VISION ASSISTED AUTONOMOUS DRIVING SYSTEM

18MHP109L - MAJOR PROJECT REPORT

Submitted by

VISHAL RAJEEV (RA2111038010036)
ALAN JOSEPH (RA2111038010032)
ARDRA MANOJ (RA2111038010002)
(Batch No.: MHP -04)

Under the guidance of

Dr. Vignesh S.M, M.E., Ph.D.
(Assistant Professor, Department of Mechatronics Engineering)

In Partial fulfilment for the degree

of

BACHELOR OF TECHNOLOGY

in

MECHATRONICS ENGINEERING with specialisation in Robotics

of

COLLEGE OF ENGINEERING & TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

May 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "***Vision Assisted Autonomous Driving System***" is the bonafide work of "**Vishal Rajeev (RA2111038010036), Alan Joseph (RA2111038010032), and Ardra Manoj (RA2111038010002)**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

9/5/25

Dr. Vignesh S.M, M.E., Ph.D.
GUIDE &
ASSISTANT PROFESSOR
Dept. Of Mechatronics Engineering

SIGNATURE

09/05/25

Dr. T. Muthuramalingam, M.E., Ph.D. PROFESSOR &HEAD OF THE DEPARTMENT
Dept. Of Mechatronics Engineering

V. Sujatha 6/5/25

Signature of the Internal Examiner

15/5/2025

Signature of the External Examiner

ABSTRACT

This project presents the development of a vision-assisted autonomous navigation system for a rover designed to operate in semi-structured outdoor environments. The system integrates a GPS-based global planner using the NEO-M9N module and the Google Maps Directions API, generating real-time waypoints from live position data. To enable dynamic path correction and obstacle avoidance, a ZED 2i stereo camera is used in combination with a custom-trained YOLOv5 model for object detection and LaneNet for lane following. Depth filtering is applied to detect obstacles to reduce false positives, while an emergency stop system based on raw depth ensures safety.

A hybrid control architecture is implemented using ROS Noetic (ROS 1) for actuator control and ROS 2 for vision modules, with a TCP socket bridge facilitating inter-process communication. The Arduino Nano controls a linear actuator for steering and a NEMA-23 stepper motor for throttle based on ROS commands. Feedback loops using potentiometer data allow steering estimation and correction in real time.

Additionally, a PyQt5-based GUI enables route visualization, ETA tracking, and destination selection through an interactive map. Real-time system states including live video, obstacle markers, and lane overlays are visualized using both RViz and OpenCV windows for debugging and analysis.

This modular and scalable system provides an end-to-end solution for autonomous rover navigation, combining map-based path planning, real-time perception, and robust actuation control. The proposed architecture serves as a practical framework for future extensions in urban robotics and autonomous delivery platforms.

ACKNOWLEDGEMENTS

It has been a great honor and privilege to undergo **B.Tech** in **MECHATRONICS** at **SRM Institute of Science and Technology**. We are very much thankful to the **Department of Mechatronics**, **SRM Institute of Science and Technology** for providing all facilities and support to meet our project requirements.

We would like to thank our Head of Department **Dr. T. Muthuramalingam, M.E., Ph.D.** and our project guide **Dr. Vignesh S.M, M.E., Ph.D.** who gave us a big helping hand during the entire project.

The success and final outcome of this project required a lot of guidance and assistance from many people, and we are extremely fortunate that we have got this all along the completion of our project work. Whatever we have done is only due to such guidance and assistance and we would like to thank them for their kind support.

VISHAL RAJEEV
ALAN JOSEPH
ARDRA MANOJ

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**iii**
	LIST OF FIGURES	**vii**
	LIST OF ABBREVIATIONS	**viii**
1	**INTRODUCTION**	**1**
	1.1 Background of the Project	1
	1.2 Challenges in Existing Systems	1
	1.3 Applications	2
	1.4 Need for the Project	2
	1.5 Methodology	2
	1.6 Organization of the Report	3
2	**LITERATURE SURVEY**	**4**
	2.1 Vision-Based Autonomous Navigation	4
	2.2 Vision-Based Autonomous System	4
	2.3 Adaptive Monte Carlo Localization in ROS	4
	2.4 GPS Navigation and Path Planning	5
	2.6 Integration of GPS and LiDAR	5
	2.7 Path Planning and Obstacle Avoidance	5
	2.8 Summary	5
3	**SYSTEM HARDWARE**	**7**
	3.1 Mechanical System	7
	3.2 Actuators	8
	3.3 Sensors	10
	3.4 Microcontroller and Computation Unit	11
	3.5 Power System	11

CHAPTER NO.	TITLE	PAGE NO.
4	**UTILIZATION OF SYSTEM COMPONENTS**	**12**
	4.1 Control System	12
	4.2 ZED Stereo Vision and Computer Vision Integration	15
	4.3 GPS Module (Neo-M9N) And GPS Based Path Planning	23
	4.4 Ros Based Navigation System	31
5	**SYSTEM INTEGRATION**	**40**
	5.1 Hybrid Global-Local Planning Architecture	40
	5.2 Socket Communication and Ros 1/Ros 2 Integration	41
	5.3 Obstacle Detection Logic and Response Strategy	42
	5.4 Decision Logic and Fail-Safe Obstacle Handling	43
	5.5 Real-Time Visualization and Monitoring	43
6	**RESULTS AND ANALYSIS**	**44**
	6.1 Introduction	44
	6.2 Testing Scenarios	44
	6.3 Performance Metrics Evaluated	45
	6.4 Summary of Results	47
	6.5 Observations and Insights	47
	6.6 Comparison to Objectives	48
7	**CONCLUSION AND FUTURE SCOPE**	**49**
	7.1 Conclusion	49
	7.2 Scope for Future Work	50

LIST OF ABBREVIATIONS

AGV	Autonomous Ground Vehicle
GPS	Global Positioning System
ROS	Robot Operating System
ZED	ZED Stereo Camera
YOLO	You Only Look Once
IMU	Inertial Measurement Unit
VIO	Visual Inertial Odometry
RViz	ROS Visualization
GUI	Graphical User Interface
API	Application Programming Interface
DWA	Dynamic Window Approach
AMCL	Adaptive Monte Carlo Localization

LIST OF FIGURES

Fig. 3.1 Initial Mechanical State of the Rover 7
Fig. 3.2 Steering Actuation Mechanical System 8
Fig. 3.3 The Entire Twist Throttle Coupling Mechanism and Control Circuit 9
Fig. 3.4 The Entire Electrical Circuit System of the Rover 9
Fig. 4.1 Lookup Table of Steering Angle in both Left and Right Tyres 14
Fig. 4.2 Lookup Table of Potentiometer w.r.t Actuator Travel 14
Fig. 4.3 The Lookup Table of Potentiometer Values and Steering Angle 15
Fig. 4.4 ZED 2i Mounted on Rover Frame 16
Fig. 4.5 Live Window Preview of Object Detection with Depth 17
Fig. 4.6 Visualization of the Object in Rviz 18
Fig. 4.7 Object Detection in Road Condition 19
Fig. 4.8 A Preview of the Lane detection System 20
Fig. 4.9 The UBLOX NEO m9n Module 24
Fig. 4.10 Path Planning Workflow 27
Fig. 4.11 GUI After Setting Destination 29
Fig. 4.12 GUI, Once Reached the Destination 29
Table 4.13 Observations 30
Fig. 4.14 GUI Logic 31
Fig. 4.15 Serial Connection 32
Table 4.16 ROS Topics Used 39
Fig. 5.1 Navigation Planner 40
Fig. 5.2 Object Detection Logic 42
Fig. 6.1: Recorded GPS Path vs Google Route 45
Fig. 6.2: Manually Annotated Objects vs Detected Objects 46
Table 6.3 Objectives and Progress 48

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND OF THE PROJECT

Autonomous ground vehicles (AGVs) represent a rapidly evolving field in robotics, aiming to navigate real-world environments with minimal human intervention. Among various approaches, vision-first navigation has emerged as a highly promising technique—enabling machines to interpret and interact with their surroundings using advanced camera systems, depth sensing, and machine learning. This project proposes the development of a vision-assisted autonomous rover, designed to operate within a campus-like semi-structured environment using real-time GPS navigation, stereo vision, and intelligent control systems.

Our rover is built with modularity in mind, combining a ZED 2i stereo camera, a u-blox NEO-M9N GPS module, ROS-based navigation architecture, and an Arduino-controlled actuation system. Together, these modules form an integrated platform capable of real-time path planning, lane detection, object recognition, and obstacle avoidance—delivering a scalable and adaptable solution for autonomous mobility across campuses, factories, or research parks.

1.2 CHALLENGES IN EXISTING SYSTEMS

Conventional autonomous navigation systems often rely solely on:

- Basic ultrasonic sensors or LiDAR for obstacle detection,
- GPS-only localization with limited precision,
- Predefined paths and hard-coded motion logic.

These systems typically face limitations such as:

- Poor adaptability to dynamic or GPS-denied environments,
- Inefficient handling of sudden obstacles or uncertain terrains,
- Limited perception capabilities, especially in detecting lanes, curves, and non-geometric obstacles,
- Inability to integrate high-level AI decision-making with low-level control.

Such shortcomings make traditional systems inadequate for **semi-structured, real-world use cases** where both **long-range planning and short-range perception** are crucial.

1.3 APPLICATIONS

The proposed vision-based autonomous navigation system has strong potential across multiple domains:

- Autonomous delivery rovers for campuses and gated communities,
- Patrol bots for indoor and outdoor surveillance,
- Smart warehouse logistics and automated guided vehicles (AGVs),
- Robotic platforms for disaster recovery, exploration, or search-and-rescue,
- Assistive robots in healthcare and service industries.

1.4 NEED FOR THE PROJECT

The increasing demand for **flexible and intelligent mobility platforms** necessitates the development of systems that can:

- Combine **global planning** with **local perception**,
- Operate in **partially mapped** or **unstructured** environments,
- React dynamically to **human presence** and **environmental changes**,
- Support **real-time visual feedback**, lane guidance, and obstacle detection.

This project bridges these needs by integrating high-level GPS navigation with low-level visual intelligence using deep learning, ROS middleware, and responsive actuation—delivering a truly **hybrid navigation system** suitable for scalable deployment.

1.5 METHODOLOGY

The project integrates a multi-sensor, multi-layer architecture comprising:

- **ZED 2i Stereo Camera**: Provides RGB-D output, depth estimation, and visual-inertial odometry.
- **NEO-M9N GPS Module**: Enables global position tracking and triggers waypoint generation using the Google Maps Directions API.

- **Custom-Trained YOLOv5**: Detects dynamic obstacles (e.g., pedestrians, vehicles) in real-time, verified using ZED's depth data.
- **LaneNet**: Performs advanced lane detection even in curved or unmarked environments.
- **ROS 1 + ROS 2 Bridge**: Manages sensor fusion, planning, and actuation across distributed nodes.
- **Arduino Actuation Layer**: Controls throttle and steering using a linear actuator and stepper motor via rosserial.

The core flow includes:

1. Fixing initial position via GPS,
2. Generating waypoints through Google Maps API,
3. Real-time lane and object detection via vision modules,
4. Hybrid decision-making (global planner + local planner),
5. Commanding actuators with feedback loops and safety overrides.

1.6 ORGANIZATION OF THE REPORT

- Chapter 1 introduces the problem statement, challenges, goals, and methodology.
- Chapter 2 presents a literature review on autonomous navigation systems and perception frameworks.
- Chapter 3 describes the mechanical and electrical hardware design of the system.
- Chapter 4 explains the utilization of individual system components including sensors, controllers, and software nodes.
- Chapter 5 details the hybrid planning system and integration strategy.
- Chapter 6 outlines testing methodology, performance metrics, and result analysis.
- Chapter 7 concludes the work and highlights the scope for future improvements.

CHAPTER 2

LITERATURE SURVEY

The development of autonomous navigation systems has rapidly progressed due to advancements in computer vision, deep learning, and real-time sensor fusion. Traditional GPS-based navigation, while effective in structured environments, struggles in dynamic or semi-structured conditions like college campuses. This literature review explores existing approaches in autonomous ground vehicles (AGVs), stereo vision, object detection, lane detection, and robotic frameworks such as ROS, which collectively inform the basis of the proposed system.

2.1 VISION-BASED AUTONOMOUS NAVIGATION

Vision-based navigation leverages visual sensors to perceive and interpret the environment, facilitating autonomous movement. Rankin et al. (2009) explored terrain mapping using stereo vision for off-road autonomous navigation, demonstrating the potential of passive perception in complex terrains. Similarly, the Jet Propulsion Laboratory (JPL) developed a robust visual navigation and mapping system, emphasizing the integration of visual data for autonomous landing and exploration. These studies underscore the efficacy of visual sensors in navigation but also reveal challenges in processing and interpreting visual data in real-time, especially in dynamic environments.

2.2 VISION-BASED AUTONOMOUS SYSTEMS

Vision-based systems rely on cameras to interpret the environment for navigation and decision-making. The ZED 2i stereo camera, as discussed in Stereolabs documentation [1], provides high-resolution RGB-D data and visual-inertial odometry (VIO), allowing depth estimation and 3D mapping. Such systems are essential for enabling real-time obstacle detection, lane following, and environment reconstruction without dependency on expensive LiDAR systems.

2.3 ADAPTIVE MONTE CARLO LOCALIZATION (AMCL) IN ROS

The YOLO (You Only Look Once) family of models has proven effective in real-time object detection due to its single-shot, grid-based detection architecture. Ultralytics' YOLOv5 [2] offers enhanced accuracy and speed, making it suitable for embedded and mobile robotics.

Custom training on domain-specific datasets further improves detection of relevant obstacles (e.g., pedestrians, roadblocks), especially when paired with depth filtering for spatial validation.

2.4 GPS NAVIGATION AND PATH PLANNING

The u-blox NEO-M9N GPS module, reviewed in [7], offers high update rates and multi-band GNSS reception, making it effective for outdoor applications with minimal drift. Coupled with Google Maps Directions API [5], global route planning becomes dynamic, context-aware, and user-friendly. Previous works such as Neil Nie's Self-Driving Golf Cart [10] illustrate how low-cost GPS combined with visual feedback can be scaled for intelligent route-following.

2.6 INTEGRATION OF GPS AND LiDAR DATA FOR AUTONOMOUS NAVIGATION

Combining GPS and LiDAR data enhances localization and mapping capabilities, particularly in unstructured environments. Patoliya et al. (2022) designed a ROS-based vehicular system integrating GPS and LiDAR, utilizing an Inertial Measurement Unit (IMU) to improve LiDAR registration. This integration addresses the limitations of individual sensors, offering a more robust solution for autonomous navigation. However, challenges remain in ensuring seamless data fusion and managing discrepancies between sensor inputs.

2.7 PATH PLANNING AND OBSTACLE AVOIDANCE TECHNIQUES

Effective path planning and obstacle avoidance are critical for autonomous navigation. Wang (2021) implemented SLAM and path planning algorithms, including the A* and Dynamic Window Approach (DWA), to achieve real-time obstacle avoidance and navigation. These algorithms enable robots to navigate complex environments by dynamically adjusting paths in response to obstacles. Nonetheless, their performance is contingent on accurate environmental perception and timely data processing.

2.8 SUMMARY

The literature indicates a clear shift from traditional GPS-only systems toward **hybrid architectures** that integrate **vision, depth, GPS, and real-time decision-making**. Tools like YOLOv5, LaneNet, ROS, and ZED SDKs provide a robust ecosystem for designing campus-scale autonomous systems. However, many works still treat visual and GPS components in

isolation. The proposed project builds upon these foundations to create a unified, hybrid navigation stack with **fail-safe obstacle handling**, **turn anticipation**, and **real-time actuator control**, aiming to push the boundary of affordable and scalable autonomous mobility.

CHAPTER 3

SYSTEM HARDWARE

In this chapter, the hardware systems involved in the autonomous navigation rover are described. These components enable the system to perceive its environment, plan paths, and execute motion accurately based on GPS and visual inputs. The hardware is modular and built from off-the-shelf components optimized for outdoor navigation.

3.1 MECHANICAL SYSTEM

The rover is based on a rigid frame platform with a four-bar steering mechanism and rear-wheel drive using a chain drive system. The chassis is custom-welded to house major components such as the battery, actuators, and perception system. A dedicated mount is installed to elevate the ZED stereo camera to ensure optimal forward field of view.

Fig. 3.1 Initial Mechanical State of the Rover

3.2 ACTUATORS

3.2.1 Steering Actuation System

The front wheels are connected via a four-bar Ackermann steering mechanism. A linear actuator is used to control the steering angle by extending and retracting the linkage. The actuator operates at 12V and provides consistent thrust of 1500N at ~7mm/s speed (Linear Actuator Stroke Length 200MM, 7mm/S, 1500N, 12V)

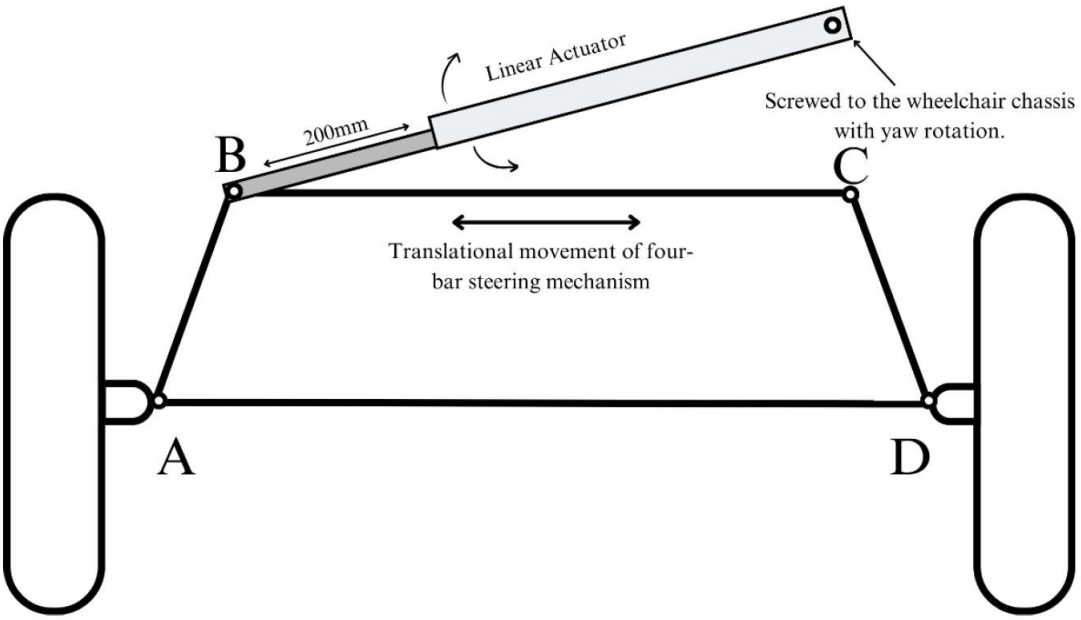

Fig. 3.2 Steering Actuation Mechanical System

3.2.2 Throttle Control System

Throttle control is achieved through a stepper motor (NEMA-23) that physically twists a handle throttle connected to a DC motor. The twist mechanism is coupled using a custom 3D printed mount fixed to a wooden platform for vibration dampening. The system operates in open-loop mode, with the stepper's steps mapped to specific speeds using a calibrated lookup table.

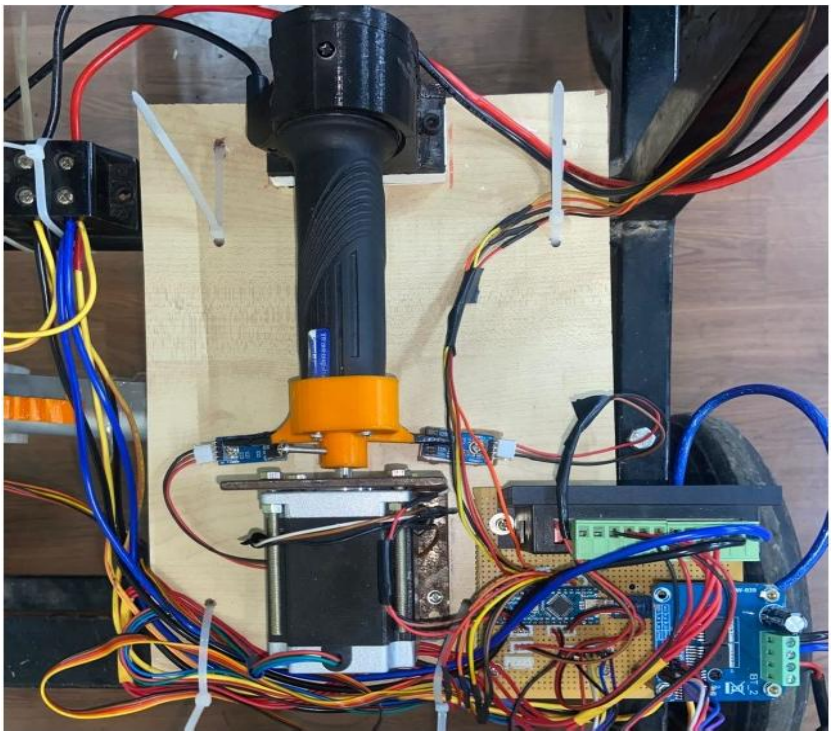

Fig. 3.3 The Entire Twist Throttle Coupling Mechanism and Control Circuit

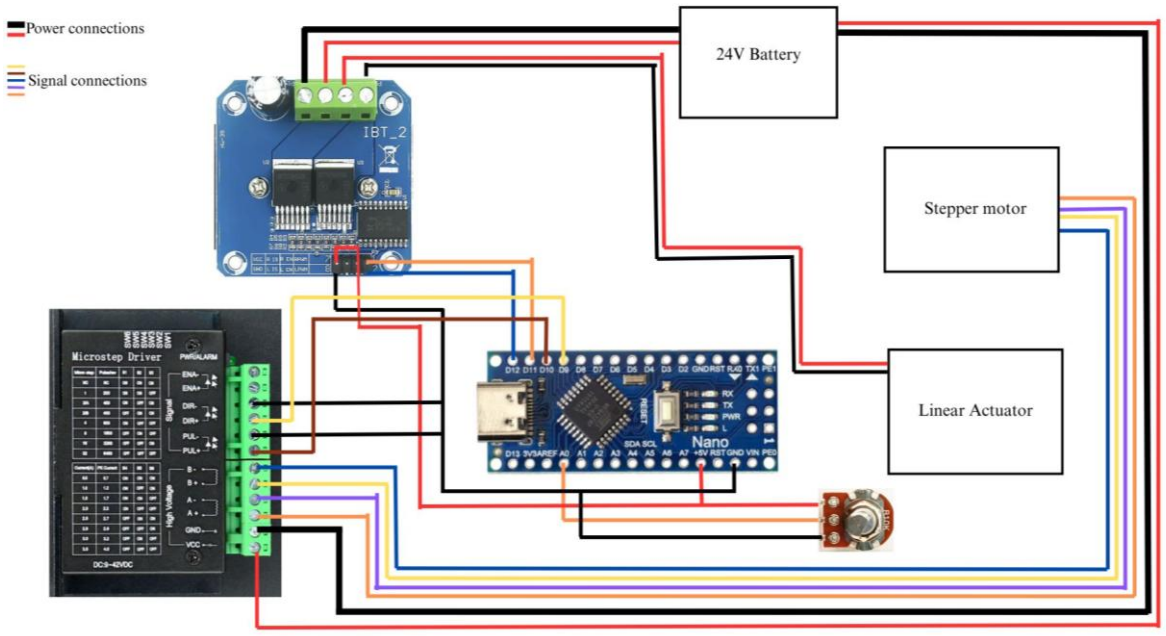

Fig. 3.4 The Entire Electrical Circuit System of the Rover

3.2.2 Wheel Actuator and Driver

The Rover's wheels are powered by a high-torque **brushed DC motor (Model: MY1016Z2)**, commonly used in e-bikes and operating at 24V. It is mounted perpendicular to the driveshaft, with power transmission via a chain drive. The motor mount is adjustable to minimize chain slack. A dedicated motor controller, powered by a 24VDC supply, manages motor operation and includes a manual twist throttle for acceleration and a brake lever for deceleration.

3.3 SENSORS

3.3.1 GPS Module

A U-blox NEO-M9N GPS receiver is mounted on the rover and provides real-time position data via UART serial interface. It is used for global navigation and route planning via Google Maps Directions API.

3.3.2 ZED Stereo Camera (ZED 2i)

The primary perception sensor is the ZED 2i stereo camera. It captures:

- RGB images
- Depth maps
- IMU-based orientation data

This camera is used for:

- Obstacle detection via a YOLOv5 custom model
- Depth filtering for dynamic throttle interruption
- Generating visual feedback in RViz and GUI tools

3.3.3 Potentiometer Feedback

Steering angle is retrieved using a rotary potentiometer attached to a custom rack-and-pinion system. The analog voltage is read via Arduino Nano to estimate real-time steering angle feedback for closed-loop control.

3.4 MICROCONTROLLER AND COMPUTATION UNIT

- Arduino Nano

Used for reading potentiometer values and publishing throttle/steering commands via ROS Serial.

- Onboard Processor (Laptop: Asus ROG STRIX, AMD Ryzen 7 & RTX 3050 Ti)

Runs the ROS-based navigation stack, including:

- YOLO-based object detection on camera stream
- Google Maps API waypoint planner
- Kinematic model translator (converts GPS and heading into steering and throttle commands).

3.5 POWER SYSTEM

The power system consists of **dual 12V lead-acid batteries** connected in series to supply 24V. Throttle control is achieved using a TB6600 stepper motor driver that actuates a twist throttle via a NEMA-23 motor. Steering is handled by a linear actuator driven by a BTS7960 H-bridge motor driver, enabling accurate directional control based on input commands from the onboard control system.

CHAPTER 4
UTILIZATION OF SYSTEM COMPONENTS

This chapter explains in detail how each component of the autonomous rover was utilized to achieve the goal of GPS-guided, vision-assisted autonomous navigation. The design integrates mechanical, electrical, sensing, and computational modules to form a coherent and responsive mobility system. Inspired by earlier autonomous wheelchair systems, this implementation adapts the approach to outdoor rover-based navigation with GPS path planning and stereo vision for obstacle detection.

4.1 CONTROL SYSTEM

The control system manages the physical motion of the rover including forward motion, steering control, and deceleration. It is centered on the **Arduino Nano** microcontroller, which interfaces with the actuator systems and executes commands sent from the onboard ROS computer using **ROS Serial communication**.

4.1.1 Throttle System

The throttle control is implemented using a **NEMA-23 stepper motor**, mechanically coupled to a manual twist throttle. This throttle interfaces with a **24V 350W motor controller**, which in turn regulates the **MY1016Z2 24V brushed DC motor**, the rover's primary drive unit. The stepper motor functions in an open-loop configuration, where the number of steps corresponds to the throttle rotation. This step count is mapped to throttle percentage based on a calibrated step-to-throttle relationship, allowing forward motion control.

4.1.2 Steering System

Steering is handled using a linear actuator connected to a custom-designed four-bar steering linkage. The actuator executes left and right steering commands by extending and retracting, which mechanically shifts the steering arms. The steering range is constrained within calibrated limits of **+22° to -22°**, representing full right and left turns, respectively.

A potentiometer-based feedback system is employed to track the actuator's position. The analog voltage from the potentiometer is pre-mapped to steering angles based on manual

calibration, establishing a reliable voltage-to-angle relationship. This feedback is processed by the Arduino, allowing the system to interpret and maintain the current steering angle relative to the desired orientation specified via ROS commands.

4.1.3 Feedback and Safety

The rover relies on a minimal but effective set of feedback mechanisms to ensure safe and accurate motion control. These feedback systems monitor actuator position, environmental hazards, and vehicle orientation to dynamically adjust behavior.

- Potentiometer Feedback: A rotary potentiometer mounted to the steering linkage provides real-time analog voltage readings. These are mapped to steering angles through manual calibration, enabling the microcontroller to estimate the current steering position and maintain alignment with ROS-issued turning commands.
- Camera-Based Obstacle Detection: The ZED stereo camera, in combination with a custom YOLOv5 object detection model, identifies objects in the rover's path. If an obstacle is detected within a critical depth threshold, a signal is sent to halt motion, acting as an emergency override system.
- IMU Data Integration: Orientation data from the ZED camera's built-in IMU is used to support heading corrections and orientation tracking, particularly useful for maintaining directional accuracy during longer path executions or environmental disturbances.

The system halts motion upon receiving a stop command either due to reaching the destination, detecting an obstacle, or exceeding predefined steering limits.

Adding a wheel encoder would provide precise odometry data, allowing closed-loop velocity control and position estimation. This would significantly improve the accuracy of distance tracking and enable more advanced control strategies like PID or MPC-based trajectory following.

Wheelbase (L) = 65cm
Track width (W) = 71cm (Wf), 65cm (Wr) (Wf- width in front, Wr- width in rear)
Steering Arm length (S) = 17 cm
Tie-rod length (T) = 44 cm
Steering ratio (Sr) = Actuator Travel (mm)/Wheel steering angle (°)

$Sr = \Delta$Actuator Travel (mm)/ $\Delta\theta$wheel

Sr mean for left and right wheels,
Sr[L] = 0.328cm/° (for left)
Sr[R] = 0.263cm/° (for right)

Max steering angle = ± 22° (0.38397 rad)
Turning Radius (R),

$R = L/\tan(\theta in)$

$= 65/\tan(0.38397)$ (θin- 0.38397)

$= 65/0.40289$

R = 160.88cm (1.92m)

For 360° turn, Turning Circle (D),

$D = 2 \times R + W/2$

$= 2 \times 160.88 + 71/2$

$= 2 \times 192.38$

D = 392.76cm (3.84m)

Actuator travel	Left wheel θ	Right wheel θ
-5cm (max left travel)	-15	-22
-4cm	-11	-15
-2cm	-4	-7
0cm (origin)	0	0
2cm	11	10
4cm	19	20
5cm(max right travel)	21	22

Fig. 4.1 Lookup Table of Steering Angle in both Left and Right Tyres

Actuator travel (wheel travel is inverse)	potentiometer value
-5cm (max left travel)(retract)	4.80V
0cm (origin)(center)	2.55V
5cm(max right travel)(extend)	0.20V

Fig. 4.2 Lookup Table of Potentiometer w.r.t Actuator Travel

potentiometer value	steering angle
4.80V	22
2.55V	0
0.20V	-22

Fig. 4.3 The Lookup Table of Potentiometer Values and the Steering Angle

4.2 ZED STEREO VISION AND COMPUTER VISION INTEGRATION

4.2.1 Introduction

This chapter elaborates the integration of the **ZED 2i Stereo Camera** with a deep learning-based object detection and lane detection system to achieve robust real-time perception for the autonomous rover. The combination of **stereo vision**, **depth sensing**, **lane following**, and **machine learning** enables the system to perceive the environment dynamically, recognize obstacles, detect road structures, and ensure safe navigation. This perception module is vital in providing real-time situational awareness to complement GPS-based global navigation and to enhance the autonomous capabilities of the rover in semi-structured environments like a campus.

4.2.2 ZED 2i Stereo Camera Overview

The **ZED 2i Stereo Camera**, developed by StereoLabs, serves as the primary perception sensor. Mounted on an elevated custom bracket and facing forward, it maximizes the horizontal and vertical field of view essential for early obstacle detection and lane visualization. The camera offers synchronized streams:

- **RGB Image Stream**: Standard color images.
- **Depth Map**: Disparity-based depth calculation, giving 3D information.
- **Visual-Inertial Odometry (VIO)**: Integrating IMU data with visual features to estimate motion.

Fig. 4.4 ZED 2i Mounted on the Rover Frame

By providing RGB-D and VIO data, the ZED 2i allows the system to perform multi-modal perception tasks essential for safe navigation.

4.2.3 Vision Pipeline Architecture

The vision system follows a modular, layered architecture to ensure efficiency and flexibility. The flow includes:

- **Step 1: Data Acquisition**: Real-time RGB and depth frames are subscribed via ROS 2 topics. High frame rates and low latency are maintained.
- **Step 2: Object Detection Using YOLOv5**: Lightweight deep learning models detect people, vehicles, and other moving/static obstacles.
- **Step 3: Depth Extraction and Validation**: For every detected object, a corresponding depth patch is extracted to validate proximity.
- **Step 4: Lane Detection Using LaneNet**: An advanced neural network is applied to RGB frames to detect lane markings and road boundaries.
- **Step 5: Control Signal Generation**: Depending on detections, "stop", "start", or "turn" signals are generated and sent to the rover control layer.

This architecture modularizes perception tasks while ensuring real-time operability.

4.2.4 YOLOv5-Based Custom Object Detection

A custom YOLOv5 object detection model was trained specifically for our project using a curated dataset consisting of relevant obstacles such as vehicles, pedestrians, and barriers encountered in semi-structured environments. The dataset was annotated using Roboflow and trained over 150 epochs with augmentation techniques to improve generalization.

Once trained, the model was integrated into the vision pipeline to process frames from the ZED 2i stereo camera in real-time. Detected objects were filtered based on depth (< 2 meters) and lane positioning to determine if they posed an immediate threat. This filtered output was used to trigger motion control decisions such as stopping or decelerating the rover during autonomous navigation.

- Real-time inference on CPU or GPU.
- Detecting multiple classes simultaneously.
- Processing frames up to 30 FPS with optimized models.

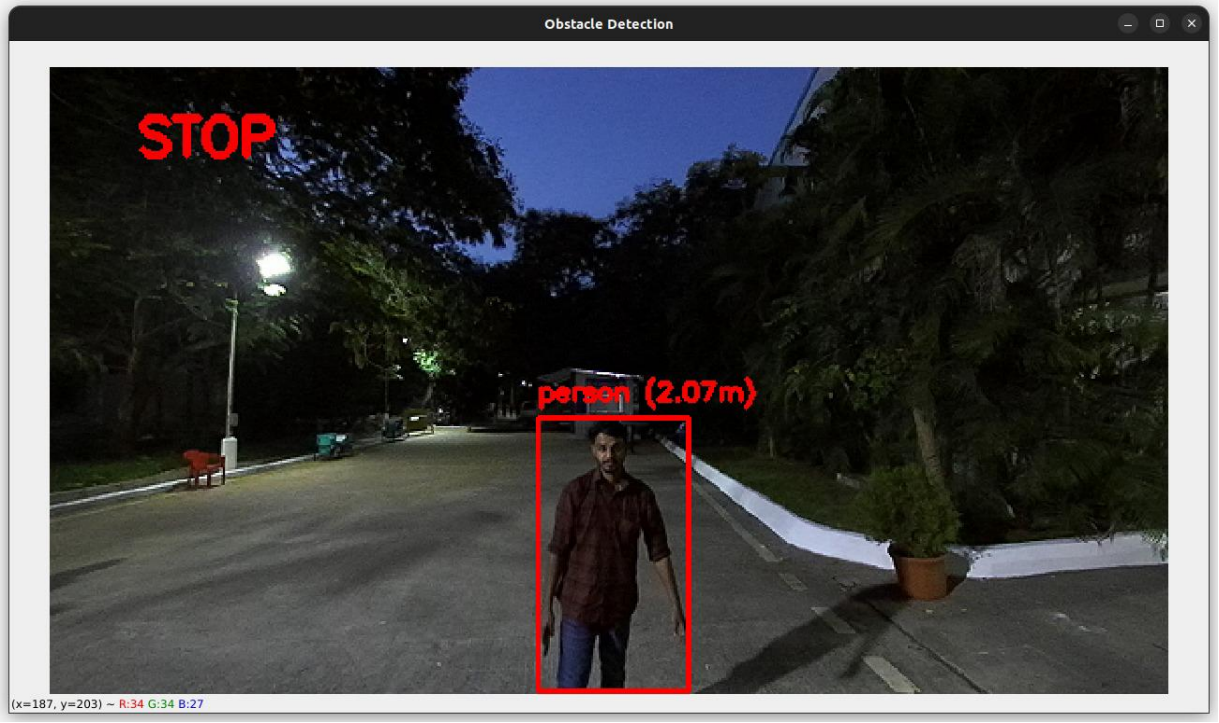

Fig. 4.5 Live Window Preview of Object Detection with Depth

Key implementation steps:

self.model = YOLO("yolov5n.pt")

self.device = 'cuda' if torch.cuda.is_available() else 'cpu'

self.model.to(self.device)

Frames are resized and normalized before being passed to the model. Output detections include:

- Bounding box coordinates.
- Class labels.
- Confidence scores.

This enables the system to efficiently detect multiple potential obstacles on the fly.

4.2.5 Depth Map Alignment and Obstacle Validation

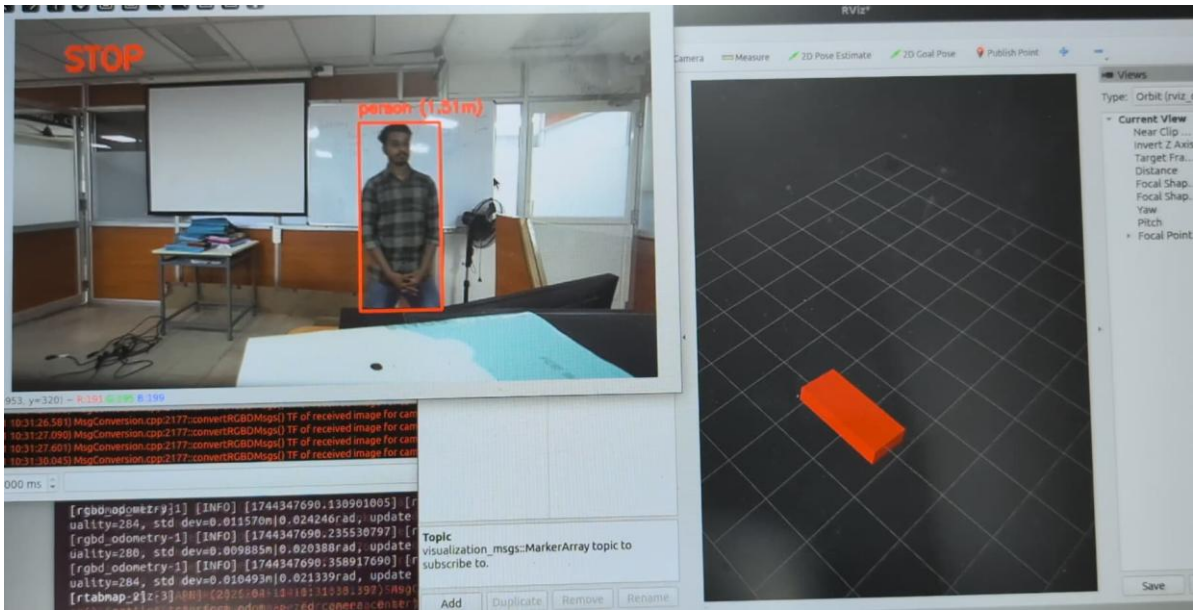

Fig. 4.6 Visualisation of the Object in Rviz2

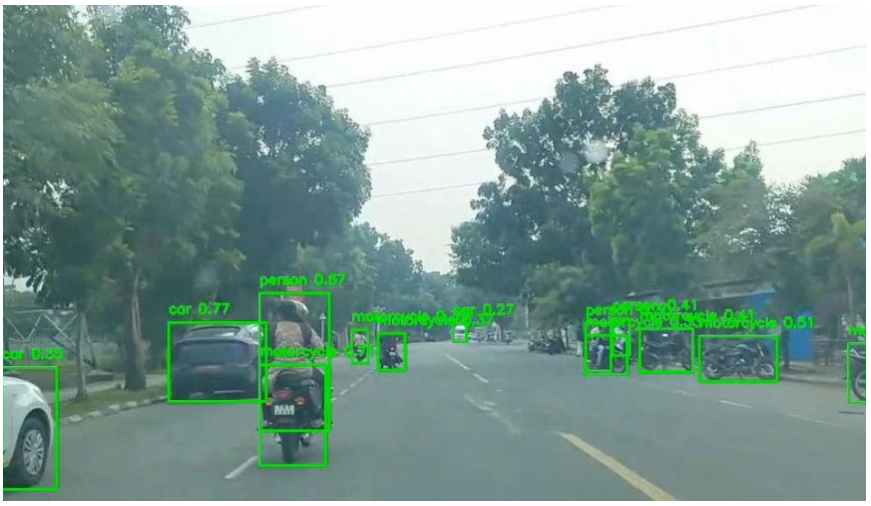

Fig. 4.7 Object Detection in Road Condition

Depth data is leveraged to validate each detected object. For each bounding box:

region = depth_map[y:y+h, x:x+w]

avg_depth = np.mean(region[region > 0])

If the average depth falls below a certain threshold (e.g., 1.8 meters), the object is flagged as an immediate obstacle.

This two-stage (visual + depth) verification reduces:

- False positives.
- Reacting to irrelevant distant objects.

The use of depth filtering significantly improves obstacle management reliability.

4.2.6 Visualization with RViz

Visualization is critical for development and testing. RViz tools are employed for:

- Live visualization of the ZED depth point cloud.
- Overlaying bounding boxes from object detection.

- Displaying lane markings detected by LaneNet.

Example RViz topics:

- `/zed/zed_node/point_cloud/cloud_registered`
- `/detected_objects`
- `/lane_detection/overlay`

RViz allows visual confirmation of the correct functioning of perception modules during real-world trials.

4.2.7 Advanced Lane Detection

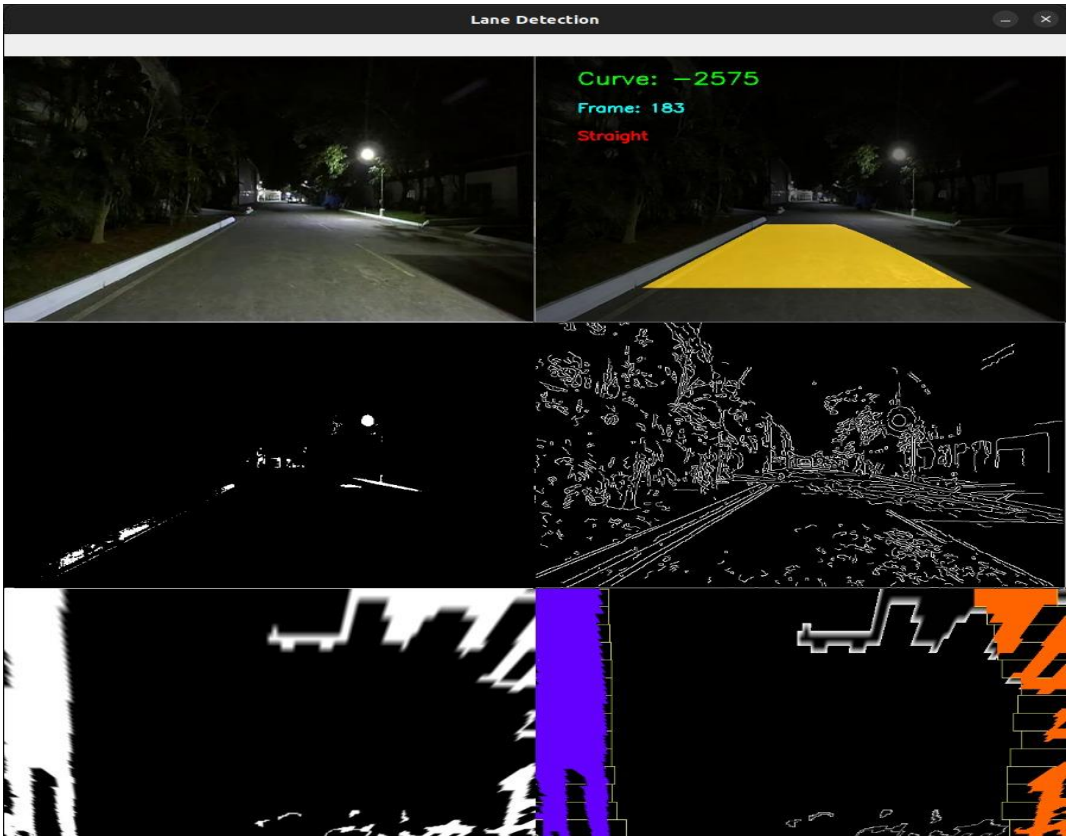

Fig. 4.8 A Preview of the Lane Detection System

The system includes an advanced **LaneNet-based lane detection** module. This is crucial for:

- Detecting curved paths.
- Ensuring the rover can maintain its lane during dynamic GPS-driven motion.

Process steps:

- Preprocess image with resizing and thresholding.
- Apply a perspective transform to generate a bird's-eye view.
- Run LaneNet for lane segmentation.
- Fit second-order polynomials to detected lanes.

Code snippets:

warped = perspective_warp(image)

lane_mask = lanenet_model.predict(warped)

left_fit, right_fit = fit_polynomial(lane_mask)

The fitted curves provide:

- Real-time steering targets.
- Turn prediction by analyzing lane curvature.

LaneNet is highly useful in urban structured environments where GPS drift could otherwise lead to incorrect steering decisions.

4.2.8 Lane Following and Turn Management

When upcoming turns are detected:

- The global planner issues turn alerts.
- The local lane planner anticipates the curve based on lane curvature.
- Steering angles are corrected proactively.

This multi-layered planning improves reliability over purely GPS-IMU fused systems.

4.2.9 Depth-Based Object Detection and Reaction

The dual-layer detection system works as follows:

- **Semantic Detection**: YOLOv5 identifies what object is in the scene.

- **Depth Confirmation**: ZED depth map measures distance.

Obstacle Reaction Logic:

```
if avg_distance < threshold:
    send_command("stop")
else:
    send_command("start")
```

This ensures:

- Safe stopping if an object approaches.
- Smooth resumption when clear.
- Fail-safe behavior is integrated with GPS planning.

4.2.10 System Strengths and Limitations

Strengths:

- Real-time RGB-D perception.
- Depth-confirmed obstacle detection.
- Lane following with turn anticipation.
- Modular separation of vision tasks and actuation control.

Limitations:

- Performance affected in low-light or adverse weather.
- Lane detection accuracy decreases in worn-out or missing lanes.
- GPS drift may still occur without RTK correction.

4.2.11 Expansion Scope

Enhancements proposed:

- **Fusion of lane detection and depth** into a unified local planner.
- **Obstacle prioritization** (e.g., emergency brake for humans, detour for cones).

- **Kalman Filtering** for smoother detection and prediction.
- **Dynamic replanning** if lanes and GPS disagree.

4.2.12 Summary

The ZED stereo vision and computer vision stack establish a strong, intelligent perception module for the autonomous rover. It enables simultaneous lane following, obstacle avoidance, and dynamic path management. These capabilities elevate the vehicle from static waypoint navigation to dynamic, responsive motion planning critical for real-world deployment. As such, this subsystem acts as a vital bridge between high-level planning and low-level actuation, ensuring safe, intelligent, and context-aware mobility.

4.3 GPS MODULE (NEO-M9N) AND GPS BASED PATH PLANNING

The GPS module is responsible for global localisation and route tracking. A **u-blox NEO-M9N** module is used, capable of high-accuracy readings with an update rate suitable for mobile navigation.

4.3.1 Initialization and Position Fixing

Upon startup, the GPS module communicates via UART and fixes the current position, which is used as the **starting point** for route planning. Then it begins transmitting standard NMEA sentences. The $GNRMC sentence contains essential navigation data such as time, position (latitude and longitude), speed, and fix validity. These values are then converted from DMM (degrees and decimal minutes) to decimal degrees using:

Decimal Degrees = Degrees + (Minutes / 60)

Example:

NMEA Latitude: 1249.22736 N → 12.820456°

NMEA Longitude: 08002.39764 E → 80.039961°

This converted coordinate represents the current position of the vehicle and is treated as the starting point for route generation. The system waits for a valid GPS fix and disregards any invalid or incomplete NMEA strings to ensure localisation accuracy.

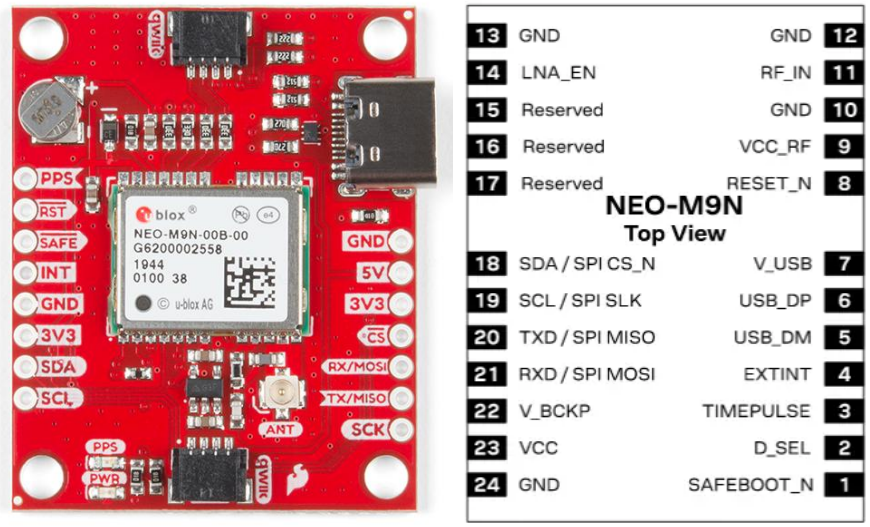

Fig. 4.9 The UBLOX NEO m9n Module

4.3.2 Path Planning with Google API

After the initial position is fixed, the route planning system formulates a request to the Google Maps Directions API using the current GPS location as the origin and a pre-configured or user-selected destination.

The request format is as follows: https://maps.googleapis.com/maps/api/directions/json?origin=LAT1,LON1&destination=LAT2,LON2&key=API_KEY

The API returns a JSON response containing an encoded polyline, which represents the optimal path between the origin and destination as a compressed string of coordinates. This polyline is decoded using Python's polyline module into a list of (latitude, longitude) tuples.

To ensure path smoothness and to maintain consistent motion control, the decoded route is interpolated such that consecutive waypoints are spaced at approximately 5-meter intervals. This is achieved using the geopy.distance.geodesic function. The result is a high-resolution

path with uniformly spaced waypoints ideal for time-based or feedback-based throttle and steering control systems.

4.3.3 Path Smoothing and Tuning

Raw GPS routes are often irregular due to road curvature, network resolution, and point density. To address this, the following post-processing steps are performed:

- **Spacing Filter:** Ensures that no two consecutive waypoints are closer than 5 metres. This prevents the rover from executing unnecessary micro-adjustments, which could lead to oscillatory actuator's behaviour.

- **Curvature Averaging:** A sliding window of 3–5 waypoints is used to calculate and average the heading angle between segments. This results in a smoothed turning trajectory and improved control signal predictability.

- **Waypoint Reach Threshold:** A proximity threshold of 1.5 to 2 metres is enforced around each waypoint. Once the rover enters this radius, it progresses to the next waypoint, ensuring logical flow and path adherence without requiring exact matches.

These enhancements contribute to actuator stability, reduce controller oscillations, and provide a more natural motion profile during navigation.

4.3.4 Real-time Execution

During autonomous operation, the vehicle receives live GPS coordinates at a frequency of 1 Hz. These coordinates are continuously compared with the destination location to assess progress. The geodesic distance between the current position and the destination is calculated using the Haversine formula:

The Haversine Formula:

The Haversine formula calculates the shortest distance over the Earth's surface between two points:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

Where:

- ϕ = latitude in radians
- λ = longitude in radians
- R = Earth's radius (6371 km)
- d = distance between points in km

This ensures precision and accuracy in distance calculation, vital for correct interpolation.

This method provides a mathematically accurate estimate of distance over the Earth's surface. Based on this remaining distance, a throttle time duration is computed. This time-duration logic allows the rover to move toward the destination even without feedback-based speed control, using pre-characterised time-to-distance mappings.

Gives the great circle distance from 2 points (longitude and latitude), which is useful for navigation. This offers precision and accuracy in distance computation, which is important for correct interpolation.

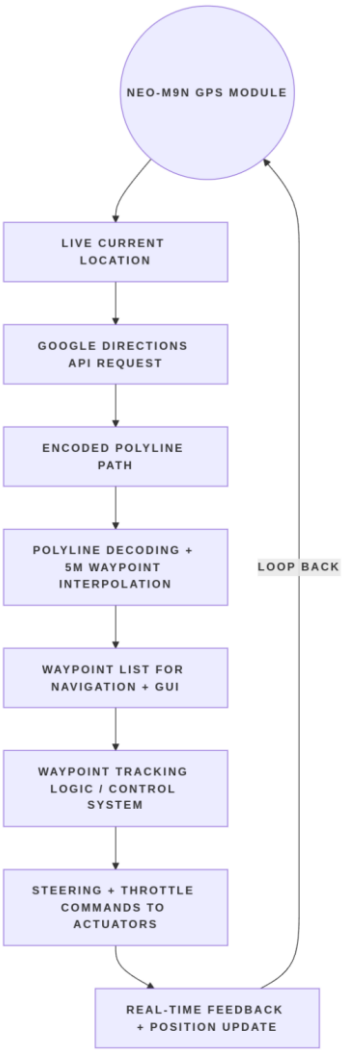

Fig. 4.10 Path Planning Workflow

4.3.5 GUI SUPPORT

A **PyQt5-based interface** was developed to simulate GPS navigation, visualise routes on a map using **Folium**. This interface supports real-time route fetching, simulation, and export of navigated paths.

This multi-layered integration of sensors, processors, actuators, and software allows the rover to operate autonomously in semi-structured environments, execute planned paths, and adapt its behavior in response to dynamic obstacles.

4.3.5.1 Real-time Location Tracking and Display

The GUI embeds an interactive HTML map generated using the Folium library (built on Leaflet.js). At each update:

- The current GPS position is plotted as a moving marker.
- A destination marker is shown in a different colour.
- A polyline connects all interpolated waypoints.

The GUI displays updated labels for current coordinates, distance remaining, and ETA.
The Folium map is regenerated periodically and saved as an HTML file, which is then loaded into a PyQt5 QWebEngineView widget.

4.3.5.2 Interactive Features

One of the key features of the GUI is its support for interactive destination setting:

- When the user clicks on a location on the map, embedded JavaScript code captures the latitude and longitude of the clicked point.
- These coordinates are passed to the Python backend using QWebChannel, which bridges JavaScript and Python.
- The backend replaces the existing destination with the new one, requests a new route via Google API, decodes and interpolates it, and refreshes the GUI.

This mechanism allows real-time rerouting without the need for a system restart, enabling flexible testing and deployment.

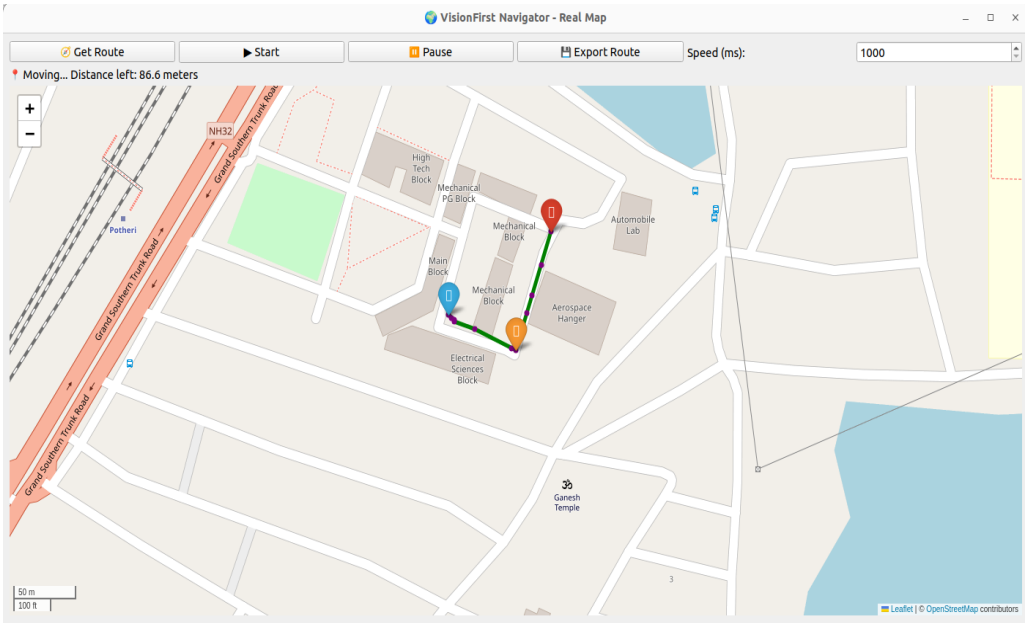

Fig. 4.11 GUI after Setting Destination

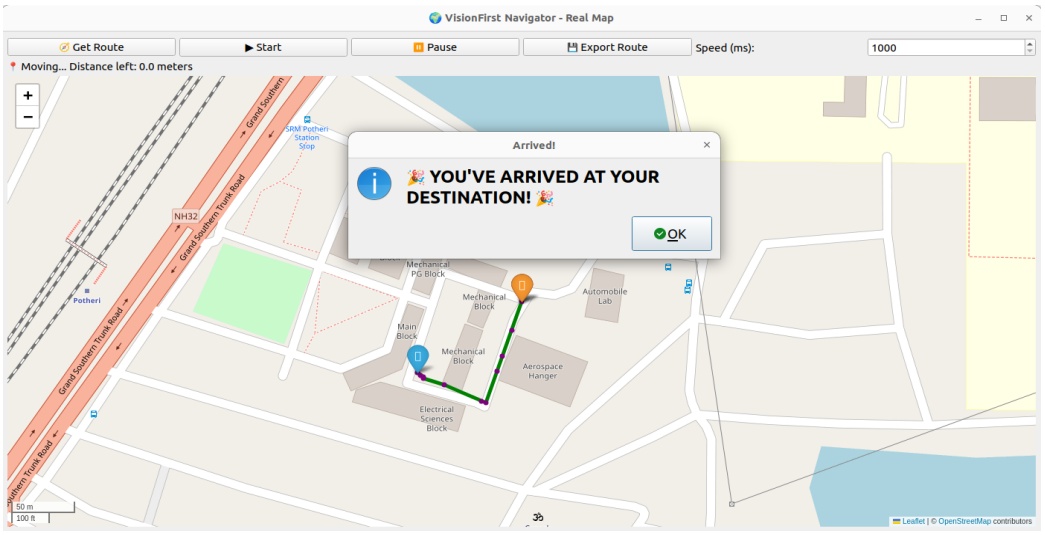

Fig. 4.12 GUI, Once Reached the Destination

4.3.5.3 PyQt5 Application Design

The GUI layout includes:

- A map window displaying the live Folium map.
- Label fields showing current latitude, longitude, ETA, and distance.
- Pop-up alerts when the vehicle nears the destination (within a 3-metre radius).
- A QTimer object trigger periodic GUI refresh without blocking the event loop.

This modular layout ensures the GUI remains lightweight and responsive while maintaining consistent visual feedback.

Experimental Observations:

Metric	**Result**
GPS update rate	1 Hz (live NMEA feed via UART)
GUI map refresh rate	3 seconds (folium HTML regeneration)
Distance calculation method	Haversine formula with geodesic accuracy
GUI rendering framework	PyQt5 with embedded QWebEngineView
Map visualization backend	folium using Leaflet.js
Interactivity	HTML map click → JavaScript → QWebChannel → Python
Destination update latency	< 200 ms
Arrival detection radius	3 meters
GUI response time	< 100 ms per update cycle
Path smoothing	Waypoint spacing = 5 m, curvature averaging enabled

Table 4.13 Observations

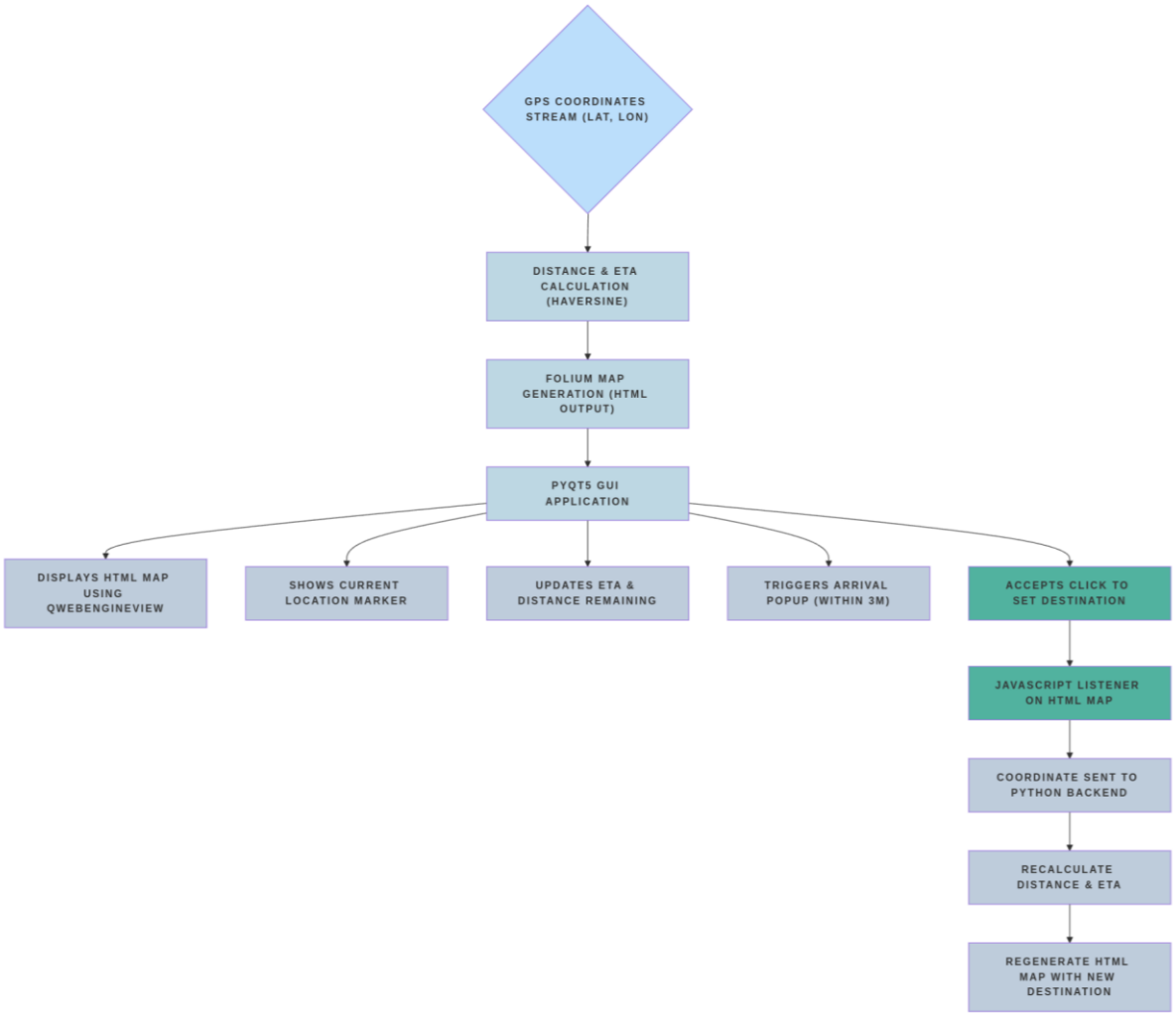

Fig. 4.14 GUI Logic

4.4 ROS-BASED NAVIGATION SYSTEM

4.4.1 ROS Serial to Arduino Control System

To enable real-time actuation of steering and throttle systems based on high-level ROS commands, the system leverages the rosserial_arduino package. This allows a microcontroller—in our case, an **Arduino Nano**—to subscribe to ROS topics and execute hardware control logic seamlessly. It bridges the gap between high-level path planning and low-level actuator execution through serial communication.

The Arduino acts as a lightweight interpreter that listens to string-based control messages and maps them to mechanical movements of the **steering linear actuator** and **stepper motor**

throttle system. Additionally, it continuously publishes **potentiometer feedback** and **steering angle** values, enabling real-time visualization and feedback loops in RViz or other ROS interfaces.

Architecture Overview

- ROS node publishes control commands ("throttle", "left", "right", "stop", etc.) to the topic /nano_control.
- Arduino Nano subscribes to /nano_control and translates the commands into appropriate motor driver signals.
- Potentiometer readings are converted into steering angles and published back as feedback on /potentiometer_dataand /steering_angle.

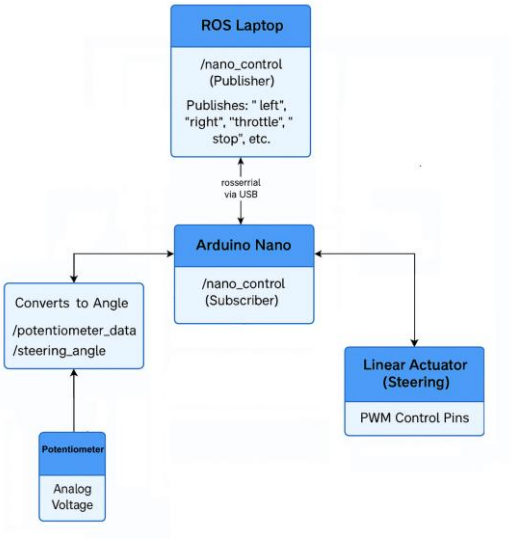

Fig. 4.15 Serial Connection

4.4.2 Key Components in the Arduino Code:

4.4.2.1 ROS Setup and Serial Initialization

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>

ros::NodeHandle nh;
ros::Subscriber<std_msgs::String> sub("nano_control", &messageCb);
ros::Publisher pot_pub("potentiometer_data", &pot_msg);
ros::Publisher angle_pub("steering_angle", &angle_msg);
```

The Arduino initializes a ROS node handle, sets up publishers for analog data, and subscribes to control commands via nano_control.

4.4.2.2 Actuator Control Logic

```
void moveLinearActuator(int direction) {
```

```
    // Extend/retract actuator based on direction
    analogWrite(LPWM_PIN, direction == 1  150 : 0);
    analogWrite(RPWM_PIN, direction == -1  150 : 0);
}
```

The linear actuator for steering is controlled by PWM signals based on direction: left, right, or neutral.

```
void moveStepper(bool forward) {
    // Rotate stepper motor to apply throttle
    digitalWrite(THROTTLE_DIR_PIN, forward  HIGH : LOW);
    for (int i = 0; i < STEPS; i++) {
        digitalWrite(THROTTLE_PUL_PIN, HIGH);
        delayMicroseconds(200);
        digitalWrite(THROTTLE_PUL_PIN, LOW);
        delayMicroseconds(200);
    }
}
```

A **NEMA-23 stepper motor** is used to twist the physical throttle knob. It moves in discrete 500-step bursts to simulate acceleration or deceleration.

4.4.2.3 Potentiometer-Based Feedback

```
float getSteeringAngle(float voltage) {
    return (m * voltage) + b;
}
```

The system maps **analog voltage values** from the potentiometer to **steering angles** using linear interpolation. For example:

- 0.20V corresponds to -22°
- 2.55V is the center (0°)
- 4.80V corresponds to +22°

This enables precise estimation of the actuator's mechanical state without needing an encoder.

4.4.2.4 Callback Function: Message Interpreter

```
void messageCb(const std_msgs::String& msg) {
   if (msg.data == "left") moveLinearActuator(-1);
   else if (msg.data == "right") moveLinearActuator(1);
   else if (msg.data == "center") moveLinearActuator(0);
   else if (msg.data == "throttle") moveStepper(true);
   else if (msg.data == "reverse_throttle") moveStepper(false);
   else if (msg.data == "stop") moveLinearActuator(0);
}
```

Each incoming command string is matched to a corresponding movement pattern—turning the steering, applying throttle, or stopping all motion.

4.4.2.5 Continuous Feedback Publishing

```
void loop() {
   potValue = analogRead(potPin);
   voltage = (potValue / 1023.0) * 5.0;
   current_steering_angle = getSteeringAngle(voltage);

   pot_msg.data = voltage;
   angle_msg.data = current_steering_angle;

   pot_pub.publish(&pot_msg);
   angle_pub.publish(&angle_msg);

   nh.spinOnce();
   delay(50);
}
```

The Arduino continuously measures the potentiometer and publishes:

- Raw analog voltage (/potentiometer_data)
- Calculated steering angle (/steering_angle)

This feedback is visualized live in **RViz** as **3D markers**, aiding debugging and tuning.

4.4.2.6. Why ROS Serial + Arduino Architecture?

This architecture is particularly well-suited for:

- **Modularity**: Allows ROS to handle high-level logic while Arduino focuses on low-level control.
- **Low Latency**: Serial communication ensures near real-time responsiveness to control signals.
- **Lightweight Processing**: The Arduino offloads actuator timing logic, reducing CPU load on the main onboard computer.
- **Scalability**: Additional control logic or sensor feedback (e.g., wheel encoders or IMUs) can be added with minimal changes.

4.4.3 Communication Architecture

The software architecture of the autonomous rover is anchored on **ROS Noetic**, a robust and flexible middleware designed for robotic systems. Deployed on an onboard Ubuntu-based laptop, ROS serves as the core framework to orchestrate communication between sensors (ZED camera, GPS module), control logic, and actuators. It manages both time-critical messaging and multi-node modularity, providing scalability and real-time interaction for complex tasks like perception, planning, and control.

4.4.3.1 GPS Listener Node

This node serves as the entry point for global localization. It establishes a serial connection with the **u-blox NEO-M9N GPS module** and listens to NMEA sentences, primarily $GPGGA and $GNGGA strings. Upon receiving a valid satellite fix, the raw data is parsed and converted into **decimal latitude and longitude values**. These coordinates are published to a dedicated topic (/gps_data), allowing downstream nodes to access real-time positional

data. In addition to continuous position tracking, this node also acts as a trigger for initiating route planning once a stable fix is achieved.

4.4.3.2 Route Planner Node

Once GPS coordinates are available, the Route Planner Node interacts with the **Google Maps Directions API** to compute a navigable path from the current position to a user-defined destination. The API returns the route as a polyline-encoded string, which is decoded into a sequence of geographic waypoints. These waypoints represent a smoothed global path and are published on the /route_waypoints topic. This output acts as the reference trajectory for both throttle control and steering logic. The dynamic nature of the API ensures that the route adapts to real-time start positions, removing the need for prior mapping or static path assignment.

4.4.3.3 Command Publisher Node

The Command Publisher Node is the system's action dispatcher. Based on current position, obstacle feedback, and waypoint proximity, it determines which movement command to issue. These include "throttle" for forward motion, "reverse_throttle" for safe backward movement, and "stop" to pause the rover when necessary. Commands are published to the /nano_control topic, which is monitored by an **Arduino Nano** via **ROS Serial communication**. The Arduino then translates these commands into motor signals, controlling both the throttle (via stepper motor) and steering actuator in precisely timed bursts.

4.4.3.4 Socket Receiver Node

To enable real-time visual feedback, this node establishes a TCP socket connection to an external Python-based object detection script. This external module processes input from the ZED 2i stereo camera, applies a YOLOv5 deep learning model, and determines if any object is within a critical proximity threshold (typically < 2 meters). If a threat is identified, it sends a "stop" signal via socket. The Socket Receiver Node listens for such messages and publishes high-priority commands to interrupt current motion states. This structure allows the system to react instantly to visual threats, without waiting for waypoint updates or GPS reevaluation.

4.4.4 Visualization and Monitoring

A crucial aspect of real-time autonomous navigation is the ability to visualize sensor data, feedback, and actuator responses for system validation and debugging. The system employs

two main interfaces for visualization: **RViz** for 3D spatial representation and **OpenCV-based windows** for camera feed overlays.

4.4.4.1 RViz Integration

RViz is configured to display the following ROS topics in real time:

- /steering_angle: Rendered as rotating markers showing current heading angle.
- /potentiometer_data: Plotted as a live value for analog steering feedback.
- ZED topics (/zed/zed_node/left/image_rect_color, /depth, /point_cloud): Used to visualize RGB streams, depth maps, and 3D point clouds.
- Obstacle locations and planned paths: Displayed using MarkerArray or Path topics to trace waypoints and detected hazards.

This layered visualization allows for complete situational awareness and ensures that all modules (perception, planning, and actuation) are operating in sync.

4.4.4.2 OpenCV Overlay Window

In addition to RViz, a custom Python script opens an OpenCV window that overlays object detection and lane curves on live camera frames. This window shows:

- Detected objects with class labels and bounding boxes.
- Lane detection using a fitted polynomial curve (from LaneNet).
- Optional HUD-style markers show heading and warnings.

Together, these visual tools form a complete real-time dashboard for system observation, making testing intuitive and providing quick feedback for parameter tuning.

4.4.5 ROS and ROS topics used

The decision to use the Robot Operating System (ROS) was central to achieving modularity, scalability, and real-time inter-node communication within the autonomous navigation stack. The system integrates both **ROS 1 (Noetic)** and **ROS 2 (Foxy) running in Docker** environments to leverage the strengths of each.

Why ROS?

- **Modular Architecture:** ROS supports separation of concerns across sensing, planning, and actuation, allowing independent development and debugging of nodes.
- **Real-Time Messaging:** ROS topics, services, and parameters offer synchronous and asynchronous data exchange across processes.
- **Community & Support:** ROS provides a rich ecosystem of packages for robotics applications, including navigation, vision, sensor drivers, and simulation.

Why ROS2 for Vision?

The **ZED SDK** (used with the ZED 2i stereo camera) is officially compatible with **ROS 2**, providing:

- Enhanced support for **depth sensing**, **visual-inertial odometry**, and **hardware acceleration**.
- Access to pre-integrated nodes like /zed/zed_node/point_cloud/cloud_registered for 3D perception.
- Better multi-threading and DDS-based (Data Distribution Service) communication support needed for high-bandwidth data like depth maps and stereo feeds.

4.4.5.1 Interfacing ROS1 and ROS2

A TCP socket bridges as discussed in 5.2 was implemented to allow ROS2-based vision modules to communicate with ROS1-based control and planning nodes. This enables data such as obstacle detections from ZED to be relayed to ROS1 for actuator control.

4.4.5.2 Key ROS Topics

The following ROS topics form the backbone of the system's communication:

Topic Name	Message Type	Description
/gps_data	sensor_msgs/NavSatFix	Publishes real-time GPS coordinates
/route_waypoints	custom_msg/Waypoint[]	Smoothed list of path coordinates from Google API
/nano_control	std_msgs/String	Commands like throttle, stop, and reverse_throttle
/steering_angle	std_msgs/Float32	Current steering angle from potentiometer voltage
/potentiometer_data	std_msgs/Float32	Raw voltage data from analog potentiometer
/zed/zed_node/image_rect_color	sensor_msgs/Image	RGB camera stream for OpenCV and RViz
/zed/zed_node/point_cloud/cloud_registered	sensor_msgs/PointCloud2	Depth + 3D environment mapping

Table 4.16 ROS Topics Used

CHAPTER 5
SYSTEM INTEGRATION

5.1 HYBRID GLOBAL-LOCAL PLANNING ARCHITECTURE

To enable intelligent and responsive navigation in real-world environments, the system integrates both a **Global Planner** and a **Local Planner**, coordinated through a central decision-making module. This hybrid architecture bridges the high-level routing capability of GPS-based path planning with the reactive, real-time decision-making strength of vision and depth-based perception.

The **Global Planner** leverages the GPS module (as described in Section 4.3) to initialize the start location using the NEO-M9N module and generate a smoothed global path to the destination via Google Maps Directions API (4.3.2). The decoded polyline route provides accurate waypoints, which are analyzed to detect significant directional changes. These are flagged as upcoming turns, and distance-to-turn metrics are continuously updated. When the vehicle is within a threshold (typically 2 meters) of a turn point, the planner raises a turn_pending signal to prepare the Local Planner for directional adjustment.

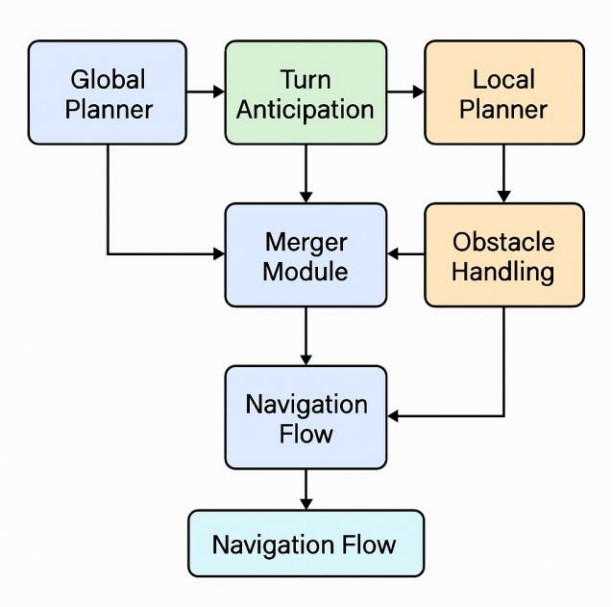

Fig. 5.1 Navigation Planner

The **Local Planner** processes ZED 2i stereo camera feeds to perform real-time lane detection using LaneNet and object detection using a YOLOv5 model paired with depth validation. The system overlays detected lane curves and bounding boxes in RViz for debugging and analysis. Turn anticipation is further refined by analyzing the curvature of detected lanes, enabling the rover to preemptively steer into upcoming turns indicated by the global path.

Obstacle management is executed using a two-stage verification system, combining semantic detection (YOLOv5) with depth filtering from the ZED's registered depth map. If the average depth in the object's region of interest falls below 1.8 meters, the system issues a stop command to prevent collisions. This dual-layer strategy significantly improves reliability by reducing false positives and eliminating unnecessary stops due to distant or irrelevant objects.

At the heart of this architecture is the **Merger Module**, which integrates global path cues with local perception results. It prioritizes local obstacle avoidance over global progression, ensuring immediate threats are handled before resuming motion. The merger also respects turn anticipation flags from the global planner, allowing seamless transitions during intersections and path deviations.

This hybrid system is visualized in real-time using RViz tools, offering developers immediate feedback on perception modules, bounding box overlays, and path alignment. Together, the layered architecture allows for responsive, context-aware navigation that adapts to both long-range routing goals and immediate environmental conditions.

5.2 SOCKET COMMUNICATION AND ROS 1/ROS 2 INTEGRATION

Given that the rover's actuation and control modules operate on ROS 1 while the perception and planning systems run on ROS 2, a TCP socket bridge is implemented to enable real-time cross-platform communication. Commands such as stop, start, and steer_left are transmitted over this low-latency bridge based on decisions made by the merger logic that integrates global and local planner outputs.

For instance, if the local planner detects an obstacle or the global planner signals an upcoming turn, appropriate commands are sent to the ROS 1 layer via the socket interface:

Example:

```
self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

self.socket.connect(('127.0.0.1', 5050))

self.socket.send(b"stop")
```

This bridge ensures the perception system can intervene on the rover's motion control independently.

5.3 OBSTACLE DETECTION LOGIC AND RESPONSE STRATEGY

The obstacle detection subsystem in the autonomous rover is designed to ensure responsive and reliable motion control in the presence of dynamic or static hazards. A **custom-trained YOLOv5 object detection model** is deployed over the RGB stream of the ZED 2i stereo camera to detect relevant obstacles such as people, vehicles, or objects on the road.

To minimize false positives and ensure that only genuine threats trigger evasive actions, the system uses a **two-tiered filtering logic** based on:

1. **Detection Class** (from YOLO)
2. **Depth Map Thresholding** (from ZED)

Obstacle Triggering Conditions:

An obstacle is considered **valid and actionable** only if:

- It is detected **within the lane boundaries** using the local planner's lane estimation.
- The **depth (distance from camera)** is **less than 2 meters**.
- It lies **in the direct projected path** of the vehicle.

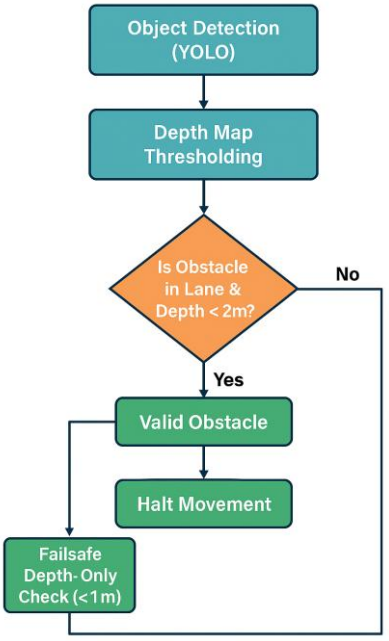

Fig. 5.2 Object Detection Logic

Obstacles outside the lane, behind safety thresholds, or not in the motion vector of the vehicle are **ignored** to prevent unnecessary halts and jerky behavior.

5.3.1 Failsafe Depth-Only Emergency Stop

In addition to vision-based detection, a **failsafe logic** continuously monitors the ZED's raw depth map for any object **within 1 meter**, even if YOLO fails to detect it. This ensures that undetected obstacles (e.g., black or transparent objects, or lighting failures) do not go unnoticed.

Any such violation results in:

- **Immediate throttle deceleration** (soft stop)
- **Publishing a stop command** to the actuator control system via ROS

This hybrid logic ensures both precision and safety, allowing the system to differentiate between valid threats and background clutter while still reacting rapidly to any genuine intrusion.

5.4 DECISION LOGIC AND FAIL-SAFE OBSTACLE HANDLING

A **flow-based decision tree** governs movement commands:

1. If **depth < 1 meter** → emergency stop (regardless of visual detection).
2. If **YOLO detects an object in-lane** and **depth < 2 meters** → soft stop.
3. If **lane curvature indicates a turn** and the global planner confirms it → initiate steering.
4. If the **path is clear** and the **lane straight** → **send the throttle** command.

This logic is designed to prevent abrupt false stops due to visual misclassification or GPS drift while prioritizing **safety, clarity, and smooth motion.**

5.5 REAL-TIME VISUALIZATION AND MONITORING

The system integrates:

- **RViz** for ROS-based visualization of point clouds, lane curves, and bounding boxes.
- A **custom PyQt5 GUI** with a **folium map**, showing current position, route, ETA, and allowing **interactive destination selection** via map clicks.

This dual interface setup supports both **developer debugging** and **user-level testing**, providing real-time awareness of the rover's behavior.

CHAPTER 6

RESULTS AND ANALYSIS

6.1 INTRODUCTION

The Results and Analysis section presents the testing procedures, observations, and performance evaluation of the autonomous rover under various operating conditions. This project aimed to validate its capability in GPS-based path navigation, visual obstacle detection using stereo vision, and lane-following under semi-structured outdoor environments such as a university campus. The outcomes presented here reflect both qualitative and quantitative analyses to assess the system's reliability, precision, and responsiveness.

6.2 TESTING SCENARIOS

To comprehensively evaluate the system, the following real-world and simulated test cases were executed:

- **GPS Navigation Test**:

  Conducted by selecting a real start and destination point within the SRM campus. The rover's current GPS coordinates were used as the origin, and the Google Maps Directions API was queried to generate a smooth path. The rover executed this route using live GPS updates and throttle control. Deviations were tracked using a GPS logger and plotted against the planned path.
- **Obstacle Detection Test**:

  Artificial obstacles (plastic cones, human standing) were placed on the predefined route. The ZED 2i stereo camera captured RGB-D data, and the YOLO model was run in real time. Each time an object was detected within the 2.5m threshold, the system was expected to send a stop command. The timestamps of detection and reaction were logged to evaluate detection reliability and latency.
- **Lane Detection Test**:

  A curved test path with painted lines was used to evaluate LaneNet. ZED RGB images were processed in real time, and lane fits were visualized on-screen. The rover's ability to follow the detected curvature was monitored. Deviation from the lane center was measured at regular intervals using image overlay techniques and manual verification.

- **Throttle Accuracy Test**:

  The GPS-triggered throttle module was tested by commanding the rover to move in 5m, 10m, and 15m increments. The actual distance covered was logged using GPS readings. The test was repeated multiple times to calculate average travel error. This test helped calibrate the stepper motor duration to distance mappings.

- **Integrated Navigation Test**:

  A complete navigation cycle was tested using the full stack: GPS global planner + LaneNet local planner + ZED YOLO obstacle detection. The rover was deployed on a mixed environment route including turns, open spaces, and obstacles. Success was defined as reaching the destination without manual interruption and handling any interruptions autonomously.

6.3 PERFORMANCE METRICS EVALUATED

The following standard performance metrics were used:

- **Localization Accuracy (m)**

Objective: To evaluate how accurately the rover's GPS-reported position follows the planned path generated by the Google Maps Directions API. This metric is essential to assess the effectiveness of the GPS-based global planner in guiding the rover.

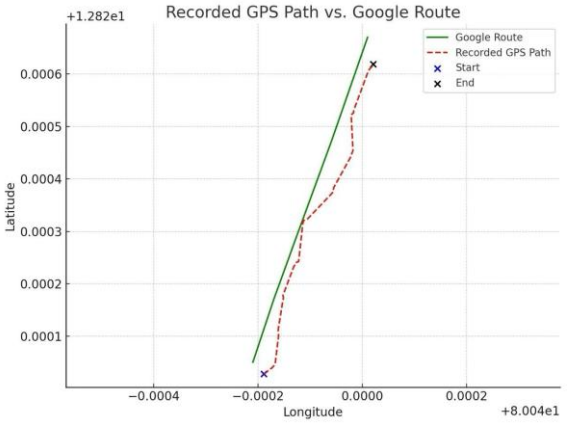

Fig. 6.1: Recorded GPS Path vs Google Route

Testing Setup: The test was conducted within the SRM campus on a straight and open path approximately 45 meters long. The rover was commanded to follow a dynamically generated route using the Google Maps API. The rover's real-time GPS data was recorded and compared against the expected path coordinates.

Visualization

Figure 6.1 illustrates the recorded GPS path (red dashed line) overlaid on the reference path (green line). The rover starts at the blue X and ends at the black X.

Results

- Average Localization Error: ~2.87 meters
- Maximum Observed Deviation: ~5.6 meters
- GPS Module Used: u-blox NEO-M9N (without RTK correction)
- **Obstacle Detection Accuracy (%)**

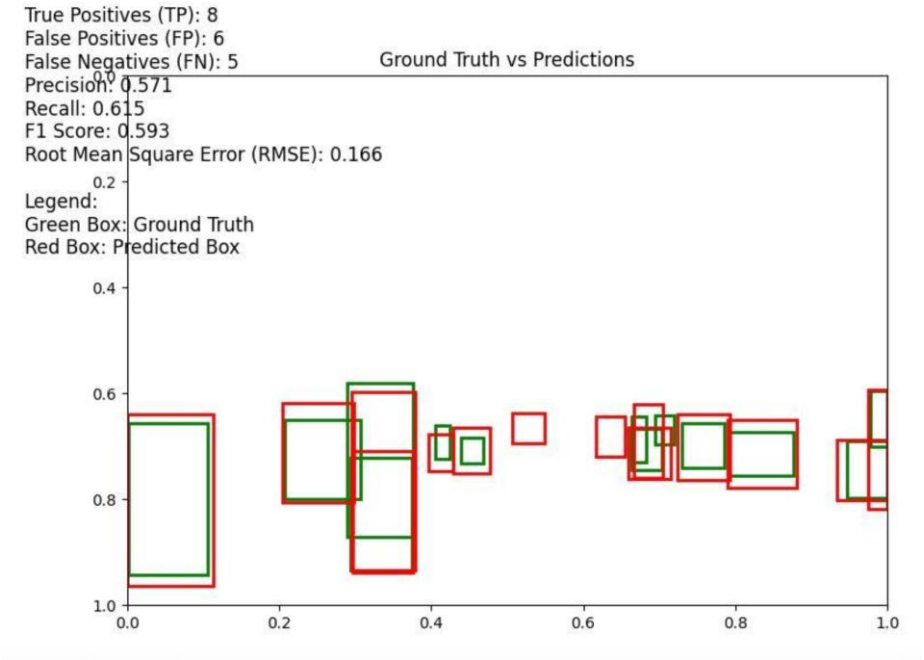

Fig. 6.2: Manually annotated objects vs detected objects

Testing Setup: To enable reliable obstacle detection tailored to our autonomous navigation setup, a YOLOv5 model was custom-trained on a dataset comprising labeled images of road obstacles such as vehicles, cones, and pedestrians. The dataset was annotated manually using Roboflow, and the model was trained for 150 epochs with augmentation techniques to enhance generalization.

The trained model was evaluated against a ground truth dataset to assess its detection performance. The visual comparison of bounding boxes (green = ground truth, red = predicted) demonstrates the model's effectiveness and highlights minor alignment offsets.

Evaluation Metrics:

- **True Positives (TP):** 8
- **False Positives (FP):** 6
- **False Negatives (FN):** 5
- **Precision:** 0.571
- **Recall:** 0.615
- **F1 Score:** 0.593
- **Root Mean Square Error (RMSE):** 0.166

These metrics indicate moderate detection performance, suitable for in-field obstacle filtering when paired with ZED 2i depth constraints. Further improvements can be achieved by increasing the dataset size and refining annotations.

6.4 SUMMARY OF RESULTS

The system was able to:

- Dynamically generate and follow GPS routes with acceptable localization drift.
- Accurately detect obstacles within a predefined depth threshold and halt motion reliably.
- Follow lanes using deep-learning-based inference even on non-linear, curved paths.
- Respond to detection events with minimal delay using real-time vision pipeline.

6.5 OBSERVATIONS AND INSIGHTS

- The system performs best in open-sky, well-lit conditions.
- False positives were minimal due to depth filtering after YOLO detection.
- Lane detection can degrade in the absence of clear line markings.
- GPS drift introduced minor heading inconsistencies, especially near turns.

6.6 COMPARISON TO OBJECTIVES

Objective	Achieved	Notes
Dynamic GPS Navigation	✅	Slight drift, but within acceptable margin
Real-Time Obstacle Detection	✅	Depth-validated YOLO effective up to 2.5m
Lane Following	🛠️	Proficient on straight paths. Curved path navigation needs significant improvement.
System Integration	✅	ROS 2 + ROS 1 socket-based communication stable

Table 6.3 Objectives and Progress

CHAPTER 7
CONCLUSION AND SCOPE FOR FUTURE WORK

7.1 CONCLUSION

The **Vision-Assisted Autonomous Navigation System** developed in this project demonstrates an effective integration of GPS-based global planning and stereo vision-based local perception, enabling a small-scale autonomous rover to navigate semi-structured environments intelligently and reactively.

The system's foundation lies in hybrid planning architecture. The **Global Planner**, using the **u-blox NEO-M9N GPS module** and Google Maps Directions API, generates a smooth route as a series of waypoints. These waypoints guide the rover using dynamic throttle control based on real-time distance estimation, allowing for GPS-based path following without pre-mapping.

Complementing this is the **Local Planner**, built around the **ZED 2i stereo camera**, which provides both RGB and depth data. A custom **YOLOv5 model** is used to detect obstacles, and these detections are filtered by depth thresholds to determine whether the obstacle is within the vehicle's path. Additionally, a **fail-safe depth-based stop logic** triggers deceleration when any object is detected within 1 meter, even without visual classification.

For accurate lane tracking, a **LaneNet-based model** processes camera input to detect road lanes and curves, aiding the rover in maintaining its position and preparing for turns. Turn instructions are flagged in advance by the Global Planner, allowing the Local Planner to anticipate and execute smooth directional changes.

The control layer is handled via **ROS Noetic** on an onboard laptop. **ROS Serial** interfaces with an **Arduino Nano**, which translates ROS commands into actuator signals for the throttle (via a NEMA-23 stepper motor) and steering (via a linear actuator). Potentiometer feedback is continuously published to estimate real-time steering angle.

A **ROS 1 to ROS 2 bridge**, implemented using **TCP sockets**, allows separation between perception (handled in ROS 2 due to ZED SDK compatibility) and actuation (handled in ROS 1), ensuring smooth inter-process communication and real-time performance.

Visualization tools like **RViz** and a custom **PyQt5-based GUI** provide full situational awareness, allowing developers to view live camera feeds, path overlays, and object detections during testing and debugging.

This modular and scalable system architecture showcases a robust approach to real-world autonomous navigation using low-cost hardware, open-source tools, and intelligent software design.

7.2 SCOPE FOR FUTURE WORK

1. Robust Localization through Sensor Fusion:
Improve localization reliability, especially in areas with limited satellite visibility, by integrating IMU data with existing GPS and vision inputs. This can be achieved using an Extended Kalman Filter (EKF) or factor graph optimization.

2. Full Migration to ROS 2:
Transitioning the entire system architecture to ROS 2 from the current hybrid ROS 1/ROS 2 setup would offer advantages such as reduced latency, real-time DDS messaging, and better compatibility with modern sensors. Utilizing the `pyserial` library is recommended for this migration.

3. Autonomous Obstacle Avoidance and Rerouting:
Enhance the system's ability to handle obstacles by implementing local trajectory replanning. Instead of simply stopping, algorithms like DWA (Dynamic Window Approach) or TEB (Timed Elastic Band) would enable autonomous navigation around obstructions.

4. Improved Lane Following in Challenging Environments:
Increase the robustness of lane following by training and testing the LaneNet model under adverse conditions. Incorporating temporal filtering or multi-frame inference could further stabilize lane detection.

5. Dynamic Throttle and Steering Control:

Move from fixed throttle bursts to a more adaptive system by implementing closed-loop speed control. Utilizing wheel encoders or Hall sensors would allow for dynamic speed adjustments based on factors like curvature, traffic, and terrain.

6. Edge Deployment and Efficiency:

Enable more compact, power-efficient, and readily deployable operation by porting the vision model to edge computing platforms such as Jetson Nano/Xavier or a Raspberry Pi with Coral TPU.

These directions not only enhance system performance but also extend its application to more complex domains such as urban autonomous driving, last-mile delivery robots, or warehouse automation.

REFERENCE

[1] Stereolabs, “ZED 2i Stereo Camera Documentation,” *Stereolabs Developer Portal*, [Online]. Available: https://www.stereolabs.com/docs/

[2] Ultralytics, “YOLOv5: Real-Time Object Detection,” *GitHub Repository*, [Online]. Available: https://github.com/ultralytics/ultralytics

[3] MaybeShewill-CV, “LaneNet: Deep Learning-Based Lane Detection Using TensorFlow,” *GitHub Repository*, [Online]. Available: https://github.com/MaybeShewill-CV/lanenet-lane-detection

[4] Open Source Robotics Foundation, “Robot Operating System (ROS) Wiki,” [Online]. Available: http://wiki.ros.org/

[5] Google, “Google Maps Directions API,” *Google Maps Platform*, [Online]. Available: https://developers.google.com/maps/documentation/directions/start

[6] ROS Drivers Community, “rosserial_arduino – ROS Serial Communication with Arduino,” *GitHub Repository*, [Online]. Available: https://github.com/ros-drivers/rosserial

[7] u-blox AG, “NEO-M9N GNSS Module – Technical Documentation,” [Online]. Available: https://content.u-blox.com/sites/default/files/u-blox-M9-NAV-Tutorial_ReceiverDescrProtSpec_UBX-20050193.pdf

[8] OpenCV.org, “OpenCV: Open Source Computer Vision Library,” [Online]. Available: https://docs.opencv.org/

[9] Python Software Foundation, “PyQt5 and Folium – Libraries for GUI and Mapping,” [Online]. Available: https://pypi.org/project/PyQt5/, https://python-visualization.github.io/folium/

[10] sigmaai, “Self-Driving Golf Cart Project,” *GitHub Repository*, [Online]. Available: https://github.com/sigmaai/self-driving-golf-cart

[11] rtarun1, “Autonomous Wheelchair Project – SRMIST,” *GitHub Repository*, [Online]. Available: https://github.com/rtarun1/autonomous-wheelchair-SRMIST

APPENDIX 1 PROJECT REPOSITORY

The entire software stack and the instructions are available on GitHub. Follow the link or scan the QR to access the repository.

https://github.com/Vishaaaaaal/vision-gps-AutonomousRover

Mhp 04 Mhp 04

MHP 04

Quick Submit

Quick Submit

SRM Institute of Science & Technology

Document Details

Submission ID

trn:oid:::1:3242376341

Submission Date

May 7, 2025, 4:41 PM GMT+5:30

Download Date

May 7, 2025, 4:45 PM GMT+5:30

File Name

MHP_04.pdf

File Size

5.8 MB

54 Pages

9,081 Words

53,598 Characters

4% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **27** Not Cited or Quoted 3%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

3% Internet sources
1% Publications
1% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **27** Not Cited or Quoted 3%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

3% Internet sources
1% Publications
1% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Publication	
Colino, Francisco José Barbosa Marques. "Dynamic Obstacle Detection and Motio...		<1%
2	Student papers	
Washington University in St. Louis		<1%
3	Student papers	
ITESM: Instituto Tecnologico y de Estudios Superiores de Monterrey		<1%
4	Internet	
developers.maxon.net		<1%
5	Internet	
fabacademy.org		<1%
6	Internet	
www.mdpi.com		<1%
7	Internet	
developers.google.com		<1%
8	Internet	
machinelearningmastery.com		<1%
9	Student papers	
University of Liverpool		<1%
10	Student papers	
Queensland University of Technology		<1%

11	Internet	
arxiv.org		<1%
12	Internet	
www2.mdpi.com		<1%
13	Internet	
cgspace.cgiar.org		<1%
14	Internet	
businessdocbox.com		<1%
15	Internet	
autodocbox.com		<1%
16	Internet	
shodhganga.inflibnet.ac.in		<1%
17	Internet	
www.intorobotics.com		<1%
18	Student papers	
Queen Mary and Westfield College		<1%
19	Internet	
ceur-ws.org		<1%
20	Internet	
dev.to		<1%
21	Internet	
robu.in		<1%
22	Internet	
123dok.org		<1%
23	Publication	
Riccardo Karim Khamaisi, Matteo Perini, Alessio Morganti, Marco Placci, Fabio Gr...		<1%
24	Internet	
blog.csdn.net		<1%