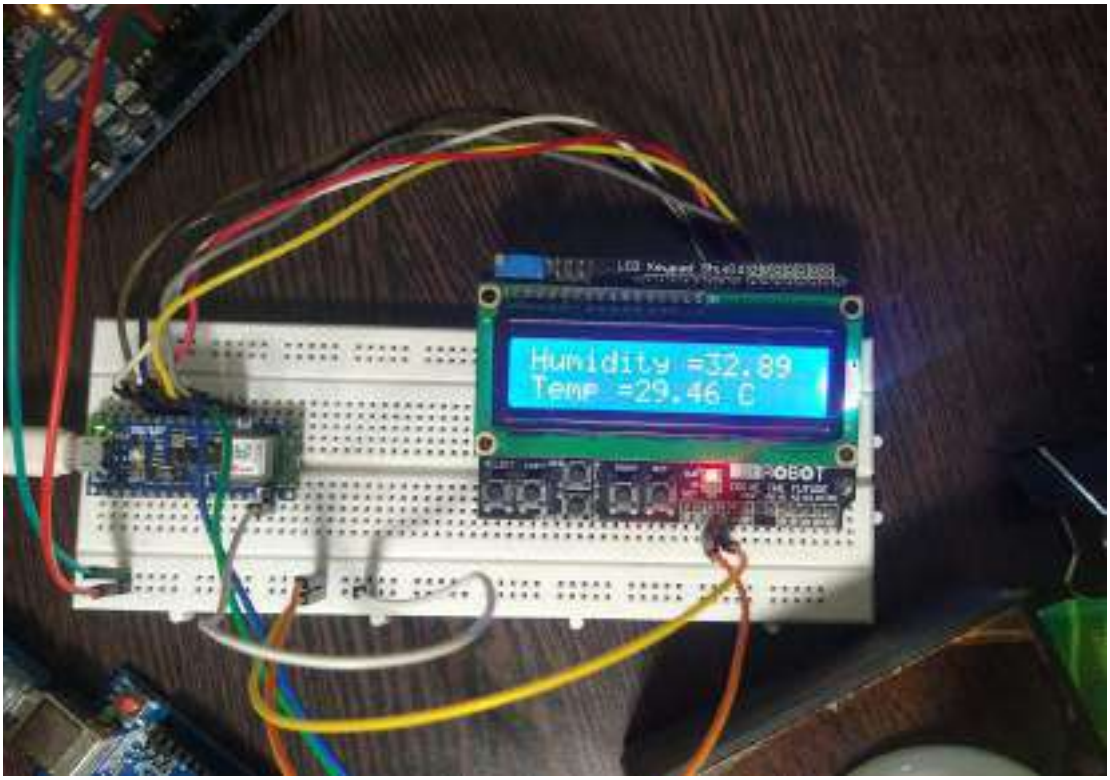


Fig Screen Displaying the result for keyword ‘TEMPERATURE’

WORK DISTRIBUTION

ARASHDEEP SINGH (M23IRM003)	CHAITAYNA PATIL (M23IRM004)	PARAG CHOURASIA (M23IRM010)
Data Collection Model Deployment(Sensors) Documentation	Data Collection Model Deployment(Peripheral) Hardware	Data Set Model Training Model Testing

References:-

Seeing is Believing: Converting Audio Data into Images | by Tony Chen | Towards Data Science

Arduino 4 Relays Shield Basics | Arduino Documentation

MFCC Technique for Speech Recognition - Analytics Vidhya

https://components101.com/switches/5v-dual-channel-relay-module-pinout-features-applications-working-datasheet

https://cdn-reichelt.de/documents/datenblatt/B300/ME114.pdf

MP34DT05-A - MEMS audio sensor omnidirectional stereo digital microphone - STMicroelectronics

Pulse-density modulation - Wikipedia

MFCC (Mel Frequency Cepstral Coefficients) for Audio format (opengenus.org)

Audacity ® | Free Audio editor, recorder, music making and more! (audacityteam.org)

Edge Impulse - The Leading edge AI platform

Leader-Follower Formation Control of Nonholonomic Mobile Robots

Arashdeep Singh
Robotics and Mobility Systems
Indian Institute of Technology,Jodhpur
m23irm003@iitj.ac.in

Venkat Pranav Deshpande
Robotics and Mobility Systems
Indian Institute of Technology,Jodhpur
m23irm013@iitj.ac.in

***Abstract*—This report focuses on the Multi-Agent Control Using Leader-Follower Control Approach. We trying to implement this approach for wheeled mobile robots(non-holonomic). But for that first need to understand kinematics of wheeled mobile robots. Then we will try to implement control strategies. We are focusing on Lyapunov and Feedback linearization based control Strategies.**

I. Introduction

Let's begin by delving into the realm of Multi-agent Control and exploring the reasons for its significance. In a multi-agent system, there are numerous agents, each capable of making decisions autonomously, all striving towards a common goal. Employing multiple agents allows us to tackle intricate problems, enhancing robustness and efficiency. Consider a courier warehouse scenario where multiple parcels need to be picked and placed at various locations. Utilizing a network of interconnected agents can create a fast and accurate system to address such complexities. Now, the pivotal question arises: how do we control these agents?

There are primarily two approaches to multi-agent control when we want that agents to coordinate :

Flocking Control draws inspiration from the collective behavior observed in nature among groups of entities such as birds, fish, or insects. Three key principles govern this approach:

- Alignment: Agents align their movement with the average direction of their neighbors, maintaining a consistent heading.
- Cohesion: Agents are attracted to stay close to their neighbors, fostering a cohesive group formation to prevent dispersion.
- Separation: Agents avoid collisions by maintaining a certain distance from their neighbors, preventing overcrowding.

However, Flocking Control has its disadvantages.

Formation Control, on the other hand, involves the coordinated control of a fleet of robots following a predefined trajectory while maintaining a specific spatial pattern. It addresses problems where agents need to stabilize at a given distance from one another. Applications range from vehicle control to UAVs, consensus and formation control of robots, industrial robots, and more.

Let us redefine Formation : A number of individual physical agents often interact locally with the global behavior desired. Agents display spatial patterns, implying inter-agent interaction. Formation should be autonomous or controlled through a single leader vehicle. We do not want 10 vehicles having 10 different controllers and struggling to maintain a formation. This means every agent acts locally, but the effect is global. To acheive this localization there are basically two ideas:-

- Acquiring a certain shape
- Maintaining a certain shape

Fig. 1. Formation as rigid structure

Certaintly these formations act like rigid frames. As we donot want that distance between pairs does not change. In rigid formation,some distances need to be explicitly maintained with the rest being consequentially maintained.
Now question is do all agents need to maintain distance or only few need to maintain distance. Like If the distance between agents X and Y is maintained,this may be:

- A task jointly shared by X and Y or
- Something that X does and Y is unconscious about or conversely)meaning one agent agent will be responsible to maintain the distance and other will just follow its path).

Fig. 2. Both responsible to maintain formation

Fig. 3. Follower to maintain formation

Certainly second one is better approach as it is easier to control and less costly. Let us now understand these kind of few approaches:

Leader-Follower Structure:
In this method, one agent is designated as the leader, responsible for generating the desired trajectory, while the others act as followers, tracking the leader's motion and maintaining a specific formation shape and distance.
Advantages include ease of implementation and adaptability to dynamic environments.
Challenges include sensitivity to the leader's performance, potential collapse if the leader deviates, and scalability issues.

Virtual Structure:
Agents maintain positions relative to their neighbors, forming a virtual structure defined by the leader's position.

Behavior-Based Structure:
Agents follow predefined behaviors, contributing to the overall formation pattern.

Formation Control can be further classified based on control strategies:
Centralized Control: A single entity controls the entire group, making decisions for each agent.
Decentralized Control: Agents make decisions independently but consider information from nearby agents.
Distributed Control: Agents have limited information but communicate and coordinate with neighboring agents.

We will be following Leader Follower approach and will further illustrate this approach on whelled mobile robots.The leader-follower approach can be implemented as both a centralized and decentralized control method. In a centralized control system, the environment is known and static, and the followers adhere to instructions from a fixed leader or specific sources. Conversely, in a decentralized control system, the leader moves forward and is followed by the followers, each of which operates autonomously and may only require local information. This flexibility allows the leader-follower approach to be adapted to various scenarios, including those where robustness and simplicity are needed, making it suitable for decentralized control applications.For that we will be considering both distance and angle between two agents as parameters that needed to be controlled.

II. PROBLEM FORMULATION

Before we try to control robots, we need to know their kinematics,how different comopenents are resulting in motion, and which are under our control. We will use a Differential Drive WMR to implement control strategies.

A. Kinematics of Differential Drive WMR

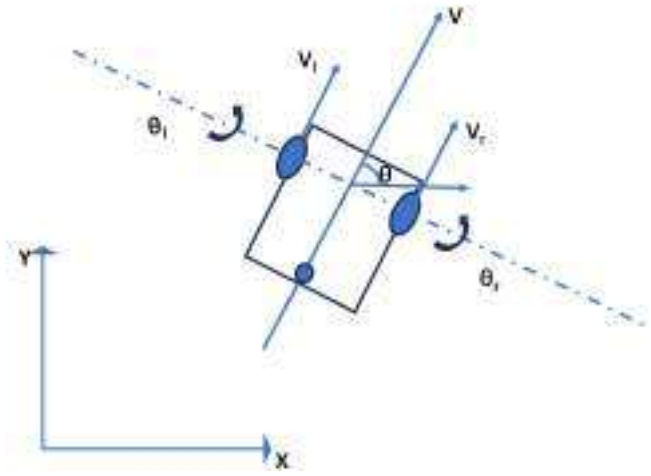


Fig. 4. Diiferrential Drive WMR

A differential drive wheeled mobile robot is a type of non-holonomic system, which means it has constraints on its motion that cannot be integrated into constraints on its position. This is due to the robot's wheels being directly linked to its control inputs, allowing for forward and backward movement and in-place rotation, but not lateral movement.There are two powered wheels and a third wheel for balancing.The difference of speed between these two wheels is responsible to make robot turn.Both wheel speeds are controlled independently.
Assumptions:
1. Point Q coincides with center of gravity of robot
2. There is no slip.
3. Velocity of right wheel is more than the left wheel
Parameters used:
V= Net linear velocity of the robot, V_L= Linear velocity of Left Wheel, V_R= Linear velocity of Right Wheel, $2a$ = Wheel

base, ϕ= Robot orientation, $\dot{\theta}_L$ = angular velocity of left wheel, $\dot{\theta}_R$ = angular velocity of right wheel, r= radius of wheel (both have equal radius), $\dot{\phi}$= Angular velocity of robot

$$V_r = V + a\dot{\phi}$$

$$V_r = V - a\dot{\phi}$$

By the addition of both

$$V = (V_r + V_l)/2$$

By the subtraction of both

$$\dot{\phi} = (V_r - V_l)/2a$$

Assuming no-slip condition

$$V_r = r\dot{\theta}_r$$

$$V_l = r\dot{\theta}_l$$

$$\dot{x} = Vcos\phi$$

$$\dot{y} = Vsin\phi$$

$$\dot{x} = (V_r + V_l)/2cos\phi$$

$$\dot{x} = (V_r + V_l)/2sin\phi$$

$$\dot{x} = r/2(\dot{\theta}_r + \dot{\theta}_l)/2cos\phi$$

$$\dot{y} = r/2(\dot{\theta}_r + \dot{\theta}_l)/2sin\phi$$

$$\dot{\phi} = \frac{r(\dot{\theta}_r - \dot{\theta}_l)}{2a}$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{bmatrix} = \begin{bmatrix}\frac{r}{2}(cos\phi) & \frac{r}{2}(cos\phi)\\ \frac{r}{2}(sin\phi) & \frac{r}{2}(sin\phi)\\ \frac{r}{2a} & -\frac{r}{2a}\end{bmatrix}\begin{bmatrix}\dot{\theta}_r\\ \dot{\theta}_l\end{bmatrix}$$

Non-holonomic Constraint

$$\dot{y}cos\phi - \dot{x}sin\phi = 0$$

B. Follower Formation Modelling

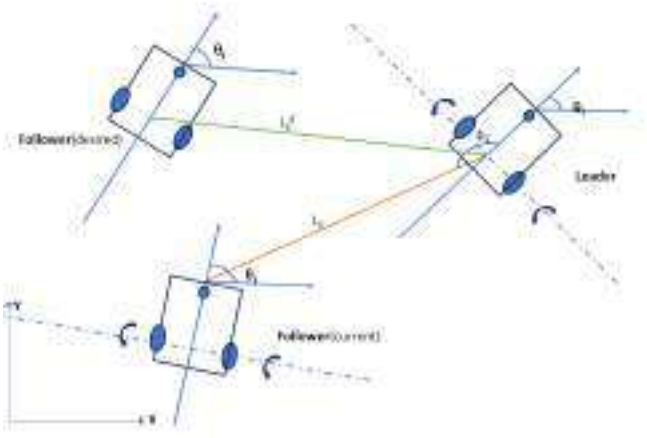


Fig. 5. Leader-Follower Formation

i represents leader vehicle
j represents follower vehicle
L_{ij} current distance between leader and follower
L_{ij}^d desired distance between leader and follower
θ_i orientation of leader vehicle
$\theta_j$orientation of follower vehicle
ϕ_{ij} represents relative angle between leader and follower

$$\begin{bmatrix}x_j\\ y_j\\ \phi_j\end{bmatrix} = \begin{bmatrix}x_i - dcos\theta_i + L_{ij}cos(\phi_{ij} + \theta_i)\\ y_i - dcos\theta_i + L_{ij}sin(\phi_{ij} + \theta_i\\ \theta_i + \phi_{ij}\end{bmatrix}$$

$$L_{ijx} = x_i - x_j - dcos\theta_i$$

$$L_{ijy} = y_i - y_j - dsin\theta_i$$

$$L_{ij} = \sqrt{L_{ijx}^2 + L_{ijy}^2}$$

As we know

$$\dot{x}_i = V_icos\theta_i$$

$$\dot{y}_i = V_isin\theta_i$$

$$\dot{\theta}_i = \omega_i$$

Differentiating

$$\dot{L}_{ijx} = \dot{x}_i - \dot{x}_j - dsin\theta_i\dot{\theta}_i$$

$$\dot{L}_{ijx} = V_icos\theta_i - V_jcos\theta_j + d\omega_isin\theta_i$$

$$\dot{L}_{ijy} = \dot{y}_i - \dot{y}_j - dcos\theta_i\dot{\theta}_i$$

$$\dot{L}_{ijy} = V_isin\theta_i - V_jsin\theta_j - d\omega_icos\theta_i$$

$$L_{ij} = \sqrt{L_{ijx}^2 + L_{ijy}^2}$$

Differentiating

$$\dot{L}_{ij} = \frac{1}{\sqrt{L_{ijx}^2 + L_{ijy}^2}}\left(L_{ijx}\dot{L}_{ijx} + L_{ijy}\dot{L}_{ijy}\right)$$

$$\dot{L}_{ij} = \frac{1}{L_{ij}}\left\{L_{ijx}\left(V_i\cos\theta_i - V_j\cos\theta_j + d\omega_i\sin\theta_i\right)\right. \quad (1)$$

$$\left. + \; L_{ijx}\left(V_i\cos\theta_i - V_j\cos\theta_j + d\omega_i\sin\theta_i\right)\right\} \quad (2)$$

$$\dot{L}_{ij} = \frac{1}{L_{ij}}\{V_i(L_{ijx}cos\theta_i + L_{ijy}sin\theta_i)\} +$$

$$(-)\frac{1}{L_{ij}}\{V_j(L_{ijx}cos\theta_j + L_{ijy}sin\theta_j)\} +$$

$$\frac{1}{L_{ij}}\{(-L_{ijx}sin\theta_i + L_{ijy}cos\theta_i) - d\omega_i\}$$

$$
\begin{aligned}
\dot{L}_{ij} = &\{V_i(-L_{ij}cos(\theta_i + \phi_i j)cos\theta_i\} + \\
&\{V_i(-L_{ij}sin(\theta_i + \phi_i j)sin\theta_i\} \\
&- \{V_j(-L_{ij}cos(\theta_i + \phi_i j)cos\theta_j\} - \\
&\{V_j(-L_{ij}sin(\theta_i + \phi_i j)sin\theta_j\} \\
&- d\omega_i \{L_{ij}cos(\theta_i + \phi_i j)sin\theta_j \\
&- L_{ij}sin(\theta_i + \phi_i j)cos\theta_j
\end{aligned}
$$

$$
\begin{aligned}
\dot{L}_{ij} = &-v_i\ cos(\theta_i + \phi_{ij})cos\theta_i + sin(\theta_i + \phi_{ij})sin\theta_i) \\
&+v_j\ cos(\theta_i + \phi_{ij})cos\theta_j + sin(\theta_i + \phi_{ij})sin\theta_j) \\
&-d\omega_i\ cos(\theta_i + \phi_{ij})sin\theta_j - sin(\theta_i + \phi_{ij})cos\theta_j)
\end{aligned}
$$

$$\theta_i + \phi_{ij} - \theta_j = \gamma_i j$$

$$\dot{L}_{ij} = -v_i cos\phi_i j + v_j cos\gamma_i j + d\omega_j sin_i j$$

$$\phi_{ij} = atan2\frac{L_{ijy}}{L_{ijx}} - \theta_i + \pi$$

III. FURTHER STRATEGY

First we will try to develop controller for one axis motion, then will develop for two axis motion. Means First we will focus on only position control assuming same orientation for leader and follower. Then We will work to develop complete controller considering both position and orientation controller.

REFERENCES

[1] "Leader-Follower Formation control of nonholonomic mobile robots," IEEE Conference Publication — IEEE Xplore, Oct. 13, 2020. https://ieeexplore.ieee.org/document/9272048

[2] ERC-ACI, Seoul National University, "The Mathematics of Formation Control:The 5th Wook Hyun Kwon Lecture," YouTube. Jan. 20, 2023. [Online]. Available: https://www.youtube.com/watch?v=C8GFFlWTnPw

[3] D. Chikurtev, M. Grueva, and Institute of Information and Communication Technologies – BAS, "Leader-follower formation control of multiple mobile robots," BULGARIAN ACADEMY OF SCIENCES, journal-article, 2018. [Online]. Available: https://www.iict.bas.bg/pecr/2018/70/2-PTKR-70-Chikurtev-Grueva.pdf

[4] Brian Douglas, "Controlling robotic swarms," YouTube. Dec. 03, 2014. [Online]. Available: https://www.youtube.com/watch?v=stzQNjtDg0g

[5] Xiangyu Meng, "Formation Control of Multi-Agent Systems Part 1 Formation Specification," YouTube. Sep. 23, 2020. [Online]. Available: https://www.youtube.com/watch?v=Z5hvAO8sQlI

Arashdeep Singh

Roll No.:M23IRM003
Autonomous Mobile Robots
Robotics and Mobility Systems
Indian Institute Of Technology, Jodhpur

+91-8146895443
arashdeepsingh1512@gmail.com
m23irm003@iitj.ac.in
GitHub
LinkedIn

Education

Degree/Certificate	Institute/Board	CGPA/Percentage	Year
M.Tech. (RMS)	Indian Institute of Technology, Jodhpur	8.52	2023–2025
B.Tech. (ME)	Guru Nanak Dev Engineering College, Ludhiana	7.58	2016–2020
Senior Secondary	PSEB Board	90.00%	2015
Secondary	PSEB Board	91.84%	2013

Experience

- **Junior Research Fellow** *May. 2025 – Current*
  IITJ
  - **Autonomous Delivery Robot for Indoor & Outdoor Environments** GitHub YouTube
    - Developing a ROS 2-based navigation system for the Husky A200 to autonomously move across mixed indoor–outdoor environments.
    - Implementing RTAB-Map with 2D LiDAR and RGB-D for real-time SLAM, loop closure, and robust mapping in varied conditions.
    - Adapting Nav2 planners/controllers for seamless navigation between narrow indoor corridors and open outdoor spaces.
    - **Tools & Technologies:** ROS 2 Humble, Nav2, RTAB-Map, `robot_localization`, Gazebo, C++, Python
  - **Autonomus Navigation for Industrial Warehouse using PM2 Base (Simulation)** GitHub YouTube
    - Simulated warehouse tasks with Nav2 (Smac Planner & DWB controller) using PAL Robotics base in Gazebo.
    - Mapped using Slam-toolbox, ran AMCL on saved maps; fused odometry + IMU for stable pose estimation.
    - **Tools & Technologies:** ROS 2 Humble, Nav2 , Slamtool box, Behavior Tree, `robot_localization`, Gazebo, C++
  - **Multi-Sensor Fusion for Localization** GitHub YouTube
    - Architected a reusable `robot_localization` pipeline fusing odometry, IMU, GPS, and LiDAR/VO for robust `/odometry/filtered`.
    - Tuned covariance matrices for indoor/outdoor modes.
    - Provided evaluation scripts (evo) for ATE/RPE analysis and comparative performance plots.
    - **Tools & Technologies:** ROS 2 Humble, `robot_localization` (EKF), navsat_transform, evo, RViz, Python

Research and Academic Projects

- **Leader–Follower Formation Control of Differential-Drive Robots** GitHub
  - Designed and simulated multi-robot formations using Lyapunov-based controllers in MATLAB.
  - **Skills:** Robotic kinematics, nonlinear control, Lyapunov stability
- **Control of Underactuated Marine Vehicles** GitHub
  - Developed cascaded feedback linearization and backstepping controllers for underactuated AUVs; validated stability via Lyapunov analysis.
  - **Skills:** Nonlinear control theory, feedback linearization, backstepping, ROS2
- **Reactive Obstacle Avoidance Methods** GitHub
  - Implemented Dynamic Modulation Matrix (DMM) and Rotational Obstacle Avoidance Method (RoAM) for real-time collision avoidance.
  - **Skills:** Reactive Obstacle Avoidance, ROS2 Jazzy, Gazebo
- **Multi-Agent Task Allocation** GitHub
  - Designed self-task allocation algorithm using capability assessment for cooperative robots.
  - **Skills:** Multi-agent systems, distributed algorithms, task allocation strategies, ROS2 Jazzy, Gazebo

Key Courses Taken

- Introduction to Robotics, Mobile Robots, Artificial Intelligence, Computer Vision, Autonomous Systems, Embedded System Design, Fundamentals of Machine Learning, Robot Operating System, Introduction to Medical Robotics

Technical Skills

- **Programming:** C, C++, Python, Matlab, Shell Scripting
- **Tools & OS:** Git, Jupyter Notebook, Google Colab, Linux, Windows, ROS, Docker, Solidworks
- **Libraries/Frameworks:** Pandas, Numpy, scikit-learn, Open-CV, Matplotlib, TensorFlow, PyTorch
- **Sensors & Hardware :** IMU, Depth Camera, Lidar, Load Cell, Common Motors, Micro-controllers, Basic Motor Control, Husky A200

Certifications

- MathWorks Certification on Matlab Fundamentals,Linear Algebra, ODE
- SWCAD1.0: SOLIDWORKS CAD Fundamentals
- CS50 Introduction to Programming Using Python Certification
- LinkedIn : Additive Manufacturing Optimizing 3D Prints
- LinkedIn: Numpy, Matplotlib, Pandas, Bash Scripting

Admittance Control for 2D Plotter

A Project Report Submitted by

Team Members:

Arashdeep Singh M23IRM003

Chaitanya Patil M23IRM004

Mayank Kumar M23IRM007

Muskan Suman M23IRM008

in partial fulfillment of the requirements for the course project of

Introduction To Medical Robotics

Indian Institute of Technology Jodhpur

Inter-disciplinary Research Division (IDRD)

April, 2025

1 Introduction

This project implements admittance control on a 2D plotter to achieve smooth, compliant motion in response to external forces. Initially, a joystick was used to control motor movement across the X and Y axes. Later, Force Sensing Resistors (FSRs) replaced the joystick to detect applied forces and drive motion accordingly. The system dynamics were modeled using Lagrangian mechanics, and a PID controller was designed to align the real system's behavior with the desired virtual dynamics. This work demonstrates basic principles of force-based control, highlighting its potential for applications requiring safe and adaptive interaction, such as in medical robotics.

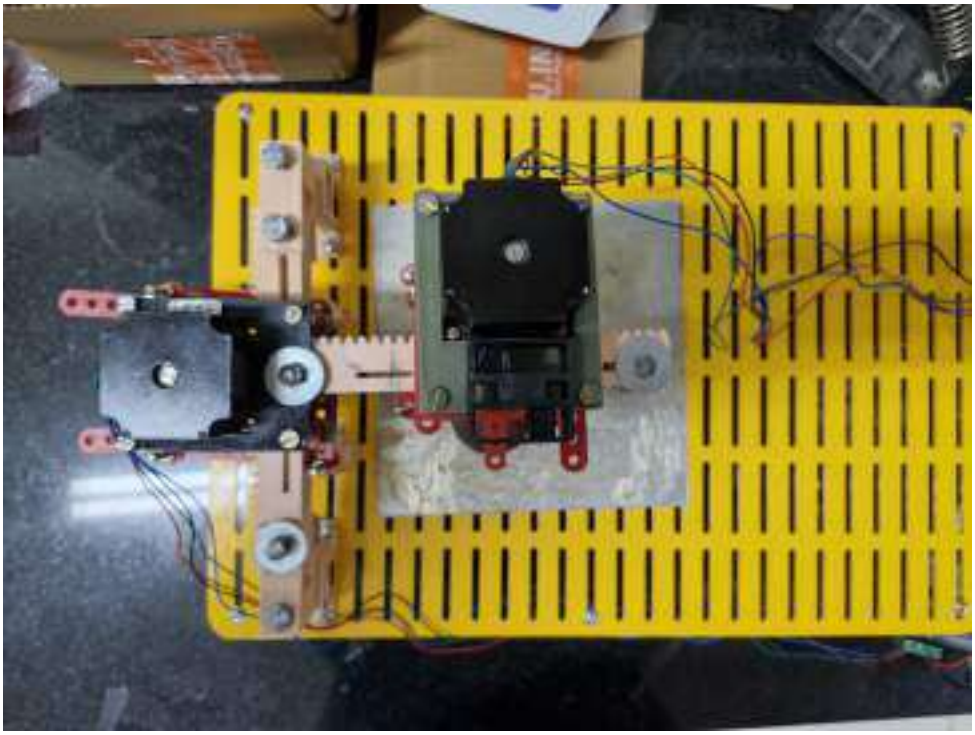


Figure 1.1: Top View of Assembly

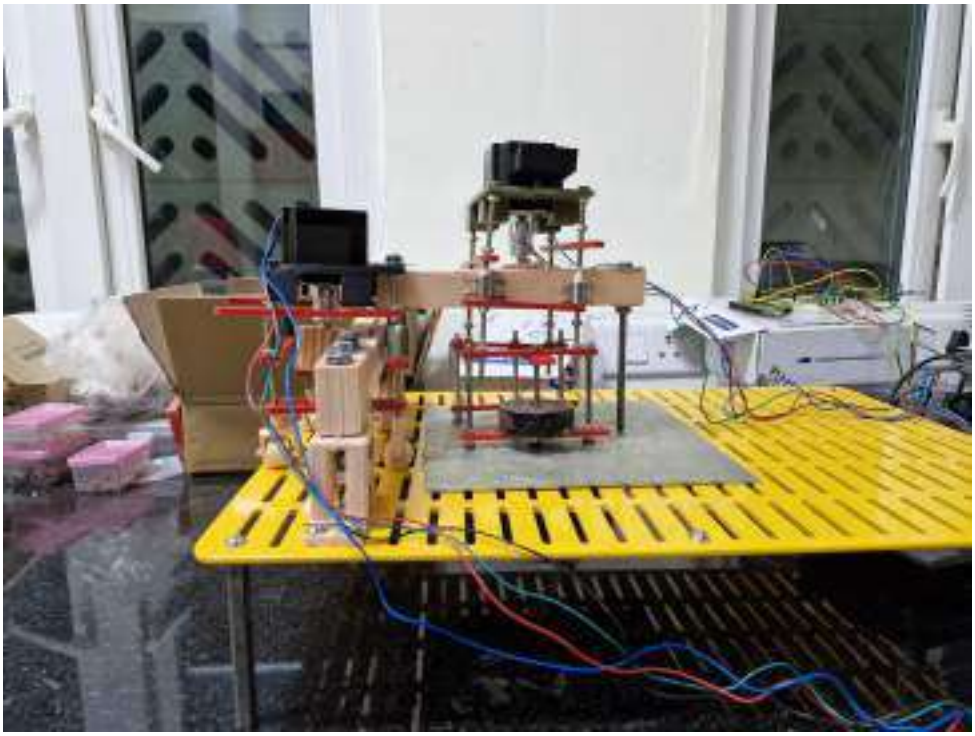


Figure 1.2: Side View of Assembly

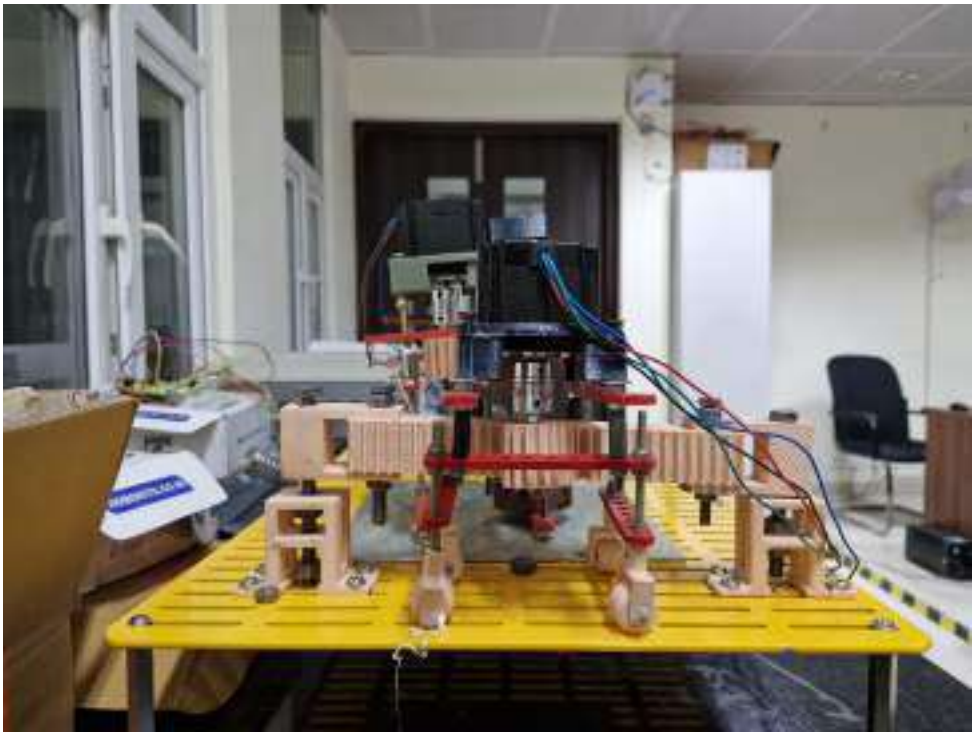


Figure 1.3: Back View of Assembly

1.1 Experimental Setup Description

The experimental setup for the 2D plotter designed to implement admittance control is shown in figurers. The platform facilitates planar motion along the X and Z axes and is constructed using a combination of 3D-printed components, wooden frames, and metallic rods. The structure is mounted on a perforated yellow acrylic base, providing both stability and flexibility for modular adjustments.

In our specific implementation, the 2D plotter serves as the robotic system whose end effector motion is governed by the admittance control strategy. The force sensor attached to the end effector measures external forces, which are processed through the virtual dynamics model to compute the reference velocity. The PID controller then determines the necessary control forces to enforce this velocity in the real system. By leveraging both the virtual model for reference generation and the real model for execution, we achieve a compliant and adaptive system capable of handling external force variations effectively.

To effectively implement admittance control, we employ a 2D prismatic system in a horizontal setup, offering multiple advantages. Operating in a horizontal plane minimizes the effects of gravitational potential energy, simplifying system modeling and ensuring force measurements remain unaffected by gravity. Unlike robotic arms with rotational joints, which require complex transformations, the prismatic system moves strictly in the X-Y plane, reducing system complexity and making control implementation more straightforward. The absence of rotational degrees of freedom results in linear and predictable motion, making the system more stable and well-suited for precision control applications. Additionally, prismatic motion directly maps applied forces to linear accelerations, enhancing control accuracy and force interpretation, which is particularly beneficial in force-based interactions. The primary components of the setup include:

- **Stepper Motors:** Two stepper motors control the motion along the X and Z directions. These motors are mounted on customized supports at different heights to enable independent axis actuation.
- **Force Sensing Resistors (FSRs):** FSRs are integrated at the end-effector to detect externally applied forces, enabling force-based motion control.
- **Couplings and Linkages:** Shaft couplings connect the motors to the moving frame, ensuring efficient torque transfer and mechanical compliance.
- **Electronics and Wiring:** Multiple color-coded wires connect the motors, sensors, and the microcontroller unit. These wires are organized to avoid interference with the moving parts.
- **Control System:** An Arduino microcontroller and motor drivers are employed to process the sensor inputs and implement the admittance control strategy.
- **Rack and Pinon**: Rack and Pinion system with pressure angle of 20 degree, module 2mm is printed on 3D printer with ABS as material.

The system is designed to operate in a horizontal plane, minimizing gravitational effects on force measurements and simplifying dynamic modeling. External forces applied on the end-effector are sensed by the FSRs, and the motion is regulated using a virtual mass-spring-damper model through admittance control.

Overall, the setup is optimized for simplicity, modularity, and functionality, providing an effective platform for studying compliant motion control strategies in robotic systems.

2 Equipment and Consumables Required

S. NO.	Product	Quantity
1	Stepper Motor + Driver	2
2	Force Sensor	4
3	Multicolored Wires	1 Bundle
4	Shaft Coupling	2
5	Joystick	1

(a) Stepper Motor (b) Coupling (c) Wires (d) Force Sensor (e) Driver

Figure 2.1: Assembly components of joystick setup

Figure 2.2: 2D Joystick

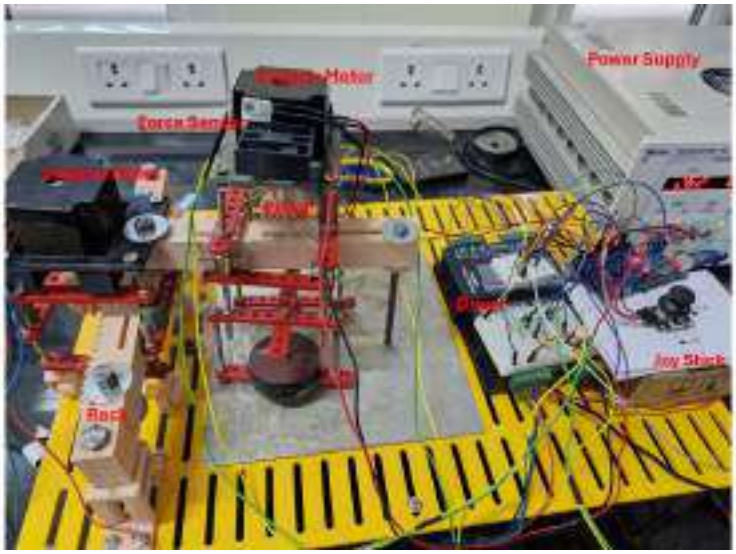


Figure 2.3: Conceptual Labeled Diagram

3 Implementation

In this project we have done implementation in three phases. In first phase we have motor control using joystick. Then in second phase will inlcude force sensor into the setup will use force values for motor control. In final phase will study system dynamics and will try to implement torque control for motor.

3.1 Motor Control Based on Joystick Movement

A dual-axis analog joystick provides two independent control signals via built-in potentiometers. Each potentiometer is linked to one axis (horizontal X or vertical Y) of the joystick thumbstick. When the stick is centered, both potentiometers output a mid-scale voltage (around half of the supply voltage, e.g., 2.5 V on a 5 V system). Moving the joystick changes the resistance of the potentiometers, causing the output voltages to increase or decrease linearly with the deflection. These voltages are typically fed into a microcontroller's analog-to-digital converter (ADC) channels (commonly labeled VRx and VRy).

The microcontroller continuously samples these two analog values to determine the joystick's position along each axis.For example, on a 10-bit ADC (0–1023 range), the neutral position reads approximately 512, pushing the stick right/up increases that reading toward 1023, and pushing left/down decreases it toward 0. This analog readout scheme allows precise and proportional control of the connected devices. The system is configured so that the joystick's horizontal (right/left) motion drives Motor 1 (the X-axis motor) and its vertical (up/down) motion drives Motor 2 (the Z-axis motor). The resulting mapping of joystick deflection to motor action is:

- **Horizontal Movement (Right/Left) – Controls Motor 1 (X-axis movement)**:
  - Joystick Right (X+): The X-axis voltage rises above mid-scale. The MCU drives Motor 1 clockwise, moving the plotter carriage along the +X direction.

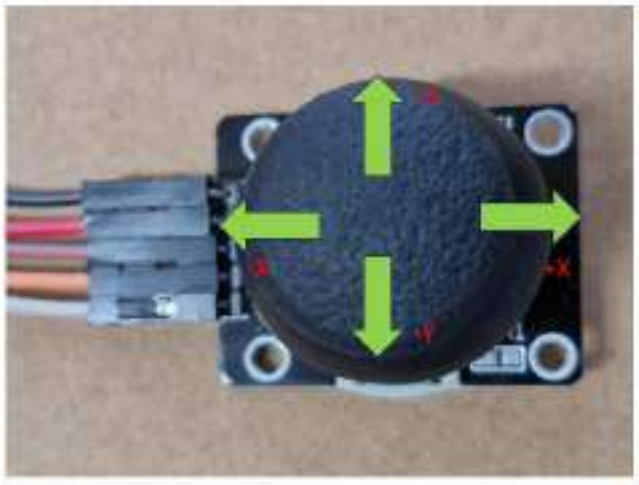


Figure 3.1: Motor control using joystick

  - Joystick Left (X–): The X-axis voltage falls below mid-scale. The MCU drives Motor 1 counterclockwise, moving the carriage along the -X direction.

- **Vertical Movement (Up/Down) – Controls Motor 2 (Z-axis movement)**:
  - Joystick Up (Y+): The Y-axis voltage rises above mid-scale. The MCU drives Motor 2 clockwise, moving the carriage along the +Z direction (upward).
  - Joystick Down (Y–): The Y-axis voltage falls below mid-scale. The MCU drives Motor 2 counterclockwise, moving the carriage along the -Z direction (downward).

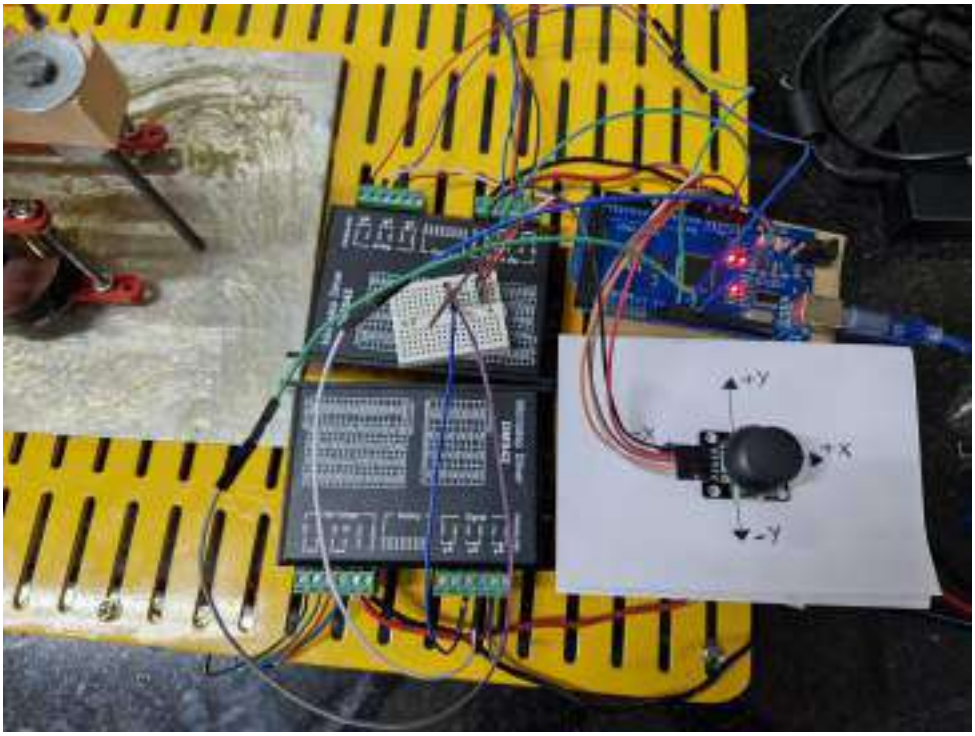


Figure 3.2: Assembly of joystick setup

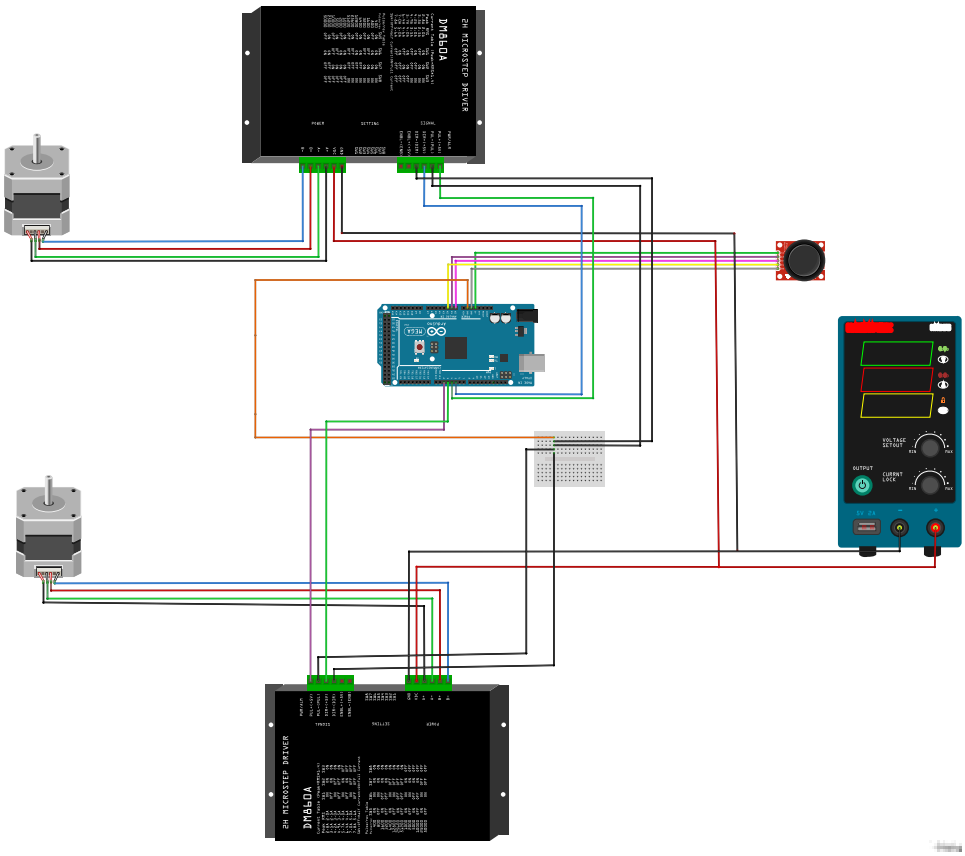


Figure 3.3: Circuit Diagram of Joy Stick Control

3.2 FSR-Based Force and Pressure Measurement for Motion Detection

In Phase 2 of the project, we integrate **Force Sensing Resistors (FSRs)** to measure applied forces and detect directional input for motion control. A total of **four FSR sensors** are used to sense forces in four directions — **right, left, up, and down** — replacing joystick input with force-based control.

Each FSR acts as a variable resistor, with high resistance ($>10\,\mathrm{M}\Omega$) when no force is applied, and decreasing resistance with applied force. It is used in a voltage divider with a fixed resistor, and the resulting voltage, which varies with force, is read by the Arduino's ADC. The ADC values range from 0 to 870 (out of 1023), and a force of 7 N corresponds to the maximum ADC reading.

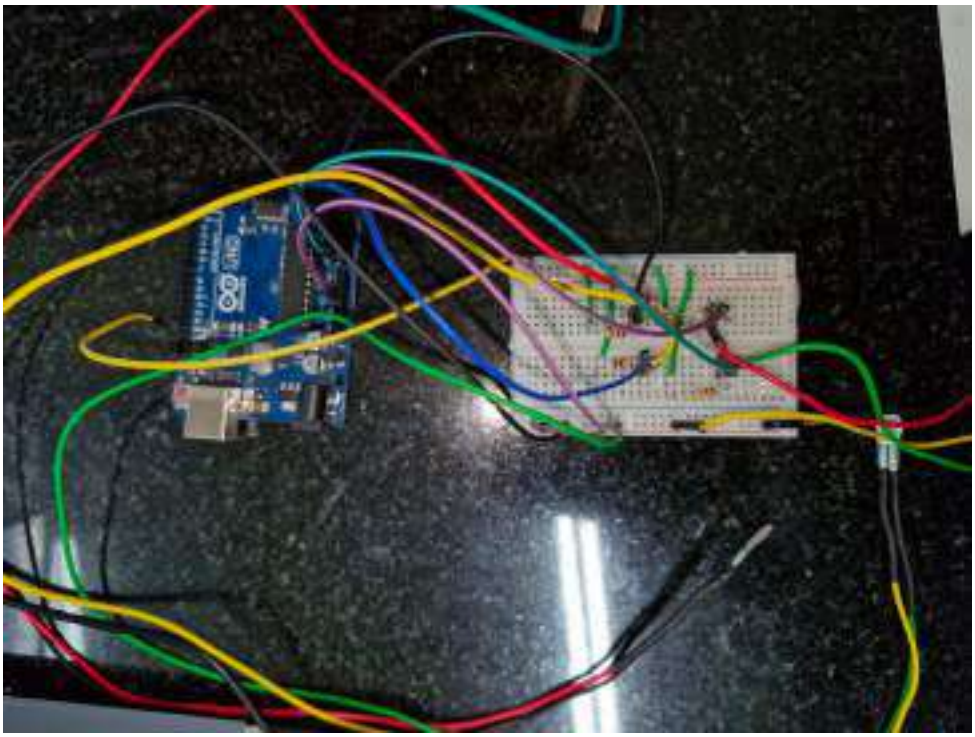


Figure 3.4: Force Sensing Resistors (FSRs)

3.2.1 Working Principle

- **Force Sensing Resistor (FSR):** An FSR is a sensor whose resistance decreases when a force or pressure is applied. Here Circuit for single Force Sensing Resistors

  The relationship between force and resistance is **inversely proportional**.

- **Basic Operation:**
  - As the applied force increases, the resistance of the FSR decreases.
  - A **voltage divider circuit** is used to convert the changing resistance into a measurable voltage signal.
  - This voltage is read by an Arduino microcontroller to estimate the applied force and pressure.

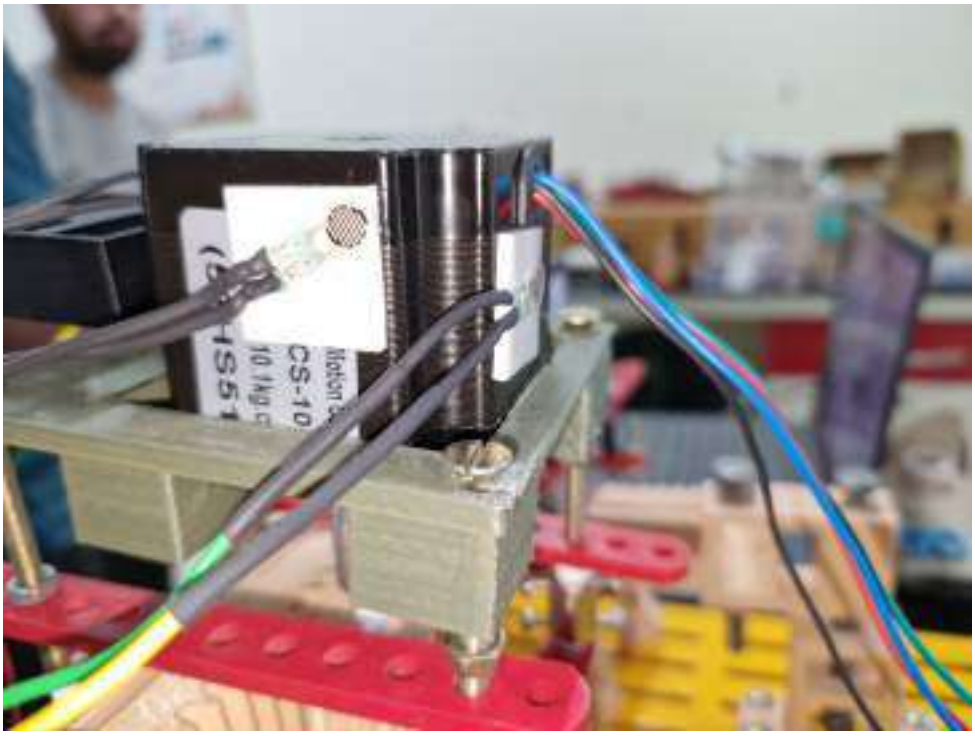


Figure 3.5: Force Sensor on End Effector

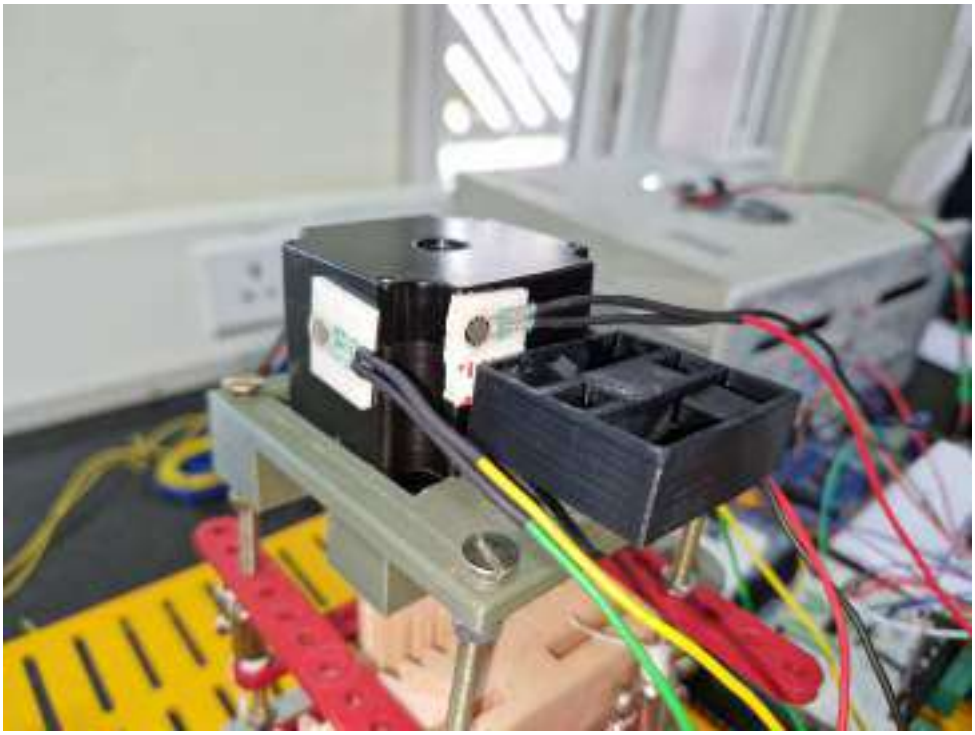


Figure 3.6: Force Sensor on End Effector

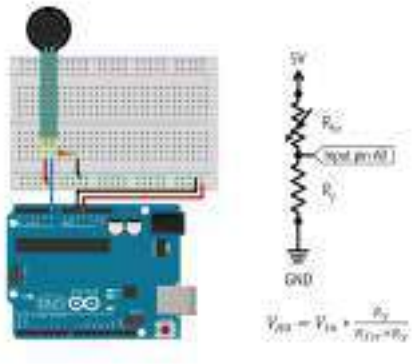


Figure 3.7: Force Sensing Resistors CIRCUIT

3.2.2 Circuit Configuration

Voltage Divider Circuit:

Each FSR is connected in series with a known fixed resistor (R_{fixed}) and powered by a constant voltage source (V_{CC}). The output voltage (V_{out}) across the FSR is given by:

$$V_{\text{out}} = V_{\text{CC}} \times \frac{R_{\text{FSR}}}{R_{\text{fixed}} + R_{\text{FSR}}} \tag{1}$$

Rearranging, the FSR resistance (R_{FSR}) is calculated by:

$$R_{\text{FSR}} = R_{\text{fixed}} \times \left(\frac{V_{\text{CC}} - V_{\text{out}}}{V_{\text{out}}} \right) \tag{2}$$

Analog-to-Voltage Conversion:

The Arduino's `analogRead()` function reads a value between 0 and 1023, corresponding to 0V to 5V. Voltage is calculated as:

$$V_{\text{out}} = \text{analogRead}(A0) \times \frac{V_{\text{CC}}}{1023} \tag{3}$$

3.2.3 Force and Pressure Calculation

Force Estimation:

The applied force (in grams) is determined using the hyperbolic relationship:

$$\text{Force (grams)} = \frac{K}{R_{\text{FSR}}} \tag{4}$$

where K is a calibration constant determined experimentally.

Force Conversion:

The force in grams is converted to Newtons:

$$\text{Force (N)} = \text{Force (grams)} \times 0.00980665 \tag{5}$$

Pressure Calculation:

Pressure is calculated as:

$$\text{Pressure (Pa)} = \frac{\text{Force (N)}}{\text{Area (m}^2\text{)}} \tag{6}$$

where the sensor's effective contact area is known.

3.2.4 Calibration Process

A calibration process is essential to determine the constant K accurately:

- Known weights are applied sequentially.
- The value of K is adjusted until the computed force values match the real force values.

3.2.5 Data Output

- The Arduino displays real-time measured values — analog reading, voltage, resistance, force, and pressure — on the **Serial Monitor**.
- Measurements are updated every **200 milliseconds**.

FSR-to-Motor Mapping

Each of the four FSR sensors corresponds to a direction of motion:

The resulting mapping of FSR activation to motor action is:

- **Horizontal Force (Right/Left) – Controls Motor 1 (X-axis movement)**:
  - Right FSR pressed: Motor 1 moves clockwise, moving the plotter carriage along the +X direction.
  - Left FSR pressed: Motor 1 moves counterclockwise, moving the carriage along the -X direction.
- **Vertical Force (Up/Down) – Controls Motor 2 (Z-axis movement)**:
  - Up FSR pressed: Motor 2 moves clockwise, moving the carriage along the +Z direction (upward).
  - Down FSR pressed: Motor 2 moves counterclockwise, moving the carriage along the -Z direction (downward).

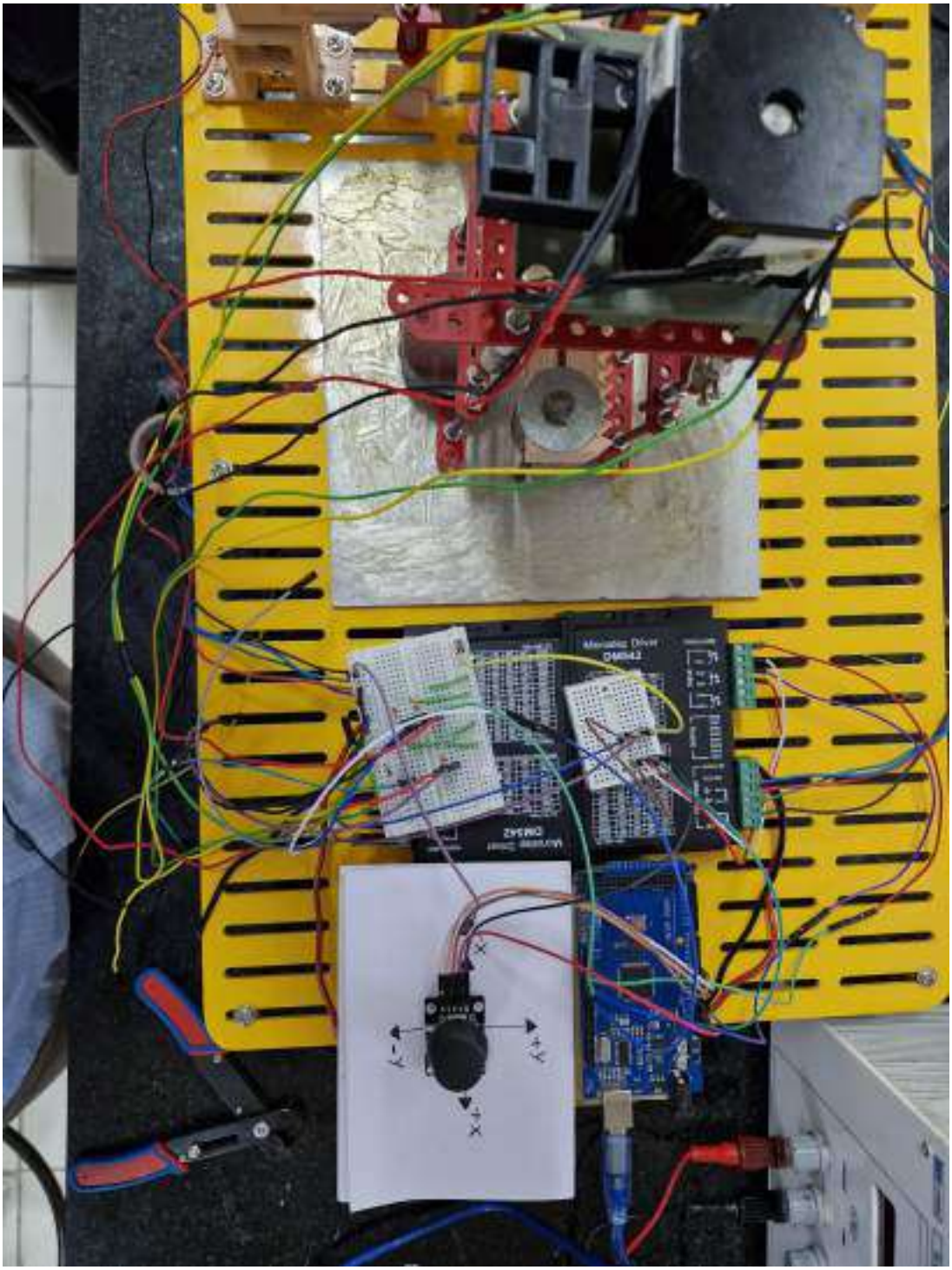


Figure 3.8: Assembly of joystick with force sensor setup

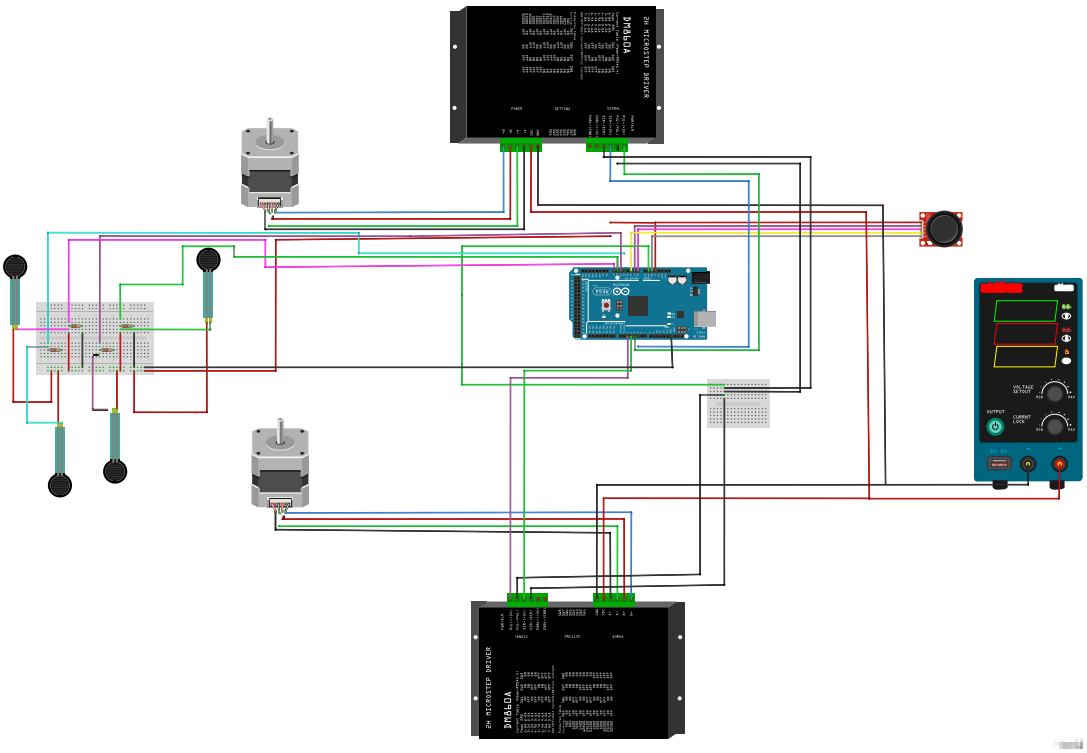


Figure 3.9: Circuit Diagram of Joy Stick and Force Control

4 Lagrangian Dynamics for the 2D Plotter

4.1 Introduction

This document presents the derivation of the Lagrange dynamics for a 2D horizontal plotter. The plotter typically has one axis that moves the entire setup, while the second axis motion affects only its own final position and motor mounting.

4.2 Coordinates

- x: The horizontal displacement of the setup, representing the absolute position in the horizontal plane.
- y: The relative position in the x direction.

4.3 Kinetic Energy

4.3.1 Translational Kinetic Energy

$$T_x = \frac{1}{2} M_x \dot{x}^2 \tag{7}$$

$$T_y = \frac{1}{2} M_y \dot{y}^2 \tag{8}$$

where M_x and M_y are the effective masses moving in the x and y directions, respectively.

4.3.2 Rotational Kinetic Energy

$$T_{\theta_1} = \frac{1}{2} I_1 \dot{\theta}_1^2 \tag{9}$$

$$T_{\theta_2} = \frac{1}{2} I_2 \dot{\theta}_2^2 \tag{10}$$

where I_1 and I_2 are the moments of inertia for the rotational components.

4.3.3 Total Kinetic Energy

$$T = \frac{1}{2}(M_x + I_1)\dot{x}^2 + \frac{1}{2}(M_y + I_2)\dot{y}^2 \tag{11}$$

4.4 Potential Energy

Assuming the mechanical design is stiff and the gravitational component is constant, the potential energy is negligible:

$$V = 0 \tag{12}$$

4.5 Lagrangian Mechanics

The Lagrangian L is given by:

$$L = T - V = \frac{1}{2}(M_x + I_1)\dot{x}^2 + \frac{1}{2}(M_y + I_2)\dot{y}^2 \tag{13}$$

4.6 Equations of Motion

Using the Euler-Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{14}$$

we derive the equations of motion for the system.

4.6.1 For x-axis

$$(M_x + I_1)\ddot{x} = F_{\text{ext},x} \tag{15}$$

4.6.2 For y-axis

$$(M_y + I_2)\ddot{y} = F_{\text{ext},y} \tag{16}$$

4.7 Admittance Control

Admittance control aims to impose a desired dynamic behavior on the system in response to external forces. This involves defining virtual dynamics and actual dynamics.

4.7.1 Virtual Dynamics

Virtual dynamics represent the desired behavior of the system. The virtual model is defined by:

$$M_v\ddot{x}_v + B_v\dot{x}_v + K_v x_v = F_{\text{ext}} \tag{17}$$

where M_v, B_v, and K_v are the virtual mass, damping, and stiffness, respectively, and x_v is the virtual position.

For Different values of dynamics coeffecients following behaviour of system dynamics is observed.

B	1
K	0.1
M	0.8

Table 4.1: Coefficents For Virtual Dynamics

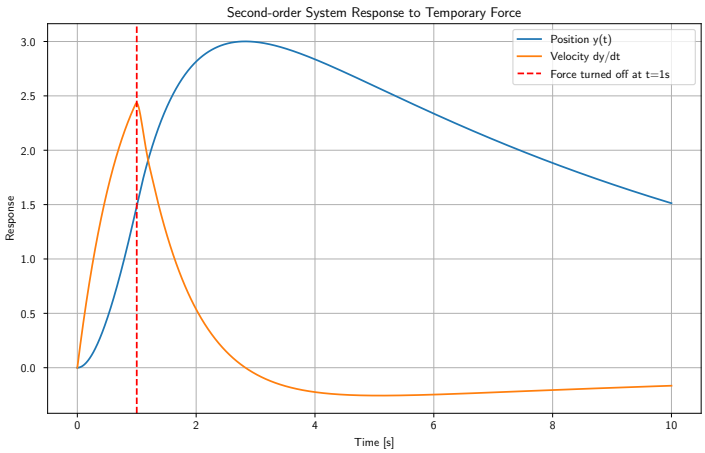


Figure 4.1: System Behaviour

B	1
K	0.1
M	0.1

Table 4.2: Coefficents For Virtual Dynamics

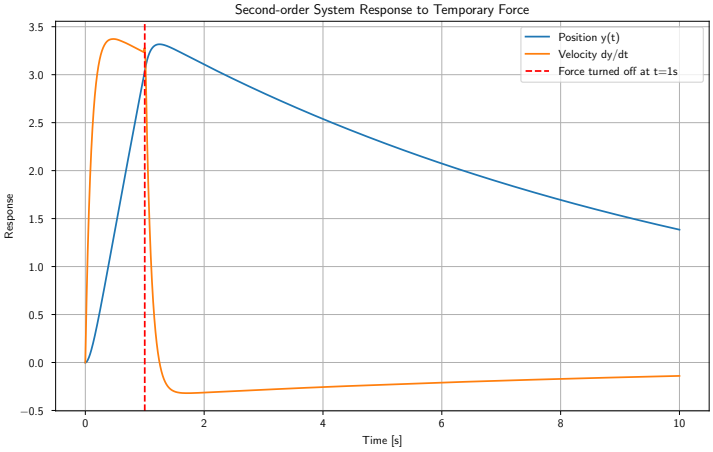


Figure 4.2: System Behaviour

B	2
K	0.1
M	0.1

Table 4.3: Coefficents For Virtual Dynamics

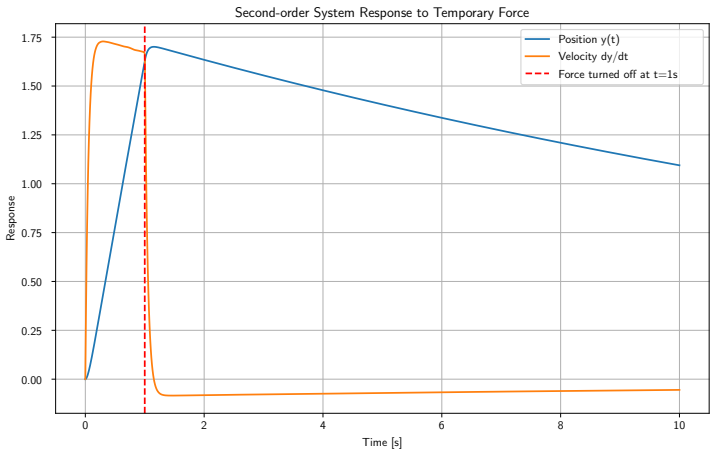


Figure 4.3: System Behaviour

B	2
K	0.8
M	0.1

Table 4.4: Coefficents For Virtual Dynamics

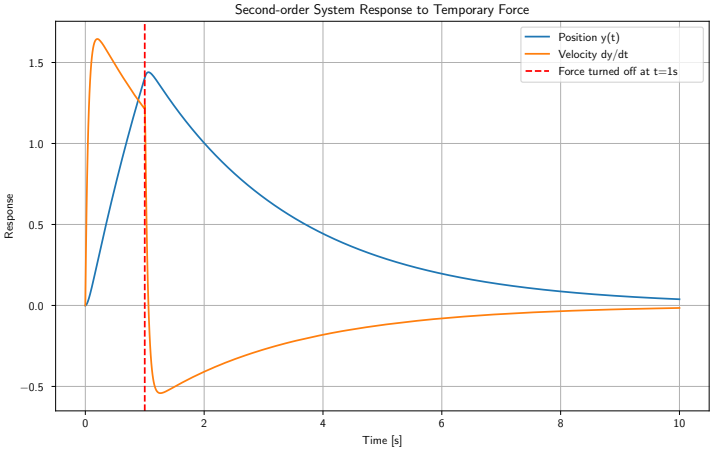


Figure 4.4: System Behaviour

4.8 Conclusion

The derived equations of motion describe the dynamics of the 2D horizontal plotter, accounting for both translational and rotational kinetic energies. The potential energy is considered negligible due to the stiffness of the mechanical design. Admittance control logic, incorporating virtual and actual dynamics, ensures that the system behaves as desired in response to external forces.

5 Working

In this project, we tried to implement admittance control on a 2D plotter setup, where the end effector is equipped with a force sensor to measure the applied external force. The primary objective is to regulate the system's response based on external force inputs, ensuring compliant motion while maintaining precise control over the end effector. The system consists of both virtual and real mathematical models, with a PID controller responsible for computing the control input to drive the real system.

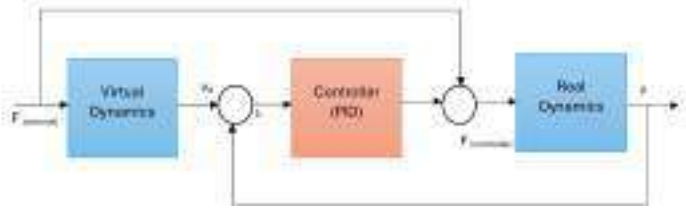


Figure 5.1: Basic admittance control diagram.

The overall structure of the admittance control system is illustrated in Fig. 1, which represents a basic stand-alone admittance control framework for an uncoupled robotic system. The control architecture consists of the following key components:

1. **Virtual Dynamics:** The external force applied by the user or environment is first processed by the virtual dynamics block, which determines the desired velocity reference v_d. This step ensures that the system behaves according to a predefined dynamic model, typically defined by mass-damping relationships, allowing the system to respond smoothly to applied forces.

2. **Error Computation:** The desired velocity is compared with the actual velocity v of the system to compute the velocity tracking error e. This error serves as the input to the controller, ensuring that the system continuously adjusts to match the desired motion profile.

3. **PID Controller:** A PID controller processes the velocity error e and generates a control force F_{control} to enforce the desired motion on the physical system. The PID controller optimizes this force to minimize deviations from the expected trajectory by adjusting the proportional, integral, and derivative terms based on system feedback.

4. **Robot Dynamics:** The real system receives the control force F_{control} from the actuator while also being influenced by the externally applied force F_{ext}. The interaction of these forces determines the actual velocity v of the system, which is then fed back into the error computation loop to maintain precise control.

5. **Feedback Loop and System Stability:** The feedback mechanism ensures that the system continuously refines its control inputs, allowing it to adapt to changes in applied forces while maintaining smooth motion. The admittance control approach enables the robot to dynamically adjust to variations in external forces, making it highly suitable for medical applications that require controlled and adaptive interactions with the environment.

6 Conclusion

In this project, we aimed to implement admittance control on a 2D plotter setup by modeling system dynamics and designing a control strategy based on external force measurements. Although we successfully developed the experimental setup, calibrated the force sensors, and designed a basic control loop using a virtual mass-spring-damper model, we were ultimately unable to implement full admittance control. The main limitation was our inability to achieve direct torque control of the stepper motors, which restricted us from realizing the intended compliant behavior based on external forces. Despite this, the project provided valuable insights into force-based motion control, sensor integration, and system modeling, laying a strong foundation for future work involving torque-controlled actuators.

Comparison of Cascaded Feedback Linearization with PID Control Law with PID for Underactuated AUV

***Abstract*— This paper focuses on the control of Underactuated autonomous Underwater vehicles (AUVs). These vehicles inherently have non-linear dynamics and limited actuation, which present significant challenges. We have proposed a cascaded controller comprising an inner loop feedback linearization with an outer loop PID. The proposed method is compared against a conventional PID controller to evaluate its performance. Simulations are carried out using the ROS2 middleware and the Gazebo simulation environment. The performance of the controller is assessed under various scenarios, including trajectory tracking accuracy, robustness to initial condition variations.**

I. INTRODUCTION

Robotics is becoming part of our daily life, industry, and defense. Robots are being used for the exploration of new areas and for protecting geographically tough areas. One such area is Underwater. Both for exploration and security purposes. Underwater Vehicles, such as submarine,s have been used for the past few decades. But with advancements, the focus is on autonomous underwater vehicles. For achieving autonomy, one major aspect is control of the AUV. The higher DoF compared to a ground vehicle makes it challenging. Apart from making their operating time longer, usually they are underactuated, which makes their control more complex. We have studied the literature related to this problem. We have found that Bashir et al. [1] reviewed control algorithms including model predictive control, adaptive control, H control, fuzzy logic, PID, and backstepping for trajectory control of unmanned underwater vehicles.

Makam et al. [2] studied the mathematical modelling of underwater vehicles in detail, along with controllers. Specifically, they discussed the depth pitch controller, the Yaw Controller for the linearized model. In nonlinear control, they had compared the SMC with conventional controllers like PID. Steenson et al. [3] focused on model predictive control for depth regulation of AUVs. In particular they have used Delphin2 AUV which is an overactuated thrusters and control surfaces.

Vadapalli et al. [4] focused on 3D path following of AUV using a backstepping approach for robust control. They have used Linear Matrix Inequality for controller design. They have not discussed about feedback linearization this paper. Esfahani et al. [5] worked on improving the tracking accuracy for motion control of AUVs under external changes.To make a robust controller they have used Backstepping,Integral Sliding Mode Control and time delay control. Zhou et al. [6] focused on 5 DoF motion model with input delays. They focused on underactuated vehicles with time delays and proposed a controller based on sliding mode control.

Vu et al. [7] focused on adaptive controller based on Sliding Mode for underactuated AUVs. They focused on bringing robust control and adaptive learning together. They proposed a two level hierarchical controller.

Lainag X .[8] addressed the path following of an underactuated AUV with external parameters. They proposed an adaptive robust control system using fuzzy logic,backstepping an sliding mode control.

There has been work on the control of underwater vehicles. Feedback linearization also has been employed, but not with respect to underactuated underwater vehicles, which poses new challenges. We will address these new challenges and will design a cascaded controller using feedback lineraization in outer loop and PID in inner loop. Will try to control major DoF like pitch, yaw, and surge under various scenarios. It demonstrates a clear control challenge (nonlinearity, underactuation, coupling via fins, dependence on surge velocity), and a structured control solution. While nonlinear and adaptive controllers have been widely studied, feedback linearization has received comparatively less attention in the context of underactuated AUVs, especially those actuated via surge-dependent fins. Most works assume fully actuated vehicles or neglect the coupling effects introduced by fin-based actuation.

II. MATHEMATICAL MODELLING AND CONTROLLER DESIGN

A. Mathematical Modelling of Underwater Vehicle

Following Fossen 's book the following kinematics and Dynamics have been used:- Following the Manuvering Theory Given by T. I. Fossen [9], We have worked on kinematics and dynamics of surface and underwater vehicles.

Kinematics of Six DOF Underwater Vehicle:- It describes how the vehicle's position (x, y, z) and orientation (roll ϕ, pitch θ, yaw φ) change over time based on its control inputs like linear velocities (u, v, w) and angular velocities (p, q, r). Essentially, it captures the complex motion dynamics of the vehicle as it navigates underwater.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} c(\varphi)c(\theta) & -s(\phi)c(\phi)+c(\varphi)s(\theta)s(\phi) & s(\varphi)s(\phi)+c(\varphi)c(\phi)s(\theta) & 0 & 0 & 0 \\ s(\varphi)c(\theta) & c(\varphi)c(\phi)+s(\phi)s(\theta)s(\varphi) & -c(\varphi)s(\phi)+s(\theta)s(\varphi)c(\phi) & 0 & 0 & 0 \\ -s(\theta) & c(\theta)s(\varphi) & c(\theta)c(\phi) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & 0 & 0 & 0 & c(\phi) & -s(\phi) \\ 0 & 0 & 0 & 0 & s(\phi)se(\theta) & c(\phi)se(\theta) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix}$$

As per craig's model:-

$$\dot{\eta} = J(\eta)\nu$$

B. Three DOF Dynamic Model

The kinetic equations of motion for an Autonomous Underwater Vehicle (AUV) in 3 Degrees of Freedom (DOF) are given by:

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau$$

Where:

- M is the inertia matrix (including added mass),
- $\dot{\nu}$ is the time derivative of the velocity vector (linear and angular accelerations),
- $C(\nu)$ is the Coriolis and centripetal matrix (depends on velocity ν),
- $D(\nu)$ is the damping matrix (due to hydrodynamic drag),
- $g(\eta)$ is the vector of gravitational and buoyant forces and moments (depends on the position and orientation vector η),
- τ is the control input (forces and moments from thrusters or control surfaces),
- ν is the velocity vector, typically:

$$\nu = \begin{bmatrix} u \\ w \\ r \end{bmatrix}$$

  where u is the surge velocity, w is the heave velocity, and r is the yaw rate,
- η is the position and orientation vector, typically:

$$\eta = \begin{bmatrix} x \\ z \\ \varphi \end{bmatrix}$$

  where x is the position in surge, z is the position in heave, and φ is the yaw angle.

Inertia Matrix (M):

$$M = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{33} \end{bmatrix}$$

where m_{11}, m_{22}, m_{33} are the mass and added mass terms in surge, heave, and yaw, respectively.

Coriolis and Centripetal Matrix ($C(\nu)$):

$$C(\nu) = \begin{bmatrix} 0 & 0 & -m_{22}w \\ 0 & 0 & m_{11}u \\ m_{22}w & -m_{11}u & 0 \end{bmatrix}$$

Damping Matrix ($D(\nu)$):

$$D(\nu) = \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix}$$

where d_{11}, d_{22}, d_{33} are the damping coefficients in surge, heave, and yaw.

Restoring Forces and Moments ($g(\eta)$):

$$g(\eta) = \begin{bmatrix} 0 \\ -(W - B) \\ 0 \end{bmatrix}$$

where W is the weight of the vehicle, B is the buoyancy force.

C. PID Controller

The following PID control laws are designed, where:
- Surge motion is directly actuated by a thruster.
- Pitch and yaw are controlled by fins that produce hydrodynamic moments only when there is sufficient forward (surge) velocity.

Surge Control (Thruster)

$$e_u(t) = u_d(t) - u(t) \tag{1}$$

$$T = K_{P,u}\, e_u(t) + K_{I,u} \int_0^t e_u(\tau)\, d\tau + K_{D,u}\, \frac{de_u(t)}{dt} \tag{2}$$

where T is the control thrust applied, and $K_{P,u}, K_{I,u}, K_{D,u}$ are the PID gains for surge.

Pitch Control (Fin, Surge-Dependent)

$$e_\theta(t) = \theta_d(t) - \theta(t) \tag{3}$$

$$\tau_\theta^{des} = K_{P,\theta}\, e_\theta(t) + K_{I,\theta} \int_0^t e_\theta(\tau)\, d\tau + K_{D,\theta}\, \frac{de_\theta(t)}{dt} \tag{4}$$

$$\delta_p = \frac{\tau_\theta^{des}}{k_\theta u^2 + \epsilon} \tag{5}$$

Here, τ_θ^{des} is the desired pitch moment, and δ_p is the pitch fin deflection. The term $k_\theta u^2$ captures the nonlinear relationship between surge speed and control effectiveness, and ϵ is a small constant to prevent division by zero.

Yaw Control (Fin, Surge-Dependent)

$$e_\psi(t) = \psi_d(t) - \psi(t) \tag{6}$$

$$\tau_\psi^{des} = K_{P,\psi}\, e_\psi(t) + K_{I,\psi} \int_0^t e_\psi(\tau)\, d\tau + K_{D,\psi}\, \frac{de_\psi(t)}{dt} \tag{7}$$

$$\delta_y = \frac{\tau_\psi^{des}}{k_\psi u^2 + \epsilon} \tag{8}$$

δ_y is the yaw fin deflection, dependent on the available surge velocity u.

Notes on Interdependence

The effectiveness of the pitch and yaw controllers is directly influenced by the surge velocity u, making it essential to maintain a minimum forward speed for adequate control authority. This interdependence is a hallmark of underactuated AUV control dynamics.

D. Cascaded Controller

Cascaded control offers a structured way to manage challenges of underactuation by decomposing the control problem into two nested loops. The *outer loop* regulates the vehicle's position by comparing the desired position p_d to the measured position p, and uses a PID regulator to generate a commanded velocity v_d. The *inner loop* then regulates velocity: it compares v_d to the measured velocity v and uses a second PID to produce a desired acceleration a_d. This separation simplifies tuning—each loop handles one integration order of the vehicle's kinematics—and improves robustness by limiting the bandwidth of each controller.

To handle the AUV's nonlinear dynamics in the inner loop, we employ *feedback linearization*. Given the standard rigid-body model

$$M(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + D(q,\dot{q})\,\dot{q} + g(q) \;=\; \tau \qquad (9)$$

where q is the configuration vector, M the inertia matrix, C the Coriolis/centripetal terms, D the hydrodynamic damping, and g the restoring forces, we choose

$$\tau \;=\; M(q)\,a_d \;+\; C(q,\dot{q})\,\dot{q} \;+\; D(q,\dot{q})\,\dot{q} \;+\; g(q) \qquad (10)$$

This exact inversion cancels all nonlinearities, yielding the linear input–output relationship $\ddot{q} = a_d$. The inner PID then effectively controls a double-integrator, vastly simplifying stability and performance analysis.

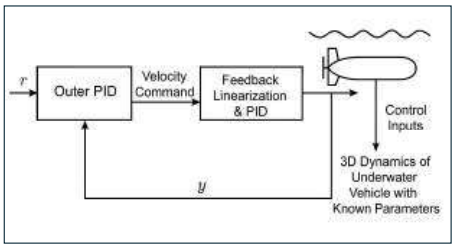


Fig. 1: AUV State Control Flow-chart

In short, the cascaded PID structure decouples position and velocity regulation, while feedback linearization cancels the AUV's nonlinear terms. The result is a controller that is both easy to tune and capable of high-performance trajectory tracking, even in the presence of strong hydrodynamic effects.

E. Controller Design

AUV Parameters Physical Properties

- Mass of AUV: $m = 148.3571$ kg
- Neutral buoyancy assumed: $W = B$
- Propeller diameter: $d_p = 0.2$ m

State Variables

- u: Linear velocity in surge (x-axis)
- q: Angular velocity in pitch (y-axis)
- r: Angular velocity in yaw (z-axis)
- θ: Pitch angle
- ψ: Yaw angle

Dynamic Equations (3 DOF)

TABLE I: AUV Parameters

AUV Parameters	
Inertia	
I_{xx}	$3.0000\ \text{kg}\cdot\text{m}^2$
I_{yy}	$41.980233\ \text{kg}\cdot\text{m}^2$
I_{zz}	$41.980233\ \text{kg}\cdot\text{m}^2$
Added Mass	
$X_{\dot{u}}$	-4.876161
$M_{\dot{q}}$	-33.46
$N_{\dot{r}}$	-33.46
Hydrodynamic Drag	
$X_{\lvert u\rvert u}$	-6.2282
$M_{\lvert q\rvert q}$	-632.698957
$N_{\lvert r\rvert r}$	-632.698957

Let the total surge, pitch, and yaw inertias be:

$$m_u = m - X_{\dot{u}} = 153.2333\ \text{kg} \qquad (11)$$

$$I^*_{yy} = I_{yy} - M_{\dot{q}} = 75.440233\ \text{kg}\cdot\text{m}^2 \qquad (12)$$

$$I^*_{zz} = I_{zz} - N_{\dot{r}} = 75.440233\ \text{kg}\cdot\text{m}^2 \qquad (13)$$

Surge

The decoupled equation in surge

$$m_u\dot{u} = -m_u ur + X_{|u|u}|u|u + \tau_u \qquad (14)$$

Where

m : Mass of the vehicle
$X_{\dot{u}}$: Added mass in surge direction
X_u : Linear damping coefficient in surge
$X_{|u|}$: Quadratic damping coefficient in surge
$|u|$: Magnitude of Surge Velocity
τ_u : Thrust along x-axis (from main propeller)

Pitch

$$(I_y - M_{\dot{q}})\dot{q} + M_q q + M_{|q|}|q|q = \tau_q \qquad (15)$$

I_y : Moment of inertia around y-axis (pitch)
$M_{\dot{q}}$: Added moment of inertia in pitch
$q = \dot{\theta}$: Pitch rate (time derivative of pitch angle)
M_q : Linear damping coefficient in pitch
$M_{|q|}$: Quadratic damping coefficient in pitch
τ_q : Pitching moment (from horizontal fins)

Yaw

$$(I_z - N_{\dot{r}})\dot{r} + N_r r + N_{|r|}|r|r = \tau_r \qquad (16)$$

Where align* I_z : Moment of inertia around z-axis (yaw)
$N_{\dot{r}}$: Added moment of inertia in yaw
N_r : Linear damping coefficient in yaw
$N_{|r|}$: Quadratic damping coefficient in yaw
τ_r : Yaw moment (from rudders/fins)

Under neutral buoyancy and symmetric design, restoring moments are negligible.

1) Feedback Linearization: **Surge (u)**

$$f_u(u,r) = -m_u ur + X_{|u|u}|u|u \tag{17}$$

Writing Surge Dynamics in standard non-linear form

$$\dot{u} = \frac{1}{m_u}\left(\tau_u + f_u(u,r)\right) \tag{18}$$

Define virtual control v_u:

$$\dot{u} = v_u \quad \Rightarrow \quad \tau_u = m_u v_u - f_u(u,r)$$

$$\tau_u = m_u v_u + m_u ur - X_{|u|u}|u|u \tag{19}$$

Pitch (q)

$$f_q(q) = M_q q + M_{|q|q}|q|q \tag{20}$$

Writing Pitch Dynamics in standard non-linear form

$$\dot{q} = \frac{1}{I^*_{yy}}\left(\tau_q - f_q(q)\right) \tag{21}$$

Define virtual control v_q:

$$\dot{q} = v_q \quad \Rightarrow \quad \tau_{pitch} = I^*_{yy} v_q - f_q(q) \tag{22}$$

$$\tau_q = I^*_{yy} v_q + M_q q + M_{|q|q}|q|q \tag{23}$$

Yaw (r)

$$f_r(r) = N_r r + N_{|r|r}|r|r \tag{24}$$

$$\dot{r} = \frac{1}{I^*_{zz}}\left(\tau_r - f_r(r)\right) \tag{25}$$

Define virtual control v_r:

$$\dot{r} = v_r \quad \Rightarrow \quad \tau_r = I^*_{zz} v_r + f_r(r) \tag{26}$$

$$\tau_r = I^*_{zz} v_r + N_r r + N_{|r|r}|r|r \tag{27}$$

PID Controller on Virtual Inputs

Once the nonlinearities are canceled, the system behaves like a set of integrators, where the virtual input directly controls the rate of change of the state. At this point, our goal becomes: Control the system states by properly shaping their rate of change — i.e., the virtual inputs. To achieve this, we design a PID controller on the virtual inputs. The PID controller computes the appropriate virtual input based on the tracking error (e.g., $(u_d - u)$ to ensure that the actual state converges to the desired value smoothly and accurately.

Surge PID

$$v_u = K_{p_u}(u_d - u) + K_{i_u}\int (u_d - u)\,dt + K_{d_u}(\dot{u}_d - \dot{u}) \tag{28}$$

Pitch PID

$$v_q = K_{p_q}(q_d - q) + K_{i_q}\int (q_d - q)\,dt + K_{d_q}(\dot{q}_d - \dot{q}) \tag{29}$$

Yaw PID

$$v_r = K_{p_r}(r_d - r) + K_{i_r}\int (r_d - r)\,dt + K_{d_r}(\dot{r}_d - \dot{r}) \tag{30}$$

Outer-Loop PID Controller (for Position Control)

Once the system is linearized using feedback linearization and the inner-loop PID controllers are applied to track the desired velocities, we can further extend the control architecture to perform **position tracking**. Since velocity is the time derivative of position, we treat the *velocity reference* (e.g., u_d, q_d, r_d) as a virtual input to a **second (outer) PID controller** that operates on **position error**.

This forms a *cascaded control structure*, where the outer-loop PID controller generates the *desired velocity* based on position tracking error, and the inner-loop PID controller ensures that the system follows that desired velocity accurately.

The outer-loop PID controller computes the velocity command by minimizing the position error over time. This allows the system states (e.g., surge position, pitch angle, yaw angle) to *converge to their respective desired values smoothly and with minimal steady-state error*.

Position PID Controllers: **Surge Position PID :**

$$u_d = K_{px}(x_d - x) + K_{ix}\int (x_d - x)\,dt + K_{dx}(\dot{x}_d - \dot{x}) \tag{31}$$

Pitch Angle PID (θ control):

$$q_d = K_{p\theta}(\theta_d - \theta) + K_{i\theta}\int (\theta_d - \theta)\,dt + K_{d\theta}(\dot{\theta}_d - \dot{\theta}) \tag{32}$$

Yaw Angle PID (ψ control):

$$r_d = K_{p\psi}(\psi_d - \psi) + K_{i\psi}\int (\psi_d - \psi)\,dt + K_{d\psi}(\dot{\psi}_d - \dot{\psi}) \tag{33}$$

Each outer-loop PID controller ensures that the vehicle moves toward its *position or orientation setpoint*. The output of this PID (desired velocity) becomes the *setpoint* for the *inner-loop controller*, which then commands the system accordingly.

Notes

- Restoring moments are neglected due to neutral buoyancy and symmetric configuration.
- PID gains (K_p, K_i, K_d) should be tuned based on desired tracking performance.

III. Simulation

A. Environment Setup

In this section, we describe the simulation environment that integrates Gazebo with ROS2 to implement a multi-agent framework with obstacle avoidance for marine robots.

a) Justification for Using Gazebo:: Gazebo is chosen as the simulation platform due to its realistic physics, robust ROS2 integration, and extensive sensor support. Its ability to simulate complex environmental effects—such as waves, currents, and drag forces—provides a highly realistic testing ground for evaluating the performance of our autonomous systems.

b) Simulation World:: The simulation world is designed to replicate an underwater and surface environment with carefully modeled physical properties.This includes factors like fluid dynamics and environmental disturbances to mimic real-world underwater conditions.I

TABLE II: Simulation Parameters

Parameter	Value/Description
Water Density	1025 kg/m³
Drag Coefficient	0.8
Gravity	9.81 m/s²
Gazebo Version	Gazebo Harmonic
ROS2 Version	ROS 2 Jazzy

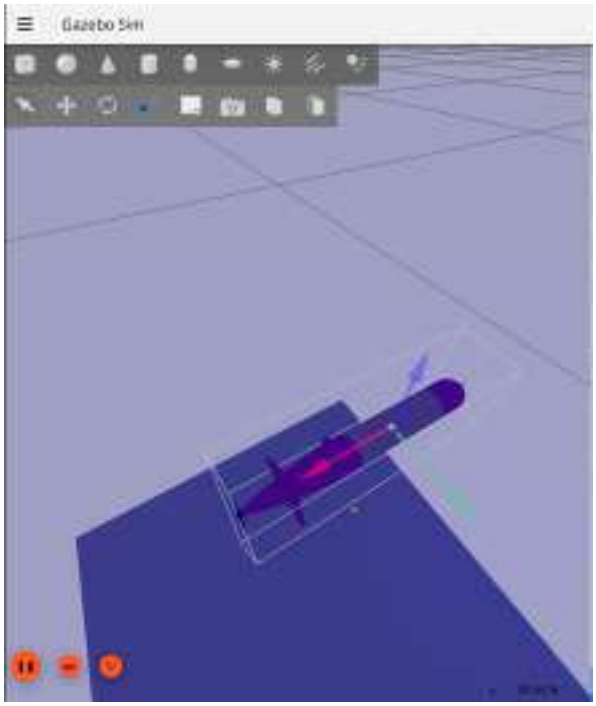


Fig. 2: Gazebo Sim Underwater World With AUV

c) Robot Model:: The robot models, such as AUVs and ASVs, are constructed using SDF/URDF. These models incorporate detailed representations of physical dimensions, mass properties, and sensor placements, ensuring accurate simulations of robot dynamics and interactions.[10]

d) Physics Engine:: For physics simulation, we employ the Open Dynamics Engine (ODE), which provides realistic dynamics and collision handling essential for underwater simulations.

e) Sensor Models:: The environment includes multiple sensor models (IMU, camera, LiDAR,GPS and Compass) integrated via ROS2 topics. This setup ensures comprehensive perception data for tasks such as navigation and obstacle avoidance.

f) Visualization Tools:: The simulation setup is complemented by several visualization tools:

- ROS2 Node Graph using `rqt_graph`
- Gazebo World Screenshots
- RViz visualization of the robot and sensor data
- Velocity/Trajectory Graphs for performance analysis

g) ROS2 Communication:: The simulation leverages ROS2 for inter-node communication.

B. Results

Performance Metrics to Evaluate Controllers

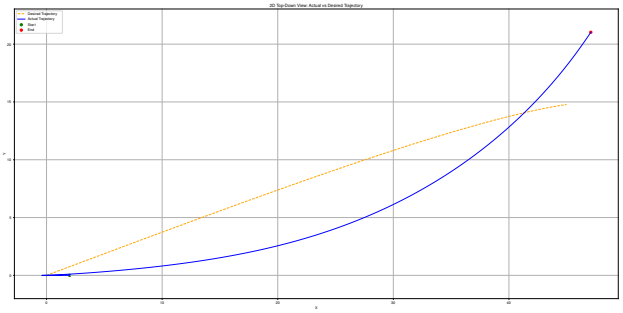


Fig. 3: Trajectory Tracking Using Cascaded Controller

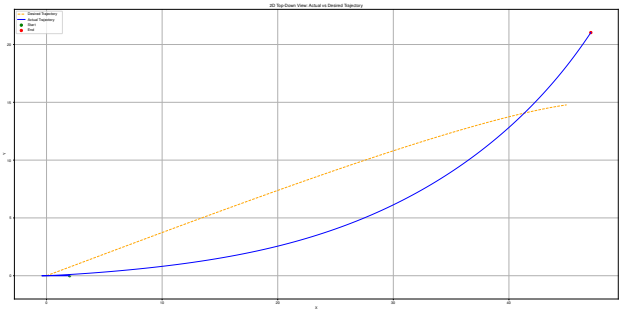


Fig. 4: Trajectory Tracking Using PID Controller

1) **Accuracy and Tracking Performance**

- **Tracking Error:**
  - Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) between desired and actual trajectory.
  - Maximum Error (e.g., overshoot, peak deviation).
  - Final Position Error (steady-state error).
  - Integral of Squared Error)
  - Integral of Absolute Error
  - Integral of Time-weighted Squared Error
  - Integral of Time-weighted Absolute Error
- **Time to Reach Target:** Time taken to converge to within a tolerance of the target state or trajectory.

Metric	Definition	Cascaded	PID
ISE	$\int_0^\infty e(t)^2\,dt$	32242.4039	29157.9052
IAE	$\int_0^\infty \lvert e(t)\rvert\,dt$	1973.2149	1858.1065
ITSE	$\int_0^\infty t\cdot e(t)^2\,dt$	2433310.4169	2119309.2168
ITAE	$\int_0^\infty t\cdot \lvert e(t)\rvert\,dt$	153288.3224	139126.2826
RMSE	$\sqrt{\frac{1}{T}\int_0^T e^2(t)\,dt}$	14.2135	13.9290

TABLE III: Integral Performance Criteria for Tracking Error

IV. CONCLUSIONS

–

REFERENCES

[1] Bashir, A.; Khan, S.; Iqbal, N.; Bashmal, S.; Ullah, S.; Fayyaz; Usman, M. A Review of the Various Control Algorithms for Trajectory Control of Unmanned Underwater Vehicles. Sustainability 2023, 15, 14691. https://doi.org/10.3390/su152014691

[2] R. Makam, P. Mane, S. Sundaram, and P. B. Sujit, "A Comprehensive Study on Modelling and Control of Autonomous Underwater Vehicle," arXiv preprint arXiv:2312.02690, 2024. [Online]. Available: https://arxiv.org/abs/2312.02690

[3] L. V. Steenson, L. Wang, A. B. Phillips, S. R. Turnock, M. E. Furlong, and E. Rogers, "Experimentally Verified Depth Regulation for AUVs Using Constrained Model Predictive Control," *IFAC Proc. Volumes*, vol. 47, no. 3, pp. 11974–11979, 2014, doi: 10.3182/20140824-6-ZA-1003.01497.

[4] Vadapalli, S.; Mahapatra, S. 3D Path Following Control of an Autonomous Underwater Robotic Vehicle Using Backstepping Approach Based Robust State Feedback Optimal Control Law. J. Mar. Sci. Eng. 2023, 11, 277. https://doi.org/10.3390/jmse11020277

[5] H. N. Esfahani, B. Aminian, E. I. Grøtli, and S. Gros, "Backstepping-based Integral Sliding Mode Control with Time Delay Estimation for Autonomous Underwater Vehicles," arXiv preprint arXiv:2111.10179, 2021. [Online]. Available: https://arxiv.org/abs/2111.10179

[6] Zhou J, Zhao X, Feng Z, Wu D. Trajectory tracking sliding mode control for underactuated autonomous underwater vehicles with time delays. International Journal of Advanced Robotic Systems. 2020;17(3). doi:10.1177/1729881420916276

[7] Vu, Q.V.; Dinh, T.A.; Nguyen, T.V.; Tran, H.V.; Le, H.X.; Pham, H.V.; Kim, T.D.; Nguyen, L. An Adaptive Hierarchical Sliding Mode Controller for Autonomous Underwater Vehicles. Electronics 2021, 10, 2316. https://doi.org/10.3390/electronics10182316

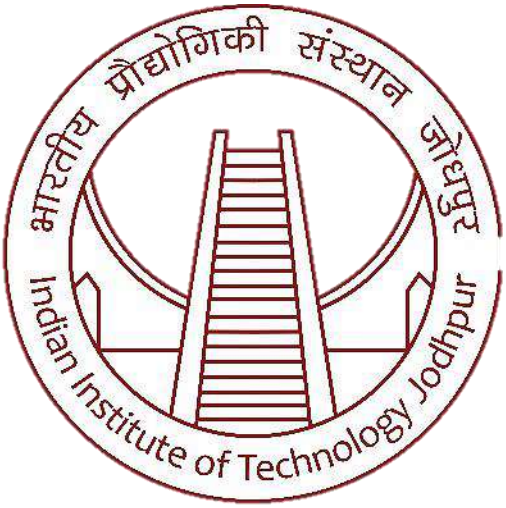
[8] Liang X, Wan L, Blake JIR, Shenoi RA, Townsend N. Path following of an Underactuated AUV Based on Fuzzy Backstepping Sliding Mode Control. International Journal of Advanced Robotic Systems. 2016;13(3). doi:10.5772/64065

[9] T.I.Fossen, "Maneuvering Theory," in Handbook of Marine Craft Hydrodynamics and Motion Control, Ch. 6, pp. 109–132, John Wiley Sons, Ltd, 2011. [Online]. Available: https://doi.org/10.1002/9781119994138.ch6

[10] , "MBARI Tethys LRAUV," Open Robotics, Apr. 2, 2021. [Online]. Available: https://fuel.gazebosim.org/1.0/accurrent/models/MBARI

CSL 7360

COMPUTER VISION



Assignment -1

Submitted by: -

ARASHDEEP SINGH

M23IRM003

Google Colab Link for all answers

https://colab.research.google.com

Question 1

Harris Corner detection: Your task is to implement the Harris Corner detection technique. You have to do the following:

- Implement the algorithm from scratch. You should not use any in-built functions.
- Adjust parameters such as the window size and threshold value to optimize corner detection performance.
- Make the code modularise to take images from the folder as an input.
- Compare the performance of your algorithm with the corner detection libraries from OpenCV.

Solution:-

Here's a simplified explanation of how the Harris corner detection algorithm works:

- Gradient Calculation: Compute the gradient of the image to determine how intensity values change in both the x and y directions. This can be done using Sobel operator.
- Structure Matrix Calculation: For every pixel in the image, calculate a structure tensor, which is a matrix summarizing the gradient information in the local neighborhood of that pixel. The structure tensor is usually represented as a 2x2 matrix:

$$M=\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

where:

I_x and I_y are the gradients in the x and y directions, respectively.

$w(x,y)$ is a weight function typically representing a Gaussian window centered at the pixel.

The summation is over the local neighborhood of the pixel.

- Corner Response Function: Compute a corner response function for each pixel using the structure tensor. The Harris corner response function is defined as:

$$R=\det(M)-k\cdot(\text{trace}(M))^2$$

where:

$\det(M)$ is the determinant of the structure tensor.

trace(M) is the trace of the structure tensor.
k is an empirically determined constant usually between 0.04 and 0.06.

- Non-maximum Suppression: Threshold the corner response function to identify potential corner points. Then, apply non-maximum suppression to eliminate multiple responses in close proximity and keep only the local maxima.

a) Here is the code implementing above-said procedure:-

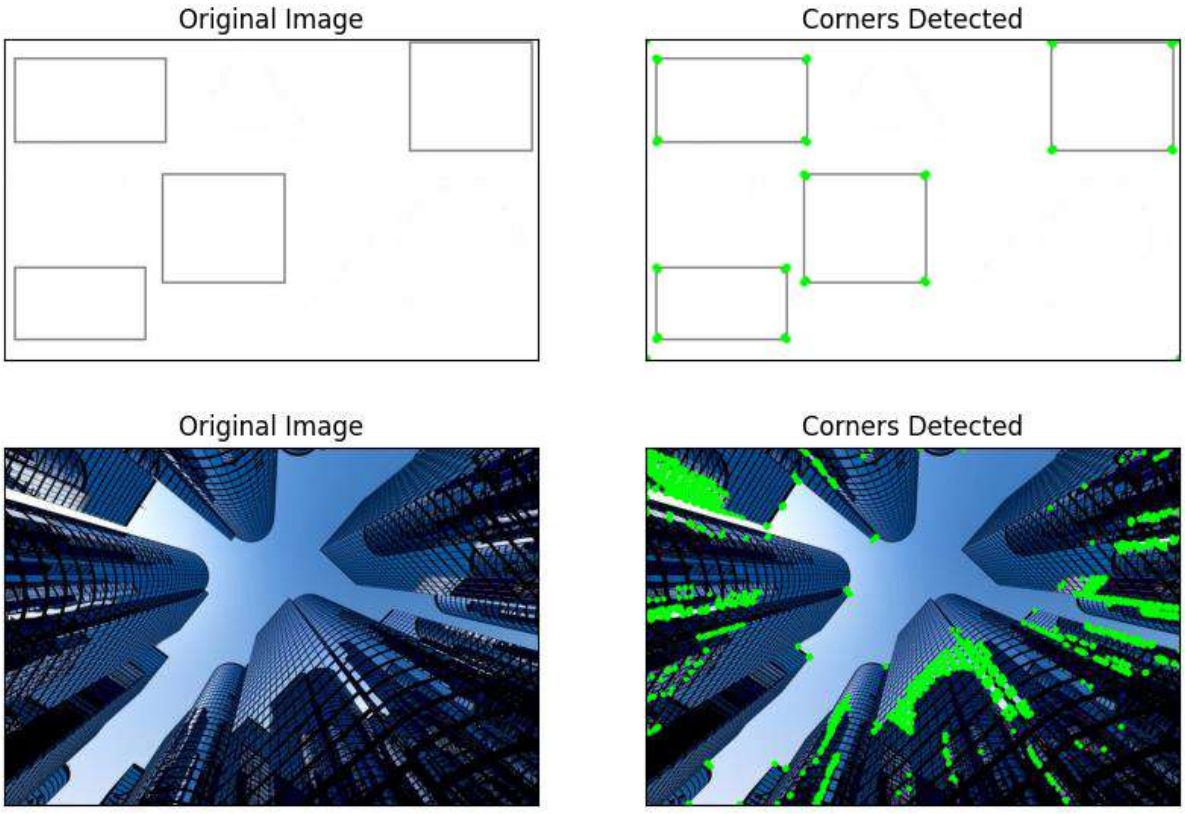
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os

SOBEL_X = np.array((
  [-1, 0, 1],
  [-2, 0, 2],
  [-1, 0, 1]), dtype="int32")

SOBEL_Y = np.array((
  [-1, -2, -1],
  [0, 0, 0],
  [1, 2, 1]), dtype="int32")

GAUSS = np.array((
  [1/16, 2/16, 1/16],
  [2/16, 4/16, 2/16],
  [1/16, 2/16, 1/16]), dtype="float64")

def convolve(img, kernel):


  kernel_sum = kernel.sum()
  if kernel_sum != 0:
```

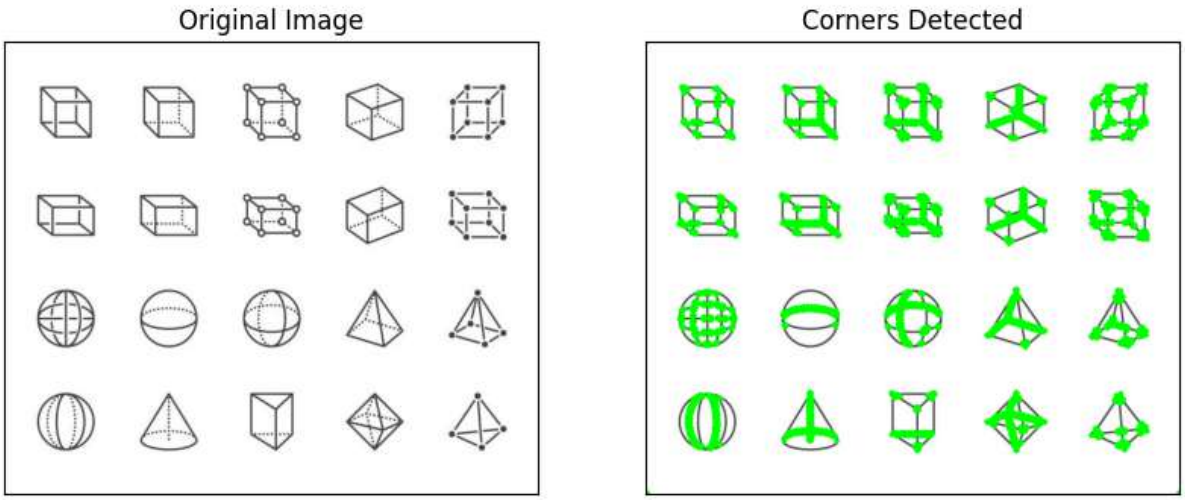
```
    kernel = kernel / kernel_sum

  img_height = img.shape[0]
  img_width = img.shape[1]
  pad_height = kernel.shape[0] // 2
  pad_width = kernel.shape[1] // 2


  output = np.zeros_like(img, dtype=np.float64)

  padded_img = np.pad(img, ((pad_height, pad_height), (pad_width, pad_width)),
mode='constant', constant_values=0)


  for i in range(pad_height, img_height + pad_height):
    for j in range(pad_width, img_width + pad_width):
      roi = padded_img[i - pad_height:i + pad_height + 1, j - pad_width:j + pad_width +
1]
      output[i - pad_height, j - pad_width] = (roi * kernel).sum()

  return output

def harris(img, threshold=0.05):

  img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  img_gray = np.float32(img_gray)

  dx = convolve(img_gray, SOBEL_X)
  dy = convolve(img_gray, SOBEL_Y)

  dx2 = dx * dx
  dy2 = dy * dy
  dxdy = dx * dy

  g_dx2 = convolve(dx2, GAUSS)
```

```
    g_dy2 = convolve(dy2, GAUSS)
    g_dxdy = convolve(dxdy, GAUSS)

    det_m = g_dx2 * g_dy2 - g_dxdy ** 2
    trace_m = g_dx2 + g_dy2
    harris_response = det_m - 0.04 * (trace_m ** 2)


    corners = np.zeros_like(harris_response)
    corners[harris_response > threshold * harris_response.max()] = 1


    coords = np.argwhere(corners)

    img_corners = img.copy()
    for coord in coords:
        x, y = coord
        cv2.circle(img_corners, (y, x), 3, (0, 255, 0), -1)

    return img_corners, coords
folder_path = '/content/drive/MyDrive/Question 1-20240314T200206Z-001/Question 1/'

image_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]

for image_file in image_files:

    img_path = os.path.join(folder_path, image_file)

    img = cv2.imread(img_path)


    if img is None:
        print(f"Error: Image {image_file} not found.")
        continue
```

```
img_corners, corner_coords = harris(img, 0.05)


plt.figure(figsize=(10, 10))
plt.subplot(121), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(cv2.cvtColor(img_corners, cv2.COLOR_BGR2RGB))
plt.title('Corners Detected'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output





Explanation of Code

Sure, let's break down the code step by step:

1. The code begins by importing necessary libraries: 'numpy' for numerical operations, 'cv2' (OpenCV) for computer vision tasks, 'matplotlib.pyplot' for plotting images, and 'os' for interacting with the operating system.

2. It defines three matrices: 'SOBEL_X', 'SOBEL_Y', and 'GAUSS'. These matrices represent Sobel operator kernels for detecting gradients in horizontal and vertical directions, and a Gaussian kernel for image blurring respectively.

3. There's a function 'convolve' which performs convolution operation on an image with a given kernel. Convolution is a mathematical operation used for applying filters to images.

4. The 'harris' function is defined for detecting corner in image using the Harris corner detection algorithm. Here's a breakdown of what it does:

- Converts the input image to grayscale.

- Applies Sobel operators ('SOBEL_X' and 'SOBEL_Y') to calculate gradients in horizontal and vertical directions.

- Computes elements required for Harris corner detection: (Ix^2), (Iy^2), (Ix dot Iy), Gaussian blurring of these elements.

- Computes the Harris corner response using the formula $R=\det(M)-k\cdot(\text{trace}(M))^2$

  - Thresholds the corner response to identify strong corners.

  - Finds coordinates of corners and marks them on the image.

  - Returns the image with corners marked and the coordinates of detected corners.

5. The code defines a folder path containing images to be processed.

6. It iterates over each image file in the specified folder:

  - Reads the image using OpenCV ('cv2.imread').

  - If the image is not found, it prints an error message and continues to the next image.

  - Applies the 'harris' function to detect corners in the image.

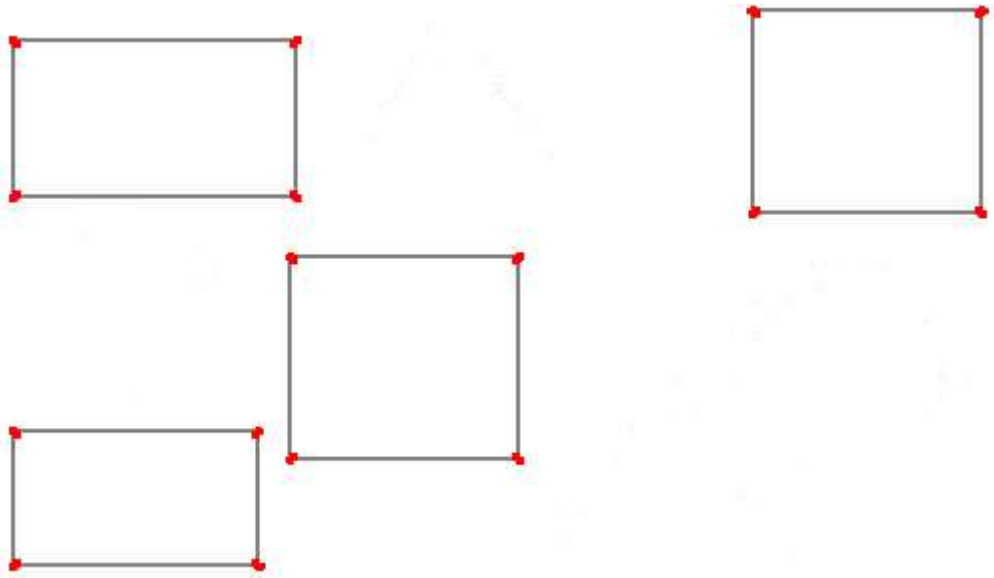
  - Plots the original image and the image with detected corners using 'matplotlib'.

b) Using Open CV

```
import cv2
import os

def harris_corner_detection(image_path):

  img = cv2.imread(image_path)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

  corners = cv2.cornerHarris(gray, 2, 3, 0.05)
  corners = cv2.dilate(corners, None)

  img[corners > 0.01 * corners.max()] = [0, 0, 255]
  from google.colab.patches import cv2_imshow
  cv2_imshow( img)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
```

```
def main(folder_path):

  image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.jpeg', '.png'))]

  for image_file in image_files:
    image_path = os.path.join(folder_path, image_file)
    print("Processing:", image_path)
    harris_corner_detection(image_path)

if __name__ == "__main__":

  folder_path = '/content/drive/MyDrive/Question 1-20240314T200206Z-001/Question 1'
  main(folder_path)
```

Output



Explanation of Code

1. The code imports necessary libraries: ‘cv2‘ for computer vision tasks and ‘os‘ for interacting with the operating system.

2. It defines a function ‘harris_corner_detection(image_path)‘:

- Reads an image from the specified ‘image_path‘ using ‘cv2.imread()‘.

- Converts the image to grayscale using the function cv2.cvtColor() because Harris corner detection typically operates on grayscale images.

- Applies the Harris corner detection algorithm using ‘cv2.cornerHarris()‘. This function calculates a Harris response map indicating the likelihood of corners at each pixel.

- Dilates the corner points using ‘cv2.dilate()‘ to enhance the corner points.

- Marks the corners on the original image by changing the pixel values to red where the corner response is above a certain threshold.

- Displays the image with detected corners using ‘cv2_imshow()‘ (specific to Google Colab).

- Waits for a key press and then closes all windows using ‘cv2.waitKey(0)‘ and ‘cv2.destroyAllWindows()‘ respectively.

3. The ‘main(folder_path)‘ function:

- Takes a folder path as input.
- Lists all image files (with extensions ’.jpg’, ’.jpeg’, ’.png’) in the specified folder.
- Iterates over each image file:
  - Constructs the full image path.
  - Prints a message indicating the processing of the current image.
  - Calls ‘harris_corner_detection()‘ function to detect corners in the image.

4. The ‘if __name__ == "__main__":‘ block ensures that ‘main()‘ function is executed only when the script is run directly, not when it’s imported as a module into another script.

Comparison of Both approaches:-

1. Simplicity:
- The first code is more complex as it involves manually implementing convolution operations for gradient calculations and Gaussian blurring. It also requires understanding of image processing concepts such as Sobel operators and Harris corner detection algorithm.
- The second code is simpler and more concise. It utilizes built-in functions provided by OpenCV (‘cv2.cornerHarris()‘, ‘cv2.dilate()‘, etc.) for corner detection, eliminating the need for manual implementation of these operations. It’s easier to understand and maintain.

2. Speed:
- The first code may be slower due to manual implementation of convolution operations and iteration over image pixels. Although NumPy operations are optimized for

performance, the manual implementation may not be as efficient as the optimized functions provided by OpenCV.

- The second code is likely faster because it leverages highly optimized functions provided by OpenCV for image processing tasks. OpenCV is written in C/C++, so its functions are typically very efficient and optimized for speed.

3. Time Consumption:

- The first code may consume more time during development and debugging due to its complexity. It requires understanding and debugging of custom implementations of convolution and Harris corner detection algorithm.
- The second code reduces development and debugging time significantly because it relies on built-in functions that abstract away complex image processing operations. It's easier to write, understand, and debug.

In summary, the second code is simpler, faster, and consumes less time compared to the first code. It achieves the same task of Harris corner detection with significantly less effort and complexity, thanks to the use of high-level functions provided by OpenCV.

Overall, both codes have the potential to provide good-quality output in terms of detecting corners in images. The first code offers more control and customization options, while the second code provides a simpler and more efficient approach with reliable results. The choice between the two would depend on the specific requirements of the application and the desired balance between flexibility and simplicity.

Question 2

Consider two stereo images I1 ("bikeL.png") and I2 ("bikeR.png") of a static scene captured from a stereo camera with the given intrinsic matrices of both the cameras in the file ("bike.txt"). Implement the stereo 3D reconstruction algorithm to find the disparity map, depth map (depth map at each pixel), and 3D point cloud representation of the underlying scene.

Solution:-

1. Stereo Calibration:
    - Determine the intrinsic parameters (focal length, principal point, distortion coefficients) and extrinsic parameters (rotation and translation) of the stereo camera setup.
    - Involves capturing images of a calibration pattern (e.g., checkerboard) from different viewpoints and using known 3D coordinates of the pattern to compute the camera parameters. The process is typically performed using libraries like OpenCV, which provide functions like calibrate Camera.
2. Rectification:
    - Align the stereo images so that corresponding epipolar lines become horizontal.
    - Calculates transformation matrices for each camera to bring their image planes into alignment. This simplifies stereo matching by ensuring that corresponding points lie along the same rows in the rectified images. OpenCV's stereoRectify function is commonly used for this purpose.
3. Stereo Matching:
    - Find corresponding points between rectified stereo images.
    - Involves comparing pixel intensities in corresponding scanlines of the rectified images to find disparities (horizontal offsets) between them. Various algorithms can be used for stereo matching, such as block matching, semi-global matching (SGM), or deep learning-based methods. OpenCV provides implementations of these algorithms, such as StereoBM, StereoSGBM, and StereoMatcher.
4. Disparity Map Calculation:
    - Represent the disparity (horizontal shift) between corresponding points in the rectified stereo images.
    - Stereo matching algorithms produce a disparity map, where each pixel value indicates the horizontal shift needed to align corresponding points. This map is a crucial intermediate result in stereo reconstruction.
5. Depth Map Computation:
    - Calculate the depth (distance from the camera) at each pixel in the rectified images.
    - Uses the disparity map and camera parameters (focal length, baseline) to compute depth values for each pixel. The depth at a pixel is inversely proportional to the

disparity and directly proportional to the baseline (distance between the cameras). The formula mentioned earlier is used for this computation.

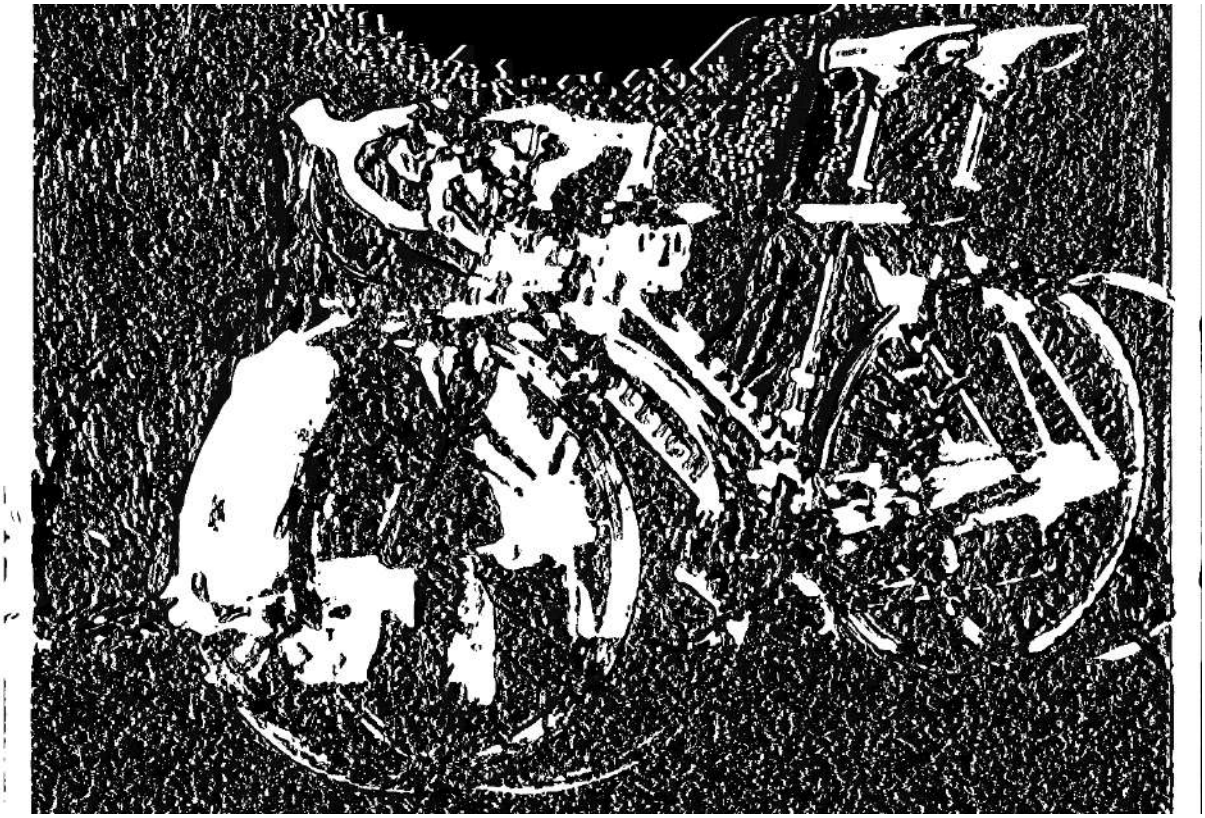
6. 3D Point Cloud Generation:
    - Reconstruct the 3D coordinates of points in the scene from the depth map.
    - For each pixel with a valid depth value, the 3D coordinates are computed using the depth value and pixel coordinates, along with the camera parameters. This results in a point cloud representation of the scene, where each point corresponds to a 3D point in the real world.

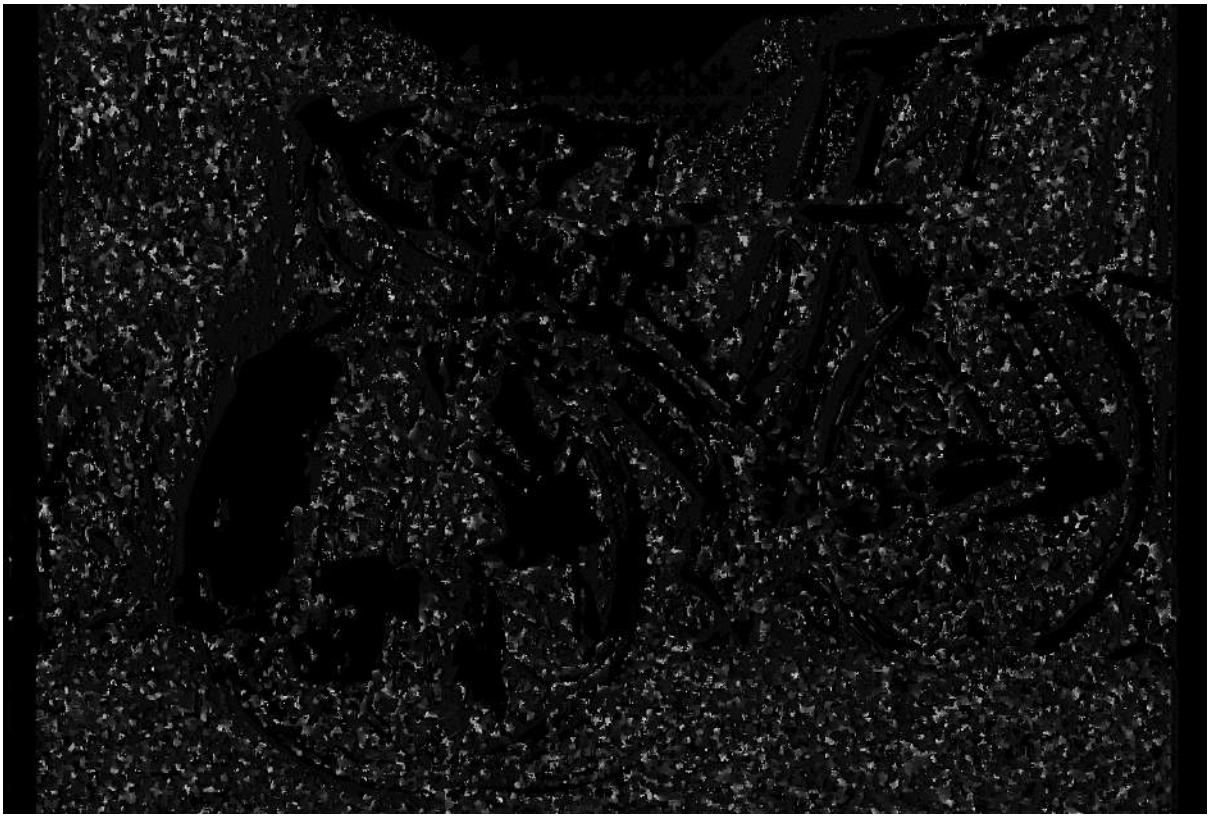
These steps together constitute the stereo 3D reconstruction pipeline, which takes stereo image pairs as input and produces a dense 3D representation of the scene as output.

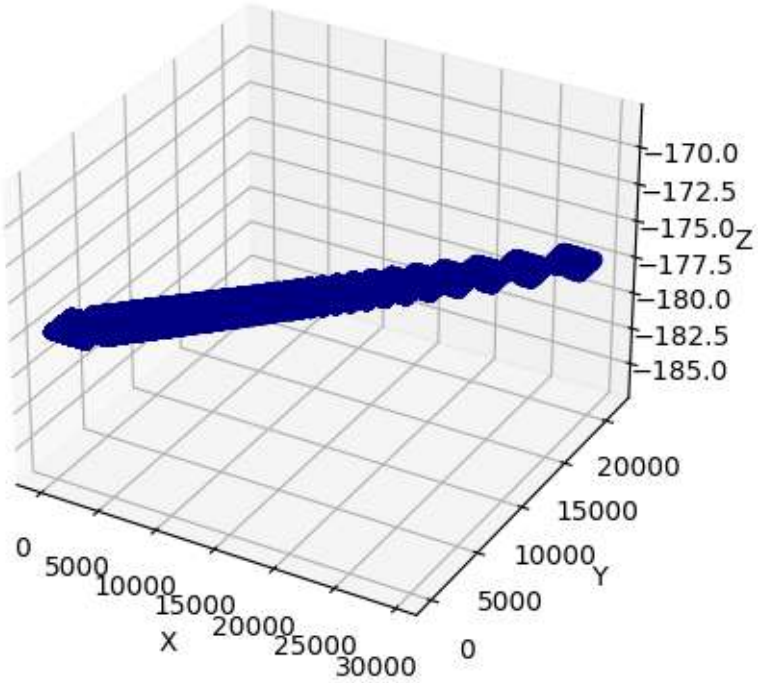
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

imgL = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/bikeL.png',
0)
imgR = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/bikeR.png',
0)


try:
    with open('/content/drive/MyDrive/Question 2 and 3 Images/bike.txt', 'r')
as f:
        content = f.readlines()
        K_left = np.array(eval(content[0]))
        K_right = np.array(eval(content[1]))
except Exception as e:
    print(f"Failed to load camera matrices: {e}")


if 'K_left' in locals():
    print("K_left shape:", K_left.shape)
if 'K_right' in locals():
    print("K_right shape:", K_right.shape)


if 'K_left' in locals() and not (isinstance(K_left, np.ndarray) and
K_left.shape == (3, 3)):
    print("K_left is not a valid camera matrix.")
if 'K_right' in locals() and not (isinstance(K_right, np.ndarray) and
K_right.shape == (3, 3)):
    print("K_right is not a valid camera matrix.")
```

```
dist_coeffs_left = None
dist_coeffs_right = None


baseline = 177.288
focal_length = K_left[0, 0]


T = np.array([[baseline], [0], [0]])


R = np.eye(3)


R1, R2, P1, P2, Q, _, _ = cv2.stereoRectify(K_left, dist_coeffs_left,
K_right, dist_coeffs_right,
                                            imgL.shape[::-1], R, T,
flags=cv2.CALIB_ZERO_DISPARITY)


map1x, map1y = cv2.initUndistortRectifyMap(K_left, dist_coeffs_left, R1, P1,
imgL.shape[::-1], cv2.CV_32FC1)
map2x, map2y = cv2.initUndistortRectifyMap(K_right, dist_coeffs_right, R2,
P2, imgR.shape[::-1], cv2.CV_32FC1)


rectified_imgL = cv2.remap(imgL, map1x, map1y, cv2.INTER_LINEAR)
rectified_imgR = cv2.remap(imgR, map2x, map2y, cv2.INTER_LINEAR)

stereo = cv2.StereoSGBM_create(numDisparities=14, blockSize=15)
disparity = stereo.compute(rectified_imgL, rectified_imgR)


depth_map = np.zeros_like(disparity, dtype=np.float32)
depth_map[disparity > 0] = baseline * focal_length / disparity[disparity > 0]


h, w = imgL.shape
y, x = np.mgrid[0:h, 0:w]
points = np.array([x, y, depth_map[y, x]])


points_3d = np.vstack((points.reshape(3, -1), np.ones((1, h*w))))


points_3d_world = np.dot(np.linalg.inv(Q), points_3d)


x_world, y_world, z_world = points_3d_world[:3, :].astype(np.float32)
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_world, y_world, z_world, c=z_world, cmap='jet')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
from google.colab.patches import cv2_imshow

cv2_imshow( (disparity/16).astype(np.uint8))
cv2_imshow( (depth_map/depth_map.max()*255).astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```






170.0
172.5
175.0
177.5
180.0
182.5
185.0
Z
0
5000
10000
15000
20000
25000
30000
X
0
5000
10000
15000
20000
Y

Question 3

Consider two images I1 ("000000.png") and I2 ("000023.png") of a static scene captured from a single camera with the given Fundamental matrix F ("FM.txt").Write down a code to find the epipolar lines and draw the epipolar lines in both theimages. Now, consider uniformly spaced 10 pixels on the epipolar line in the firstimage and find the corresponding pixels on the second epipolar line. Now, consideruniformly spaced 10 pixels on the epipolar line in the second image and find thecorresponding pixels on the epipolar line in the first image. You should search only on the epipolar lines.

Solution:-

Here is the procedure to find epipolar lines:-

1. Find a keypoint using SIFT detector to detect distinctive points.
2. Select a point in first image, represent it in homogenous coordinates.
3. Using fundamental matrix find equation of epipolar line in second image. (matrix multiplication of position vector and fundamental matrix will result in a vector. The elements of this vector will represent the coefficients of equation of line.

A little about SIFT

1. Scale-Space Peak Selection:
    - The first step is to build a scale space, which is a series of images that are progressively blurred using a Gaussian filter.
    - This is done to identify potential keypoints that are invariant to scale changes.
    - The scale space is divided into octaves, each octave being a series of images where each subsequent image is blurred more than the previous one.
    - After each octave, the image is downsampled, and the process is repeated.
    - Potential keypoints are then identified as local maxima and minima of the Difference of Gaussian (DoG) images obtained by subtracting adjacent Gaussian-blurred images within an octave.
2. Keypoint Localization:
    - Once potential keypoints are found, they are refined to get more accurate locations.
    - This involves a detailed fit to the nearby data for location and scale.
    - This step helps in eliminating points that have low contrast are poorly localized along an edge.
3. Orientation Assignment:
    - Each keypoint is assigned one or more orientations based on local image gradient directions.

- This is achieved by creating a histogram of gradient orientations within a region around the keypoint.
- The peaks in this histogram correspond to dominant directions of local gradients.
- This ensures that the keypoint descriptor is invariant to image rotation.

4. Keypoint Descriptor:
   - The local gradients are measured around each keypoint, and these are transformed into a descriptor.
   - The descriptor is a vector that captures the intensity gradients in various directions and at different scales around the keypoint.
   - This vector is designed to give a robust and distinctive representation of the keypoint that allows for significant levels of local shape distortion and change in illumination.
5. Keypoint Matching:
   - The final step is to match keypoints between different images.
   - This is done by finding the nearest neighbor of each keypoint descriptor in the database of keypoints from other images.
   - A match is confirmed based on the distance ratio between the closest neighbor and the second-closest neighbor, ensuring that the match is unique and reliable.

At first I have developed code to detect key points in both images.

```
import numpy as np
import cv2
from scipy.ndimage import gaussian_filter

img1 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000000.png')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

img2 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000023.png')
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()

keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

keypoints_with_orientations1 = []
for keypoint in keypoints1:
  angle = keypoint.angle
```

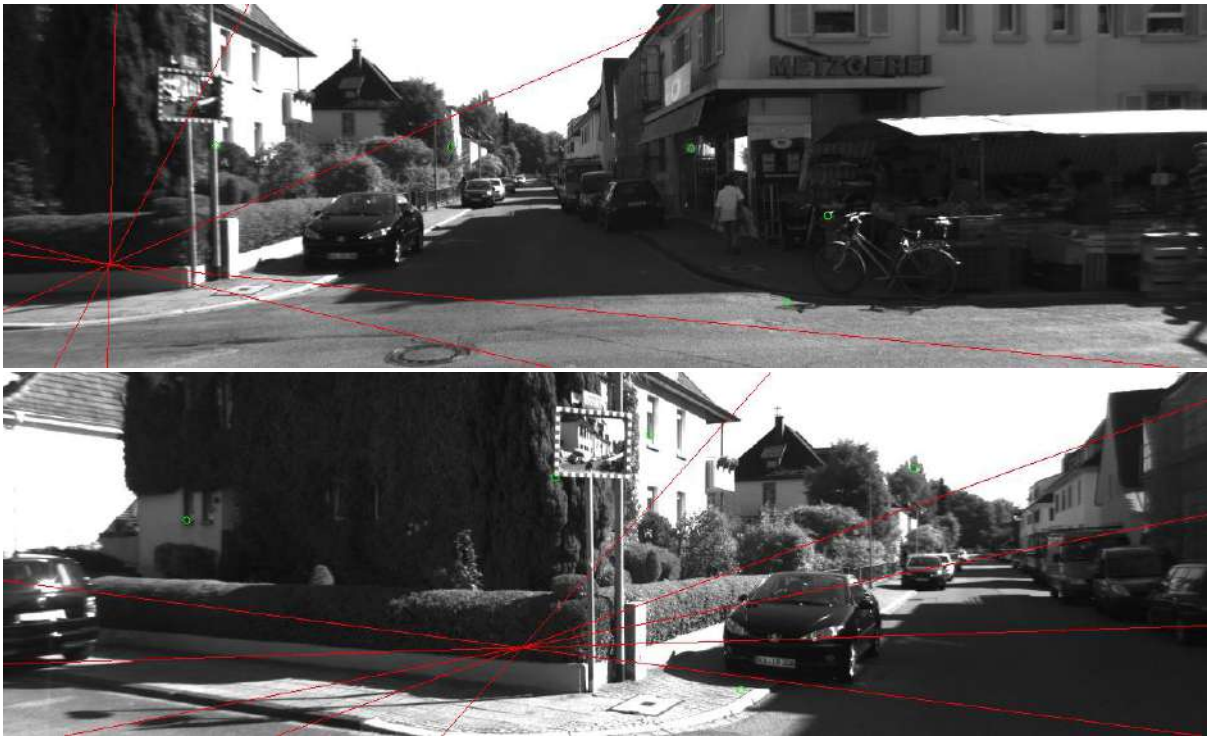
```
    if angle < 0:
        angle += 360
    keypoints_with_orientations1.append((keypoint.pt, angle))

keypoints_with_orientations2 = []
for keypoint in keypoints2:
    angle = keypoint.angle
    if angle < 0:
        angle += 360
    keypoints_with_orientations2.append((keypoint.pt, angle))

for keypoint in keypoints_with_orientations1:
    x, y = np.int32(keypoint[0])
    angle = keypoint[1]
    cv2.circle(img1, (x, y), 4, (0, 255, 0), 1)
    cv2.line(img1, (x, y), (int(x + 10 * np.cos(np.radians(angle))), int(y + 10 * np.sin(np.radians(angle)))),
(0, 255, 0), 1)

for keypoint in keypoints_with_orientations2:
    x, y = np.int32(keypoint[0])
    angle = keypoint[1]
    cv2.circle(img2, (x, y), 4, (0, 255, 0), 1)
    cv2.line(img2, (x, y), (int(x + 10 * np.cos(np.radians(angle))), int(y + 10 * np.sin(np.radians(angle)))),
(0, 255, 0), 1)

cv2_imshow( img1)
cv2_imshow( img2)
cv2.waitKey(0)
```

Here is the output of the code

Now I will chose 5 random keypoints in one image and will draw epipolar line on other image and vice-versa.

```
import numpy as np
import cv2

img1 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000000.png')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img1_display = img1.copy()

img2 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000023.png')
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
img2_display = img2.copy()

sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)
```

```
F = np.array([[3.34638533e-07,  7.58547151e-06, -2.04147752e-03],
          [-5.83765868e-06,  1.36498636e-06, 2.67566877e-04],
          [1.45892349e-03, -4.37648316e-03,  1.00000000e+00]])
selected_indices1 = np.random.choice(len(keypoints1), 5, replace=False)
selected_indices2 = np.random.choice(len(keypoints2), 5, replace=False)

for i in range(5):
  pt1 = np.int32(keypoints1[selected_indices1[i]].pt)
  cv2.circle(img1_display, tuple(pt1), 4, (0, 255, 0), 1)
  pt2 = np.int32(keypoints2[selected_indices2[i]].pt)
  cv2.circle(img2_display, tuple(pt2), 4, (0, 255, 0), 1)

  line1 = np.dot(F, np.array([*pt1, 1])).T
  line1 /= np.sqrt(line1[0]**2 + line1[1]**2)
  line2 = np.dot(F.T, np.array([*pt2, 1])).T
  line2 /= np.sqrt(line2[0]**2 + line2[1]**2)
  x0, y0, x1, y1 = map(int, [0, -line1[2]/line1[1], img2.shape[1], -
(line1[2]+line1[0]*img2.shape[1])/line1[1]])
  cv2.line(img2_display, (x0, y0), (x1, y1), (0, 0, 255), 1)


  x0, y0, x1, y1 = map(int, [0, -line2[2]/line2[1], img1.shape[1], -
(line2[2]+line2[0]*img1.shape[1])/line2[1]])
  cv2.line(img1_display, (x0, y0), (x1, y1), (0, 0, 255), 1)
from google.colab.patches import cv2_imshow
cv2_imshow( img1_display)
cv2_imshow( img2_display)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

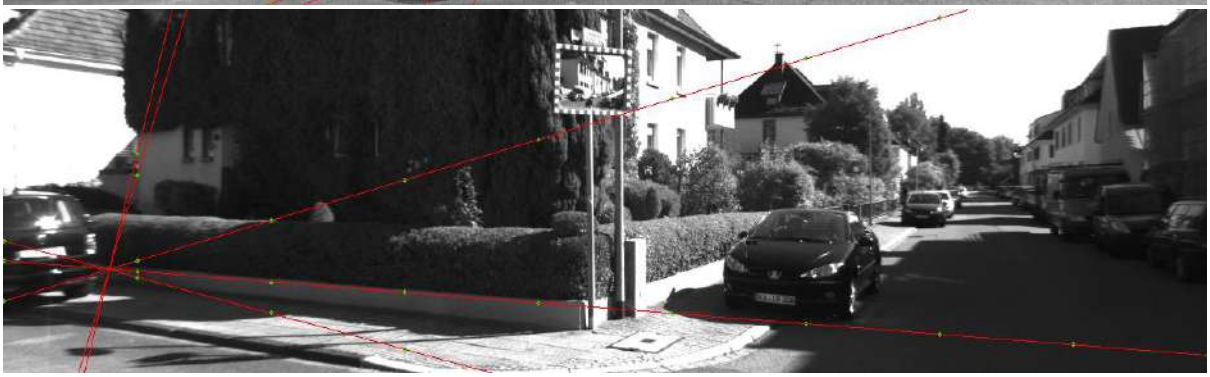


Now as per question choosing 10 equaly spaced point on epipolar line and drawing epipolar line on second image.

```
import cv2
import numpy as np
import os

def find_corresponding_pixels(line, image_shape, num_points=10):
    corresponding_pixels = []
    for i in range(num_points):
        x = int(i * image_shape[1] / (num_points - 1))
        y = int((-line[2] - line[0] * x) / line[1])
        corresponding_pixels.append((x, y))
    return corresponding_pixels

def draw_epipolar_lines(img, lines):
    for line in lines:
        x0, y0 = map(int, [0, -line[2]/line[1]])
```

```
    x1, y1 = map(int, [img.shape[1], -(line[2]+line[0]*img.shape[1])/line[1]])
    cv2.line(img, (x0, y0), (x1, y1), (0, 0, 255), 1)

def main():
  img1 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000000.png')
  gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
  img1_display = img1.copy()


  img2 = cv2.imread('/content/drive/MyDrive/Question 2 and 3 Images/000023.png')
  gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
  img2_display = img2.copy()


  F = np.array([[3.34638533e-07,  7.58547151e-06, -2.04147752e-03],
        [-5.83765868e-06,  1.36498636e-06, 2.67566877e-04],
        [1.45892349e-03, -4.37648316e-03,  1.00000000e+00]])
  sift = cv2.SIFT_create()

  keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
  keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

  selected_indices1 = np.random.choice(len(keypoints1), 5, replace=False)
  selected_indices2 = np.random.choice(len(keypoints2), 5, replace=False)


  epipolar_lines_img1 = []
  epipolar_lines_img2 = []
  for i in range(5):
    line1 = np.dot(F, np.array([*keypoints1[selected_indices1[i]].pt, 1])).T
    line1 /= np.sqrt(line1[0]**2 + line1[1]**2)
    line2 = np.dot(F.T, np.array([*keypoints2[selected_indices2[i]].pt, 1])).T
    line2 /= np.sqrt(line2[0]**2 + line2[1]**2)

    corresponding_pixels_img1_to_img2 = find_corresponding_pixels(line2, gray2.shape)
    corresponding_pixels_img2_to_img1 = find_corresponding_pixels(line1, gray1.shape)

    for x, y in corresponding_pixels_img1_to_img2:
      cv2.circle(img2_display, (x, y), 2, (0, 255, 0), -1)
    for x, y in corresponding_pixels_img2_to_img1:
```

```
        cv2.circle(img1_display, (x, y), 2, (0, 255, 0), -1)

    epipolar_lines_img1.append(line1)
    epipolar_lines_img2.append(line2)

  draw_epipolar_lines(img1_display, epipolar_lines_img1)
  draw_epipolar_lines(img2_display, epipolar_lines_img2)

  cv2_imshow( img1_display)
  cv2_imshow( img2_display)
  cv2.waitKey(0)
  cv2.destroyAllWindows()

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



References:-

https://github.com/root-master/unified-hrl/blob/3f0851f02cb7ebfbb66edde817c5b4e542baf58d/montezuma-ICLR/image_processing.py

https://github.com/adityaintwala/Harris-Corner-Detection/blob/master/find_harris_corners.py

GitHub - saurabhkemekar/Stereo-3D-Reconstruction: This repo contains the implementation of 3D reconstruction using the pair of stereo images using OPENCV python

https://doi.org/10.1016/j.jcis.2018.10.026

https://fdocuments.fr/document/3d-reconstruction-from-multiple-views-based-on-scale-pic4-now-we-want-to-transform.html

https://doi.org/10.12785/amis/072L49

https://en-academic.com/dic.nsf/enwiki/604851

https://fdocuments.fr/document/3d-reconstruction-from-multiple-views-based-on-scale-pic4-now-we-want-to-transform.html

. https://doi.org/10.12785/amis/072L49

https://fdocuments.fr/document/3d-reconstruction-from-multiple-views-based-on-scale-pic4-now-we-want-to-transform.html

GitHub - cdcseacave/openMVS: open Multi-View Stereo reconstruction library

GitHub - agavazov/3d-stereo-vision: 3D Stereo Vision Research (Python, OpenCV, C++, OpenALPR)

OpenCV: cv::StereoBM Class Reference

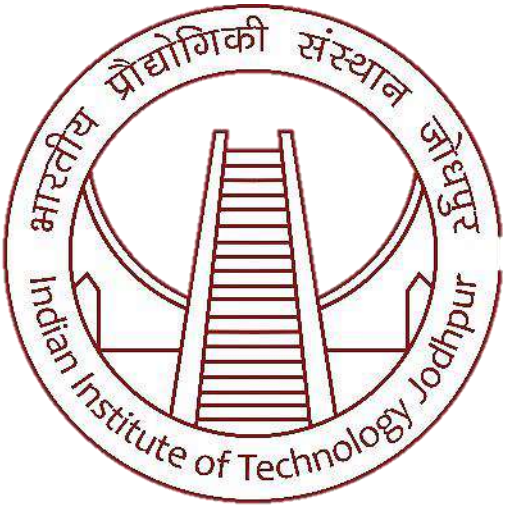
OpenCV: cv::StereoSGBM Class Reference

GitHub - mundheda/Direct-Linear-Transform_Epipolar-Lines: Direct linear Transform (2D-3D) from an image to given points in the 3D world. Then using Zhangs method to find K matrix of the Camera. Also Generating Epipolar Lines b/w two images with different camera positions.

GitHub - Sagarkaw1995/EpipolarLines_on_StereoImages

CSL 7360

COMPUTER VISION



Assignment -2

Submitted by: -

ARASHDEEP SINGH

M23IRM003

K-Means Segmentation

Let us understand this clustering algorithm.

First of all,we define number of clusters we want. Then for each cluster we choose a mean. Points closer to a mean will become part of that cluster. After that each cluster mean is computed again. The mean will have highest feature description of that segment. This process is repeated until it converges.

So based on this approach a python code is developed. I am describing main functions of code that define the flow of task.

1.Initialization

- n_clusters: The number of clusters to be formed (adjustable parameter).
- max_iter: The maximum number of iterations for the K-means algorithm (default: 100).
- random_state: Controls the randomness for centroid initialization (default: 123 for reproducibility).

2.Initialize centroids

This function initializes the centroids (cluster centers) for the K-means algorithm. It uses np.random.RandomState to set a seed for generating random numbers, ensuring consistent results. It then randomly permutes the indices of data points using np.random.permutation. Finally, it selects the first n_clusters data points (based on the permuted indices) as the initial centroids.

3. Compute centroids

This function calculates the new centroids based on the current cluster assignments. It iterates through each cluster (k). For each cluster, it computes the mean of the data points assigned to that cluster. The function returns an updated array of centroids with the means for each cluster.

4. Compute distance from centroid

This function calculates the squared Euclidean distance between each data point and all centroids.It creates a distance matrix (distance) with dimensions (number of data points, number of clusters).For each cluster, it computes the squared distance between all data points and the corresponding centroid. The function returns the distance matrix

5. Find closest cluster

This function finds the closest cluster for each data point based on the computed distances.It uses np.argmin along the axis=1 (columns) to find the index of the minimum value in each row of the distance matrix. The index corresponds to the closest cluster for each data point, and the function returns this array of cluster labels.

6 Compute SSE

This function calculates the Sum of Squared Errors (SSE) for the current clustering. It iterates through each cluster (k). For each cluster, it computes the squared Euclidean distances between data points assigned to that cluster and its centroid. The function sums these squared distances for all clusters and returns the total SSE.

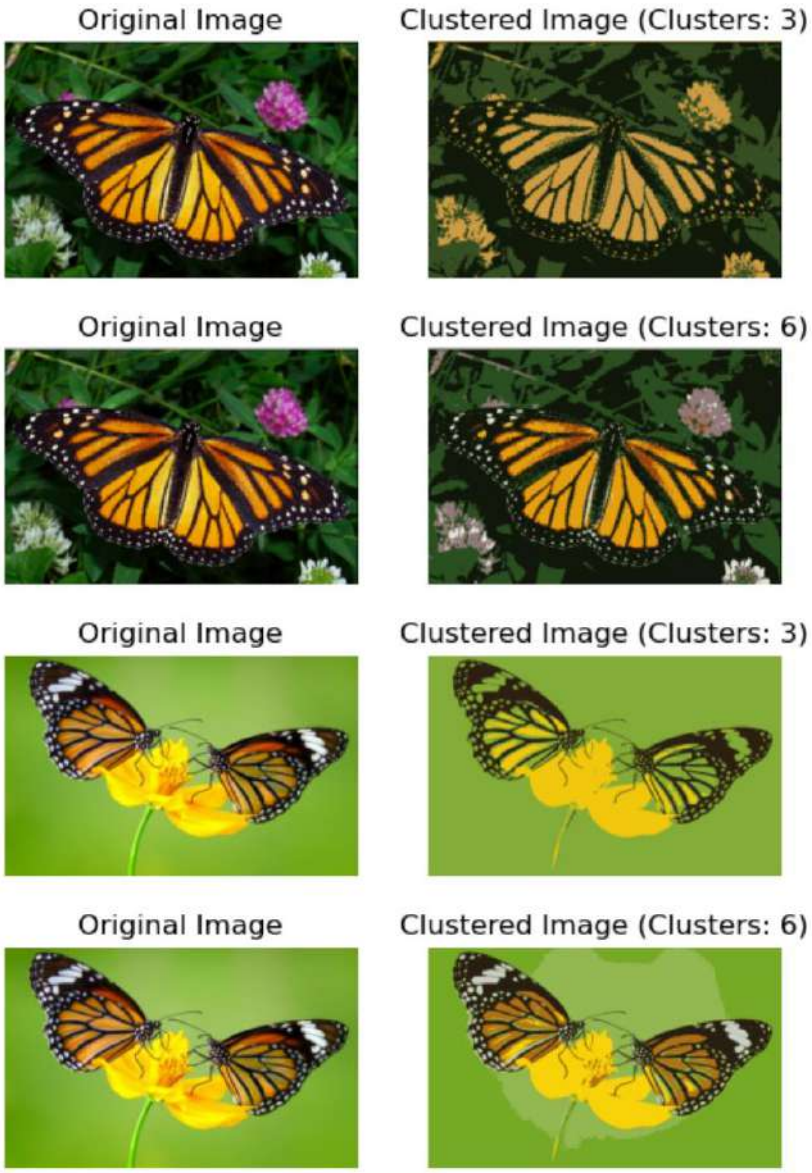
7. Fit

- This function implements the core K-means algorithm.It initializes the centroids using initializ_centroids. The function stores the final centroids and the computed SSE (error).
- It iterates for a maximum of max_iter times:
    - It computes the distances between data points and current centroids using compute_distance.
    - It assigns each data point to the closest cluster using find_closest_cluster.
    - It recomputes the centroids based on the new cluster assignments using compute_centroids.
    - It checks if the centroids haven't changed, indicating convergence.

8. Predict

- This function predicts cluster labels for new data points. It calculates the distances between new data points and the trained centroids using compute_distance. It assigns each new data point to the closest cluster using find_closest_cluster. The function returns the predicted cluster labels for the new data points.

Here is the output of the code:-



Ratio-Cut Segmentation

It is a graph-cut segmentation technique. Here first of all a graph is made from image. Pixels are represented as vertices and edges represent similarity between vertices. Then a adjuncy matrix(A)/adffinity Matrix is formed representing weights of each edge. Then a second matrix degree matrix(D) is formed which represents how many vertices are connected to a particular one. It is a diagonal matrix. Then a Laplacian matrix is found by substracting adjuncy matrix from degree matrix. Then eigen vectors of Laplacian matrix is found which form cluster assignment matrix(H). There are two ways to find ration cut.

Ratio Cut = trace(H^TLH)

Or

$(D-W)x=\lambda x$

Here is the explanation of code

1. **Compute Affinity Matrix:**

   cdist. **dist_matrix_sq** is computed using the **cdist** function from **scipy.spatial.distance**. This function calculates the squared Euclidean distance between pairs of points in the input arrays **X** and **X**, resulting in a symmetric distance matrix where each element represents the squared Euclidean distance between two points.

   The **cdist** function with the parameter **'sqeuclidean'** computes the squared Euclidean distance because taking the square of the Euclidean distance simplifies computations and avoids the costly square root operation.

   After obtaining the squared distance matrix **dist_matrix_sq**, the code applies the Radial Basis Function (RBF) kernel to compute the affinity matrix **A**. The RBF kernel is defined as:

   $K(x,y)=e-\gamma\cdot||x-y||2K(x,y)=e-\gamma\cdot||x-y||2$

   Where:

   ee is the base of the natural logarithm (Euler's number).

   $\gamma\gamma$ is a parameter that determines the width of the kernel and controls the influence of each data point on its neighbors.

   $||x-y||2||x-y||2$ is the squared Euclidean distance between points xx and yy.

2. **Compute Degree Matrix:**

   The degree matrix (D) is a diagonal matrix where each element D[i, i] represents the sum of the similarities between pixel i and all other pixels in the image (its degree).

3. **Construct Laplacian Matrix:**

   The Laplacian matrix (L) is indirectly constructed by subtracting the degree matrix (D) from the affinity matrix (A). This highlights regions with high connectivity between pixels (low similarity values in L).

4. **Dimensionality Reduction:**

   Since spectral clustering relies on eigenvalues and eigenvectors, we perform Singular Value Decomposition (SVD) on the Laplacian matrix (M in the code, which is equivalent to L here) using linalg.svd. Only a subset of the top k+1 eigenvectors (Usubset) corresponding to the smallest eigenvalues are kept. This step reduces the dimensionality of the data while preserving the most relevant information for clustering.

5. **K-Means Clustering:**

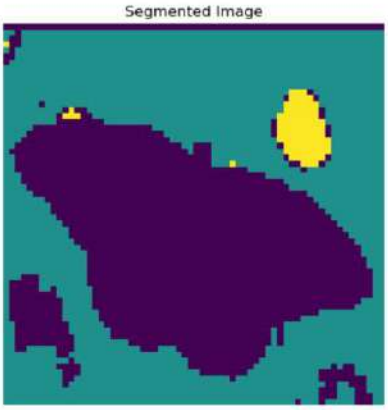
   - K-Means clustering is applied to the normalized Usubset matrix using KMeans from scikit-learn. The number of clusters is set to k.By applying K-Means to the reduced-dimensionality data, the algorithm identifies groups of pixels with similar characteristics.

6. **Image Segmentation:**

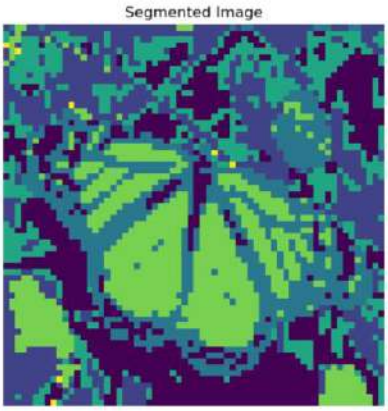
The cluster labels obtained from K-Means are reshaped back into the original image dimensions (64x64) to create a segmented image. Each pixel now has a label representing its assigned cluster.

Here is the output of the code:-

```
Processing: Assign2\image1.jpg
Displaying segmented image with 3 clusters
```




```
Displaying segmented image with 6 clusters
```



Alternative Approach using Euclidean distance for computing affinity matrix.

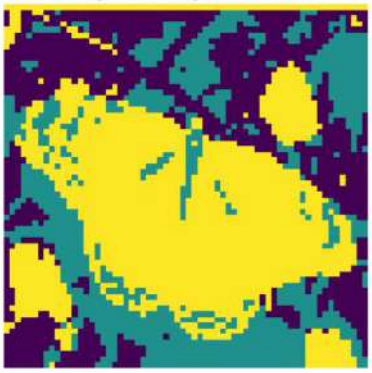
Calculating affinity matrix based on pairwise Euclidean distances between points .

A = np.exp(-4 * distance.cdist(X, X, metric='euclidean'))

Here is the output of the code:-

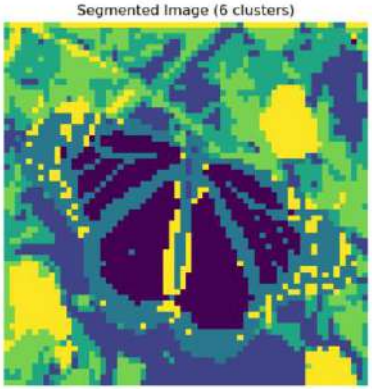
Original Image

Segmented Image (3 clusters)



Original Image

Segmented Image (6 clusters)

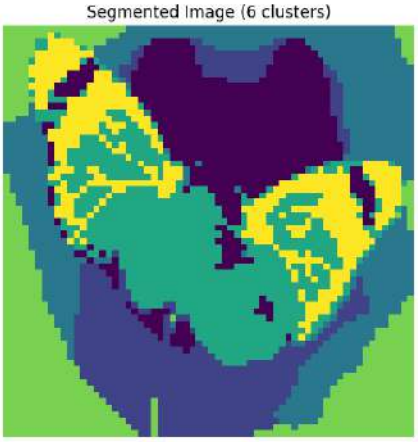


Original Image

Segmented Image (3 clusters)

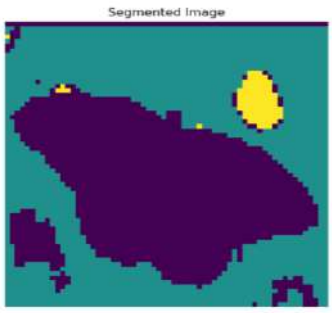
Original Image

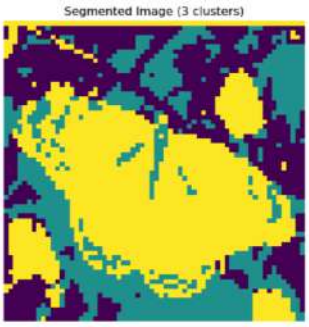
Segmented Image (6 clusters)

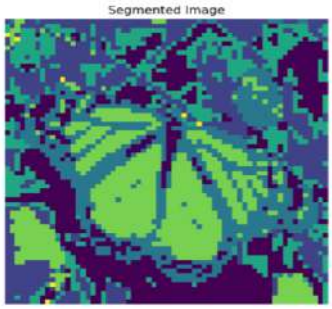


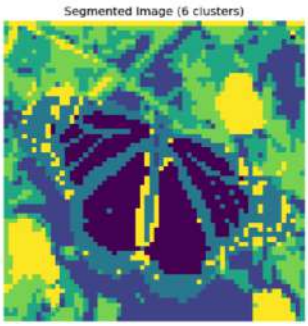
Comparison

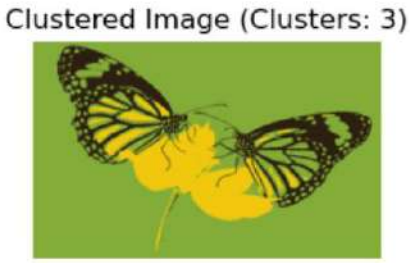
K-Means | Ratio Cut with RBF | Ratio Cut with Euclidean

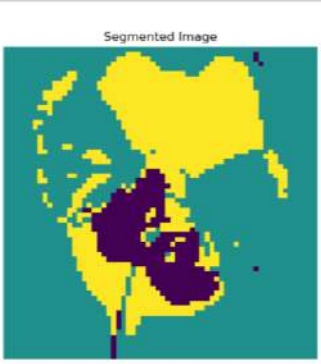


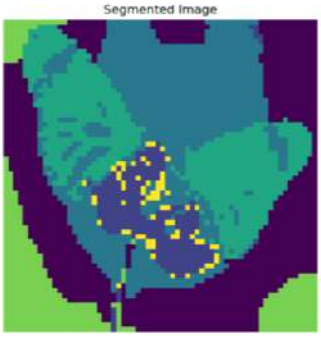












Ratio Cut is found to be more computation expensive.

K-means performed much better in terms of complexity, time and computation power.

References :-

K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks | by Imad Dabbura | Towards Data Science

sklearn.cluster.SpectralClustering — scikit-learn 1.4.2 documentation

Research on spectral clustering algorithms based on building different affinity matrix | IEEE Conference Publication | IEEE Xplore