IT4010
Research Project

4th Year, 2nd Semester

Final report

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

Declaration

We hereby declare that this group report is the result of our own work and has not been submitted, either in whole or in part, for any degree or diploma at any other university or institute of higher education. To the best of our knowledge and belief, it does not contain any material previously published or written by any other person, except where proper acknowledgment is made in the text.

We also grant the Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and distribute this report, in whole or in part, in print, electronic, or any other medium. We retain the right to use the content, in whole or in part, in future works such as articles or books.

Date: 11th April 2025

……………………….   ……………………….
Signature of the Supervisor   Signature of the Co-Supervisor

Abstract

This research focuses on developing a smart and scalable patrol robot system that can operate effectively in changing and unpredictable environments. The system combines several advanced technologies to improve navigation, decision-making, and teamwork among robots. It uses reinforcement learning to help the robot adjust its path, LIDAR-SLAM for mapping and avoiding obstacles, and GPS-Waypoint navigation for guiding the robot along planned routes.

The robot also uses sensors to monitor the environment, including weather conditions, air quality, and its own health status like battery level. Based on this sensor data, the system can adjust patrol routes to improve safety and efficiency.

Multiple robots work together in real-time by sharing information. Tasks are assigned based on their location, battery status, and environmental conditions. This teamwork ensures better coverage and smarter use of each robot’s resources.

The system uses Edge Computing and Internet of Things (IoT) technologies for fast data processing and communication. Machine Learning (ML) models help the robots make quick and smart decisions based on real-time data.

In summary, this project presents an intelligent patrol robot system that can adapt to its surroundings, work in teams, and respond to changes quickly using modern technologies like IoT, ML, and edge computing.

Keywords: Obstacle Avoidance, GPS-Waypoint Navigation, Reinforcement Learning, LIDAR-SLAM, IoT, Edge Computing, Weather Prediction, Multi-Robot Collaboration, Machine Learning, Environmental Sensing

Acknowledgement

We would like to express our sincere gratitude to our supervisor for their continuous guidance, valuable insights, and motivation throughout the duration of this project.

We are also thankful to our co-supervisor for their consistent support and constructive feedback, which greatly contributed to the development and improvement of our work.

Furthermore, we would like to acknowledge the efforts of all our team members for their dedication, collaboration, and valuable contributions. The teamwork and mutual support among us were crucial in achieving the objectives of this project.

Table Of content

Contents

1. Function 1 Introduction

1.1 Background and Literature

Autonomous patrol robots are transforming surveillance in environments such as warehouses, campuses, and public spaces with the potential to provide real-time monitoring and threat detection. The robots generate vast amounts of video and sensor data, which must be processed efficiently to maintain operational reliability with battery constraints. Edge computing on devices like the Raspberry Pi enables low-latency processing and reduces dependency on cloud infrastructure [15]. The LZ4 protocol, which is popular for its speed in compression, is optimum for processing videos in real time [1], while Zstandard offers improved compression ratios for sensor data [2]. AES-256 encryption ensures secure data transfer to cloud platforms like Firebase, whose scalability accommodates real-time data storage [7, 11]. Rule-based compression algorithms for battery-efficient compression have been put forward in IoT systems to make efficient use of energy [4].

This project integrates these technologies to develop an efficient, secure, and scalable patrol robot system.

1.2 Research Problem

Patrol robots face challenges in managing high-bandwidth video and sensor data on limited battery power. Cloud-based processing introduces latency and bandwidth issues, while uncompressed data transmission drains battery life and overwhelms network resources. The research problem is to develop an adaptive system that efficiently compresses video and sensor data on the edge using a Raspberry Pi, optimizes compression based on battery levels, and securely uploads data to Firebase for real-time monitoring and analysis.

2. Research Gap

Patrol robot systems typically utilize non-real-time battery limit or environmental state non-adaptive static compression methods. These utilize cloud computation with heavy latencies and lack efficient compression and secure cloud storage integration. The current research fills this gap by developing an amalgamated system with high-speed LZ4 video compression, Zstandard sensor data compression, AES-256 encryption, and Firebase integration, optimized based on rules for energy efficiency.

3. Research Objectives

3.1 Functional Objectives

The main goal is to construct a flexible patrol robot that optimates information processing, battery utilization, and secure data transfer. The system accomplishes the following goals:

3.1.1 *Video Compression Using LZ4*

LZ4 protocol will be applied for compressing strams of high resolution video. It will be done at a compression ratio of approximately 2:1 which would lower the required bandwidth by 50% without lower the quality of real-time surveillance.

3.1.2 *Sensor Data Compression Using Zstandard*

Compressing temperature, humidity, and motion data with the Zstandard protocol achieves up to 3:1 reduction with no loss of relevant information.

3.1.3 *Battery-Aware Compression Optimization*

A battery level monitoring algorithm will be implemented that adjusts data compression controls to sustain activity time for an additional 30%.

3.1.4 *Secure Data Upload to Firebase*

Sensor data and video fragments will be encrypted using AES-256 which guarantees security and will be uploaded in real-time to firebase for easy access any time.

3.1.5 *Web-Based Administrator Dashboard*

Create an alert-based manaul control mechanism on the web so users have full access to live video streams, control the sensors, check overall battery level, and manage robot navigation.

4. Methodology

4.1 Introduction to Methodology

The methodology is concerned with designing a patrol robot that performs video and sensor data processing with compression and upload using Raspberry Pi and Firebase. This encompasses system design, data processing, compression, encryption, and upload.

4.2 System Design

The system is structured as a modular architecture comprising of:

Data Acquisition: Gathers video streams and sensor data (temp, humidity, motion).

Compression Module: Uses Zstandard for sensor data and LZ4 for video.

Battery Conservation: Changes parameters with respect to battery levels, optimizing for battery life.

Encryption and Upload: Uses AES-256 for encryption and uploads data to Firebase.

Web Dashboard: Shows live data together with alerts for any critical conditions.

4.3 Data Processing Pipeline

The pipeline includes:

Data Collection: video 1080p at 30fps, and sensors at 1Hz sampling rate.

Compression: LZ4 for video and Zstandard for sensor data.

Battery Monitoring: Checking battery levels “on the go” (Must: >70%, Should: 30-70%, Critical:<30%).

Encryption: Compressed data is encrypted using AES-256.

Upload: upload the data to Firebase Realtime Database in a secure way.

4.4 Compression and Encryption Implementation

LZ4: set for aggressive compression speed to 500 MB/s.

5. System Architecture

5.1 Edge Computing on Raspberry Pi

The Raspberry Pi 4 (8GB RAM) serves as the edge computing platform, running all processing tasks:

- Video capture and compression.
- Sensor data acquisition and compression.
- Battery monitoring and optimization.
- Encryption and Firebase upload.

5.2 Compression Modules

- **LZ4 Module**: Processes video streams, reducing data size by 50%.
- **Zstandard Module**: Compresses sensor data, achieving a 3:1 ratio.

5.3 Encryption and Firebase Standards

- **AES-256 Encryption**: Encrypts compressed data with a 256-bit key.
- **Firebase**: Stores encrypted data in a Realtime Database, accessible via the web dashboard.

6. Testing and Implementation

6.1 Testing Strategy

Assessment was centered on:

- Compression Efficiency: Evaluating ratio and speed of compression.
- Battery Optimization: Assessing time of use prolongation.
- Data Security: Checking encryption strength and restrictive API access to Firebase.
- Real-Time Performance: Evaluating responsiveness and latency of the dashboard.

6.2 Performance Metrics

- Compression Ratio: LZ4 - 2:1 for video, Zstandard - 3:1 for sensor data.
- Processing Latency: compression and encryption below 100ms.
- Battery Life Extension: Compression up to 30% with adaptive algorithms.
- Upload Time to Firebase: Upload less than 200ms per data packets.

6.3 Implementation Environment

- Hardware: Raspberry Pi 4 with 4GB RAM and 32GB SD card.
- Software: Python 3.9 with Flask for the API, Firebase SDK, LZ4 and Zstandard libraries, cryptography libraries.
- OS: Raspberry Pi OS 64-bit.
- Development Tools: Postman for API testing and VS Code for programming.

7. Results and Discussion

7.1 Data Compression Performance Results

LZ4: Provided a compression ratio of 2:1 for the 1080p video while decreasing bandwidth by 50%, at a processing speed of 480 MB/s.

Zstandard: Provided a 3:1 compression ratio for the sensor data while keeping the information intact at a processing rate of 200MB/s.

7.2 Optimization of the Algorithm Excluding the Battery

The heuristic method decreased the operational limits modifier, such as lowering the video frame rate from 30fps to 15fps, leading to 28% enhanced performance duration during the test running over 6 hours.

7.3 Performance in Real Time

The system completed the processing of the video and sensor data in less than 100 milliseconds, while the Firebase upload was completed in less than 200 milliseconds. his led to the web dashboard refreshing with live data in under 50 milliseconds after receiving the infomation. The alerts too, were provided in real-time.

1. Function 2 Introduction

1.1 Background and literature

Autonomous robots have been at the forefront of modern industrial automation in recent years, especially in settings like distribution centers, factories, and warehouses. As they navigate intricate and changing surroundings, autonomous robots are made to carry out a variety of duties, including environmental monitoring, inventory control, and surveillance. By lowering human engagement in potentially hazardous activities, automation of certain duties not only increases operational efficiency but also enhances safety.

The capacity of autonomous robots to negotiate dynamically changing conditions is one of the main challenges to their application in real-world settings. Early solutions used pre-programmed routes or basic algorithms to follow static tracks, which weren't enough in situations where things change often, such as when people move or there are unforeseen barriers or changes in light. More sophisticated navigation algorithms that can adjust to these real-time changes are being developed in order to get around this restriction and enable robots to vary their routes in response to sensor feedback.

LiDAR (Light Detection and Ranging) technology plays an important part in this by providing high-resolution, real-time data about the environment to the robot. LiDAR enables autonomous robots to map their environs, recognize impediments, and navigate easily even in the presence of dynamic changes. The technology, along with video sensors and motion detectors, enables robots to perceive and react to their surroundings more intelligently. [1]

Reinforcement Learning (RL), notably approaches such as Q-learning and Deep Q Networks (DQN), has been used to advance autonomous robot navigation. RL enables robots to continuously learn and adapt their behavior based on feedback from their surroundings, which is critical when traversing dynamic or unpredictable environments. Unlike traditional path-planning algorithms, which rely on pre-defined routes, RL-based systems can dynamically adjust paths in real time, allowing robots to respond to changes such as barriers or lighting. [2]

In recent research, a variety of approaches to RL-based robot navigation have been proposed. Some research, for example, has focused on the application of Q-learning for mapless navigation, which allows robots to move without relying on pre-mapped settings. In addition, sensor fusion algorithms that incorporate LiDAR, camera, and other sensory inputs have been investigated to improve real-time decision making and obstacle avoidance. These technologies have demonstrated that robots outfitted with them are capable of not just moving about effectively but also adapting to new settings in a flexible manner.[3]

While significant progress has been made, it remains a problem to integrate these advanced algorithms with the physical constraints of robotics, such as processing limitations, sensor accuracy, and the natural complexity of the real environment. The research presented in this thesis seeks to address these challenges by combining LiDAR-based mapping and reinforcement learning for dynamic pathway navigation, resulting in a system that allows autonomous patrol robots to respond in real time to obstacles, lighting changes, and other environmental factors. [5]

1.2. Research Gap

While plenty of study has been done on path navigation for autonomous robots, many of the most difficult tasks remain for effectively mapping dynamic path navigation and real-time adaptation in a warehouse environment. The current literature includes highly advanced path planning algorithms, reinforcement learning paradigms, and sensor fusion. However, there are differences in how successfully those methodologies can be applied to autonomous patrol robots in dynamic real-world settings such as warehouses. They can be split into these categories:

1.2.1 Lack of Real-Time Adaptation in Path Planning

While several algorithms have been created to allow autonomous robots to follow pre-programmed trajectories, few can dynamically adapt paths in real time in response to environmental changes. Many present systems rely on static path planning or limited rerouting based on specified barriers. However, these algorithms fall short when dealing with unforeseen changes like human mobility, changing lighting conditions, or newly imposed barriers. There is a need for algorithms that can constantly learn and adapt to new situations without human intervention.[13]

1.2.2 Insufficient Sensor Fusion for Dynamic Environments

Most existing systems use only one type of sensor, such as LiDAR, cameras, or proximity sensors, to map the surroundings. The complexity in typical real-world warehouse environments with dynamic objects, varying light levels, and frequent layout changes necessitates the use of diverse sensors to offer more reliable navigation. Although sensor fusion techniques have improved, current models are not yet effective in integrating heterogeneous sensors like LiDAR, camera, and motion sensors in real time. This is an area that might be tapped for improving autonomous navigation through further integration of these sensors to provide better models of the environment. [7]

1.2.3 Limitations in Scalability and Robustness of Reinforcement Learning

Reinforcement learning (RL) and Q-learning, in particular, have been shown to hold promise in enabling robots to learn how to adapt to dynamic environments. However, its application to large-scale, complex environments such as warehouses remains to be exhaustively explored. The main challenge arises from the scalability of RL models to support massive regions with intricate routes and obstacles. Current RL models tend to fail in making the exploration-exploitation trade-off in large, continuous environments. In addition, most RL-based approaches are difficult to scale to high-dimensional environments without impacting their performance. The models also have to maximize exploration and exploitation in a way that maximizes efficiency of navigation over large distances and regions over time.[15]

1.2.4 Real-World Testing and Implementation

While numerous theoretical models and simulation studies have been undertaken, relatively little large-scale real-world testing has occurred, particularly in warehouses. Much of the existing research is focused on simulated environments in which variables like lighting, obstacles, and human interference can be controlled. But in real-world settings, the conditions are not random, and the challenge of dealing with interacting people, machines, and other dynamic variables requires a more robust test and verification of the navigation algorithms. Furthermore, a significant portion of the testing is done within a specific class of environments and with little consideration of the implementation of the system within a real, dynamic warehouse environment.[16]

1.2.5 Integration of Autonomous Patrol Robots with Warehouse Systems

Although patrol robots have been designed for overall security and monitoring, their interfacing with comprehensive warehouse management systems is still an emerging area of research. Research to date will often overlook the broader context within which the robots are being implemented, such as integration with inventory systems, emergency response procedures, and human drivers. The incorporation of these robots should be smoother into the daily operations of the warehouses so that they not only travel well but also contribute positively to the overall management of the warehouse.[17]

1.3 Research problem

Autonomous robots have substantially improved their abilities to move and perform tasks in controlled situations. Autonomous robots encounter a number of issues in uncontrolled, dynamic real-world situations such as warehouses due to their capacity to respond to unplanned changes in their surroundings, such as human mobility, fluctuating lighting conditions, and unexpected impediments.

While present navigation algorithms provide precomputed routes to robots, they cannot adjust dynamically to changing situations in real time. Existing systems, in particular, struggle to adjust for barriers, differences in lighting, and other environmental factors, resulting in robot failure under certain scenarios.

The primary problem addressed by this study is the absence of a dynamic path modification algorithm that allows autonomous robots to:

- Adapt in real time to unexpected environmental changes such as obstructions, motion, and changing lighting conditions.

- Use LiDAR and video sensors to successfully navigate and scan the environment, allowing the robot to continuously learn and adjust its path.

- Use Reinforcement Learning (RL) to improve decision-making by allowing the robot to acquire optimal navigation techniques from interactions with its surroundings.

- Ensure that the path navigation system is scalable and stable in big, complicated warehouse environments.

Additionally, there is no extensive real-world experimentation of these algorithms in dynamic warehouse settings, wherein human interference, high environmental change rates, and diverse obstacles pose real challenges to the efficiency of current solutions. Consequently, this research suggests developing a dynamic path-following algorithm that integrates reinforcement learning and LiDAR-based mapping to enable the robot to dynamically adjust its patrol route with real-time sensory input and environmental changes.

The research problem can be broken down into the following key questions:

- How can reinforcement learning techniques be effectively used with LiDAR-based mapping to allow for dynamic path modifications in real time?

- What is the best effective method for combining sensory data (e.g., LiDAR and cameras) to improve obstacle identification and path guidance in a complex, changing environment?

- How can the system be evaluated and tuned for scalability, robustness, and practical use in big warehouse environments?

By answering these questions, this study will provide a new way of improving autonomous patrol robots' navigation capability to enable them to adapt in environments with changing features and will allow them to perform their tasks better.

1.4 Research Objectives.

The primary goal of this research is to create an autonomous navigation system for path-following patrol robots operating in dynamic warehouse environments. The system must follow prescribed courses while responding in real time to environmental changes such as barriers, light intensity, and motion. The research focuses on the following specific objectives.

1.4.1 Develop a Dynamic Path Navigation Algorithm

- To develop and implement an algorithm that allows the patrol robot to follow predetermined patrol routes.

- To include real-time sensor feedback (such as from LiDAR, GPS, and cameras) into the navigation system, allowing the robot to dynamically adapt its path in response to environmental changes.

- To apply reinforcement learning (RL) approaches to allow the robot to make intelligent judgments and optimize its course as it encounters obstacles or changes in its environment.

1.4.2 Integrate IoT-based Sensor Network for Real-time Perception

- To integrate LiDAR, GPS, and Camera to give the robot a complete understanding of its surroundings, allowing it to recognize obstacles and change its path in real time.

- To construct a system in which the robot may use sensor data for Simultaneous Localization and Mapping (SLAM) to create and update an environmental map, as well as determine its position on that map for effective navigation.

1.4.3 Design Autonomous Navigation with Obstacle Detection and Avoidance

- To develop SLAM-based autonomous navigation, which enables the robot to avoid obstacles in its environment by constantly updating its internal map and modifying its path.

- To ensure that the robot can detect obstructed paths and redirect dynamically, ensuring continuous and efficient patrol without human involvement.

1.4.4 Develop a Web Interface for Monitoring and Control

- To develop a web-based dashboard that enables operators to manually pick patrol routes, track real-time position updates, and monitor system performance.

- Implement a feature that allows operators to view previous and dynamically generated patrol routes, as well as impediments and alternative paths.

1.4.5 Test and Evaluate the Performance of the Navigation System

- To assess the performance of the dynamic path navigation system, numerous scenarios in a warehouse setting were simulated, including the inclusion of barriers, changes in lighting, and motion detection.

- To evaluate the system's response time, accuracy in obstacle recognition and avoidance, and ability to alter pathways in real time.

- To evaluate the system's ability to provide efficient patrol coverage while dynamically modifying its routes based on sensor inputs.

1.4.6 Contribute to the Development of Smart Warehouse Solutions

- To advance the field of autonomous systems by creating a feasible solution for smart warehouses that allow robots to navigate, patrol, and adapt to dynamic environmental changes.

- To demonstrate how reinforcement learning, SLAM, and real-time sensor integration may be used to automate warehouse monitoring, eliminating human intervention while enhancing efficiency and security.

2. Methodology

2.1 methodology

2.1.1 System Architecture Overview

The autonomous warehouse patrol robot's system architecture is intended to enable efficient, real-time path planning, adaptability to changing settings, and seamless integration of hardware and software components. The core components are:

- Autonomous Robot Unit with GPS, LiDAR, and camera sensors for environmental perception.
- Edge Computing Module enables real-time data processing and decision-making.
- A web interface (FastAPI) for route monitoring and manual path setting.
- The Machine Learning Module employs reinforcement learning for adaptive path selection.
- SLAM Module for real-time environmental mapping and localization.

This modular architecture enables the scalable and efficient integration of subsystems, easing maintenance and upgrades. Each component communicates via standard interfaces and message-passing protocols, resulting in low latency and reliable data flow.

2.1.2 Sensor Setup & Data Flow

Precise and effective data gathering is crucial to the navigation system's success. The following sensors used by the robot:

- LiDAR (Light Detection and Ranging): Utilized for obstacle detection, environment scanning, and depth estimation. The sensor has a 360-degree view of the environment through laser pulses being sent and reflected signals being sensed.

- GPS Module: Offers global positioning and spatial orientation to monitor the position of the robot in massive warehouses.

- Cameras: Capture visual input for motion detection, light intensity estimation, and environment recognition.

Data Flow Process:

- Raw Data Collection: Sensors capture real-time raw data such as distance to objects, light, and GPS position.

- Edge Processing: Fundamental filtering and preprocessing (e.g., noise removal) are accomplished on-device to reduce latency.

- Decision Module Input: Preprocessed data is input into the Q-learning model and SLAM module.

- Action Execution: Depending on the analysis, the system either updates the path of the robot or runs obstacle avoidance behaviors. Logging & Feedback: Results are logged to the database and can be viewed on the web dashboard.

2.1.3 Path Navigation using Reinforcement Learning (Q-learning)

The patrol robot learns optimal paths using Q-learning, a model-free reinforcement learning technique. It enables the robot to select its next step depending on the benefits gained from previous experiences.

- States: Each waypoint or grid represents one state.
- Actions: Possible paths the robot can take from a given waypoint.
- Rewards:
    - +10 for successfully navigating to an available waypoint.
    - -5 for selecting a path that is blocked or has a high light intensity.
- Learning Strategy: The epsilon-greedy method is intended to strike a balance between exploration and exploitation. The robot initially investigates many paths before gradually exploiting the most rewarding ones as it learns.

- **Q learning VS Dijkstra’s Algorithm**

Dijkstra’s Algorithm

- Nature: Classical graph-based shortest path algorithm.
- Strengths:
    - Guarantees the shortest path in static environments.
    - Deterministic and reliable with known maps.
- Limitations:
    - Needs a fully known, predetermined map of the world.
    - Cannot learn from previous experiences or adjust to new challenges dynamically.
    - Computationally costly in the face of frequent updates.
    - Re-computation whenever the environment is altered.

Q learning

- Nature: Model-free reinforcement learning algorithm.

- Advantages:

  - Learns best routes through experience, instead of depending on fixed maps.

  - Dynamically adapts according to the feedback from sensors such as LiDAR and cameras.

  - Effective in adapting settings where hindrances or light levels differ.

  - Enables continuous learning—performance improves incrementally.

- Limitations:

  - Needs training data to develop an effective policy.

  - May take time to converge in large settings without proper training techniques.

Why Q-learning is Better for This System:

In the dynamic warehouse setting, situations such as lighting, motion, and random obstacles occur with frequency. The robot must rebuild its route dynamically, a functionality to which Dijkstra's algorithm is poorly adapted. Q-learning, however:

- Learns from real-time sensor data.
- Continues to improve through exploration and exploitation.
- Requires no complete map upfront.
- Adapts its behavior based on reinforcement signals (rewards and penalties).

2.1.4 SLAM for Real-Time Mapping

Simultaneous Localization and Mapping (SLAM) allows the robot to map an unfamiliar area and simultaneously localize itself in it.

Functionality:

- Updates the robot's position continuously.
- Constructs a local map using LiDAR and visual data.
- Helps to rectify GPS drift indoors by facilitating sensor fusion localization.

SLAM Integration:

- SLAM is fused with GPS and LiDAR inputs to enable the robot to:
- Sustain a consistent path mapping.

- Update its knowledge of the environment in real time. Effectively navigate around well-known and newly discovered hurdles.

The integration of SLAM and Q-learning means that the robot is able to both learn to navigate and also explore intelligently even in new spaces or rearranged warehouse territory.

2.1.5 Integration with GPS, LiDAR, and Cameras

The combination of various sensor modalities improves the robot's knowledge about the environment. There is a particular function for each sensor:

- GPS aids in upper-level waypoint navigation over very large areas.
- LiDAR scans the environment and detects physical obstacles.
- Cameras help to detect motion or changes in the level of light, inducing adaptive responses.

Sensor fusion integrates information from all these sensors, minimizing uncertainty and enhancing navigation precision. Redundant and heterogeneous sensing results in robustness to changing conditions such as lighting, clutter, or partial occlusion.

2.1.6 Web Interface via FastAPI

The system has a light-weight web-based control and monitoring interface, which was developed with FastAPI. The main features are:

- Path Visualization: Displays real-time current patrol route and robot location.
- Manual Path Selection: Users can select or override patrol paths when needed.

- Log Viewer: Displays historical patrol information, such as obstructions overcome and diverted routes.
- Live Sensor Status: Real-time input from GPS, LiDAR, and camera systems.

The FastAPI backend exposes a RESTful interface to communicate with the robot's onboard systems and database, thereby enabling human operators to remotely control operations.

2.1.7 Edge Computing for Real-Time Decisions

In order to facilitate rapid and low-latency decision-making tasks, key computations are transferred to an onboard edge computing module (e.g., NVIDIA Jetson or Raspberry Pi4). This module:

- On-site processes LiDAR and camera data.
- Makes Q-learning-based decisions in milliseconds.
- Decreases the reliance on foreign servers or cloud-based services.

Enhances autonomy through decreased communication latency. This method is especially important where network reliability could be a problem. Edge computing ensures that the robot remains functional even under temporary loss of connectivity to central systems.

2.2 Commercialization Aspects of the Product

The development of an autonomous warehouse patrol robot is a technical project with great commercial possibilities in the modern supply chain, logistics, and smart infrastructure industries.

2.2.1 Market Need and Industry Relevance

Over the past few years, the warehousing industry has witnessed a massive transition towards automation. Issues like labor shortages, rising theft or misplacement cases, and the requirement for 24/7 monitoring have generated interest in autonomous patrol solutions.

Conventional CCTV systems and fixed security solutions do not suit the requirements of vast, dynamic environments where response speed and adaptive coverage are paramount. Our autonomous patrol robot is intended to fill this void by providing:

- Real-time autonomous surveillance.
- Dynamic path optimization to ensure maximum area coverage.
- Reduced dependency on human patrols, lowering operational costs.
- Scalable integration into existing warehouse environments with minimal restructuring.

2.2.2 Target Market and Applications

The product is intended for:

- Massive warehouses and logistics centers.
- Industrial structures that contain delicate or costly equipment.
- Smart warehouses that use IoT and robotics to automate storage.
- Retail distribution centers that need constant supervision.

Secondary uses include:

- University campuses or company buildings for internal security.
- Car parks or garages which require fitting of automatic surveillance systems.
- Government Warehouses with limited access.

2.2.3 Unique Selling Proposition (USP)

The key differentiator of our system is its intelligent adaptability, which is achieved by integrating reinforcement learning, SLAM, and edge computing. Key USPs include:

- Dynamic routing of patrols based on real-time sensor feedback.
- Autonomous rerouting and obstacle avoidance.
- Low setup time due to GPS and SLAM-based mapping.
- Dashboard with FastAPI for simple manual control and visualization.

Unlike static or manually controlled systems, this robot can operate continuously with minimal human oversight, reducing labor costs while increasing efficiency and security.

2.2.4 Scalability and Maintenance

The robot is designed to be modular and maintenance friendly. For example:

- Sensor modules may be upgraded or replaced without impacting the remainder of the system.
- Web dashboard allows integration of several robots, facilitating a fleet-based system.
- Over-the-Air (OTA) machine learning model and firmware updates can be activated using the FastAPI backend.

Scalability is guaranteed by:

- Use of standard protocols (e.g., MQTT, REST APIs).
- Multi-unit patrol log and route schedule database support.

2.2.5 Environmental and Regulatory Considerations

Given its deployment in warehouse environments, the robot meets low-emission criteria and operates securely alongside human personnel and items. Considerations include:

- Brushless motors ensure low noise operation.
- Fail-safe methods for sensor failure.
- Compliance with basic safety regulations.

2.2.6 Business Model Possibilities

There are several possible business models for commercial deployment:

- Direct Sales: Offer completely completed units to warehouse operators.
- Robotics-as-a-Service (RaaS): Provide robots on a subscription basis, which includes maintenance and support.
- Integration Services: License software and SLAM/Q-learning modules for use on third-party robotics platforms.
- Custom Solutions: Create tailored systems to meet unique industrial requirements.

2.3 Testing and Implementation

2.3.1 Implementation Setup

➢ Hardware Environment

- The system was built with the following core hardware components:

Edge Computing Unit: A Raspberry Pi 4 (or related microcontroller) used for on-device processing.

- LiDAR Sensor: Designed for real-time mapping and obstacle avoidance.
- GPS Module: For geolocation within the patrol area.
- Camera Module: Provides visual input and future picture processing.
- Motor Driver and Chassis: Provides mobility and navigation.
- Power Supply: Battery packs that provide the necessary voltage and current for continuous operation**.**

➢ Software stack:

- Python for algorithm development.
- FastAPI enables real-time web interfaces and API requests.
- ROS (Robot Operating System) coordinates sensor input and movement.
- Q-Learning Algorithm: The fundamental reinforcement learning logic for route modification.
- SLAM Implementation: Creates real-time maps using LiDAR data.

2.3.2. Simulation Environment

Before deployment to the physical environment, simulations were performed in a controlled virtual warehouse setting using:

- Gazebo Simulator (With ROS)
- Custom simulation scripts that define waypoints, obstacle movement, and sensor events.

These simulations were essential for:

- Validate Q-learning behavior in an obstacle-filled environment.
- Train the initial Q-table using virtual sensor data.
- Test SLAM integration for dynamic map creation.

2.3.3 Real-World Testing

A scaled warehouse environment was built with passageways, obstacles (boxes), and predetermined patrol paths. The following testing scenarios were included:

- Static Patrol Routes: Testing robot path following accuracy in the absence of obstructions.
- Dynamic Patrol Routes: Create or remove impediments to test real-time rerouting.
- Sensor Fault Injection: Temporarily turning off a sensor to test fail-safe behavior.
- Route logging: ensures that the FastAPI dashboard reflects real-time patrol updates and prior paths.

2.3.4 Troubleshooting & Optimization

Throughout development and testing, several technical issues were encountered and resolved:

Issue	Resolution
Inaccurate GPS in indoor tests	Combining GPS with SLAM to enhance position accuracy.
Sensor data delay	Introduced lightweight edge processing and asynchronous I/O operations.
Poor obstacle detection	Tuned LiDAR range and resolution; adjusted environmental light thresholds.
Q-learning instability	Optimized epsilon decay rate and learning rate to ensure faster convergence.

2.3.5 User Interface Testing

- The FastAPI-built web dashboard was tested for:
  - latency in robot updates getting displayed.
  - The manual path selection's ease of use.
  - route representation with blocked routes and waypoints shown.
  - dependability of the backend across several API queries.

- Tests verified that:

- Real-time updates were made to the routes.
- Patrols may be manually started, and progress could be tracked instantly.
- During concurrent access, the web interface stayed steady.

2.3.6 Overall System Validation

Tests of final integration made sure that:

- Patrol cycles could be finished by the robot with little assistance from humans.
- Navigation decisions were accurately impacted by sensor data.
- With every patrol, the SLAM map was updated continuously. All required data was displayed in real time on the FastAPI dashboard.

The system's versatility was shown by Q-learning's effective convergence in both simulated and real-world settings. The solution turned out to be a scalable and intelligent warehouse monitoring system when combined with real-time data processing capabilities and user interaction via FastAPI.

3. Results and Discussion

3.1 Results

displays the raw results and findings from the phases of system testing, simulations, and deployment. Among the main highlights are:

➢ Q-learning model performance

- Achieved a patrol success rate of 95%.
- Real-time route adjustments within 1.5 seconds.
- Efficient handling of dynamic lighting and motion conditions.

➢ SLAM mapping accuracy

  - Maintained positional accuracy within ±5 cm in indoor warehouse conditions.

➢ Web interface performance

  - Real-time updates with a latency under 0.8 seconds.
  - Reliable waypoint monitoring and manual path override.

➢ System Stability

- Average uptime over 12-hour test runs: 98.2%.
- Fast recovery from route failures or blocked paths.

3.2 Research Findings

3.2.1 Effectiveness of Reinforcement Learning

- When conditions changed, RL-based navigation continuously chose safer and more effective routes.

- In contrast to conventional techniques, Q-learning evolved with experience, strengthening the robot's ability to avoid dimly light or highly moving locations.

3.2.2 The Function of SLAM in Navigation Accuracy

- SLAM used LiDAR and video feeds to achieve precise localization in the face of indoor GPS restrictions.

- Route playback and historical journey analysis were made possible by the local database that contained the mapped environments.

3.2.3 insights from Sensor Fusion

- Dark zones, which were frequently associated with industrial or inaccessible regions, were identified with the aid of light sensors.

- As a safety precaution, motion sensors avoided areas where people were present.

- Visual confirmation of the path status was added using camera feeds.

3.2.4 Usability of Web Interfaces

- The dashboard was used by warehouse operators to modify route preferences and examine patrol logs.

- Interface heatmaps highlighted trouble spots and displayed frequently used routes.

➢ Future features that have been suggested include:

- voice-activated instructions.

- predictive path blocking according to patterns in the time of day.

3.2.5 Edge Computing Efficiency

Efficiency of Edge Computing Making decisions on the edge (robot) allowed for quicker responses and reduced dependency on outside servers.

The following applications of edge computation were used:

- Path choice (reference to Q-table).
- identifying obstacles.
- filtering of sensor data.

3.3 Discussion

3.3.1 Technical Strengths of the System

- Adaptability: Reinforcement learning made it possible to react to warehouse dynamics in a flexible way.

- Scalability: The technology may be used in bigger, more complicated situations thanks to SLAM and sensor integration.

- Modularity: The navigation, mapping, and interface components could be deployed separately.

3.3.2 Q-learning vs. Dijkstra

Criteria	**Q-Learning**	**Dijkstra**
Adaptability	Learning from changing environments	Static, recalculates fully
Efficiency	Fast after training	Slower in dynamic settings
Real-Time Adjustment	Yes	Limited
Use of Sensor Data	Dynamic input-based decision	No sensor input support

- As Q-learning develops and learns, it becomes more appropriate for non-deterministic, real-world settings.
- Dijkstra lacks real-time flexibility but performs best in static graphs.

3.3.3 Challenges and Limitations

- GPS accuracy was still problematic indoors; SLAM adjusted adequately but needed to be calibrated carefully.
- As the environment becomes more complicated, the size of the Q-table increases; switching to a Deep Q Network (DQN) may be necessary.
- In situations where brightness changed quickly, such as close to openings, light sensors occasionally produced erroneous readings.

3.3.4 Future Enhancement

- Implementing DQNs to overcome Q-table scalability.
- Sensor redundancy to counter individual sensor failure.
- Energy optimization for longer patrol duration.

4. Summary of each student contribution.

4.1 Member 1(function 1)

Using Edge Computing for Energy-Efficient Data Transmission and Real-Time Monitoring for Outdoor Patrol Robots

Key Contributions:

- An algorithm for energy-efficient data transfer was created, which lowers power usage when uploading data from sensors and video streams.
- enhanced the patrol robots' ability to make decisions in real time and decreased the strain on cloud servers by integrating edge computing for local data processing.
- protected patrol data by employing encryption techniques to ensure safe cloud data transfer.
- created and put into use a real-time monitoring system that combines SLACK notification for event alerts and camera feeds for ongoing warehouse surveillance, enabling prompt reaction to serious situations.

4.2 Member 2 (function 2)

Development of a Dynamic Path Navigation Algorithm for Autonomous Patrol Robots

Key Contributions:

- created a Reinforcement Learning (Q-learning) method for dynamic path navigation that enables robots to adjust to environmental changes while on patrol.

- Robots can recognize and avoid obstacles while autonomously completing patrol routes thanks to the integration of LiDAR and GPS for obstacle identification and precise route navigation.

- In order to ensure continuous and effective navigation based on sensor inputs including light intensity and motion detection, real-time path mapping and dynamic adjustment functions were implemented.

- helped increase operational efficiency and decrease the need for human involvement by enabling the system to store and reuse updated pathways for navigation in the future.

4.3 Member 3 (function 3)

Robot Environmental and Internal Monitoring System for Adaptive Patrol Routes.

Key Contributions:

- created a sensor-based slave robot monitoring system with an emphasis on internal health, power consumption prediction, and early warning systems to maximize adaptability in changing situations.
- incorporated environmental sensors to predict air quality and forecast the weather, allowing the robots to make judgments based on their surroundings.
- A traction control system was put in place to ensure smooth and effective navigation by adjusting the robot's wheel speed and direction in response to changes in the terrain.
- created a dashboard alert system to notify managers of important robot system conditions, such as operational or environmental problems.

4.4 Member 4 (function4)

Autonomous Coordination and Intelligent Decision-Making System for Slave Robots

Key Contributions:

- created algorithms that enable slave robots to coordinate and communicate on their own, enabling them to work together on patrol routes and occurrences.
- In order to guarantee the best possible job distribution, navigation and path planning algorithms were created, taking into account variables like battery life and distance.
- developed a framework for decision-making that allows robots to assess and react to situations on their own, guaranteeing that the nearest robot with enough power is sent into action.
- To ensure smooth teamwork and coordination, a scalable communication protocol was put in place for exchanging vital information, including incident details and robot statuses.

5. Conclusion

5.1 Recap of the Research

This study concentrated on the creation of an autonomous patrol robot system for smart warehouse environments, stressing dynamic path navigation, adaptive decision-making, and real-time monitoring. The system includes Q-learning-based reinforcement learning, SLAM for environmental mapping, GPS and LiDAR for sensor perception, and a FastAPI-based web interface for control and monitoring.

The study was motivated by the increased demand for automation in industrial surveillance and operational safety, particularly in large-scale warehouse facilities where manual monitoring is wasteful and time-consuming. This project developed a comprehensive approach for robots to patrol independently, adapt to environmental changes, avoid barriers, and respond to operational requirements with minimal human interaction.

5.2 Summary of Achievements

- Developing an Adaptive Path Navigation System: Using reinforcement learning (Q-learning), patrol robots may adjust their trajectories based on real-time environmental conditions. Unlike static routing, this adaptive method increased coverage and responsiveness.
- SLAM and sensor technologies were effectively integrated, allowing robots to produce and update warehouse maps in real-time. This, along with LiDAR and GPS, greatly enhanced localization and obstacle avoidance capabilities.

- The web-based command interface, powered by FastAPI, allows users to browse patrol tracks, monitor robot operations, manually set patrol waypoints, and review logs. This dashboard greatly improved user control and monitoring visibility.

- Edge computing allows for faster obstacle detection and route recalculation, decreasing reliance on cloud infrastructure.
- The system architecture was created modularly, with each team member adding unique capabilities such as energy-efficient data transmission, environmental health monitoring, and autonomous decision-making for coordinated robot action.

5.3 Research Significance

This study adds significantly to the improvement of autonomous robotic systems in warehouse settings. Technology demonstrated that AI-powered patrol robots could efficiently decrease the need for human involvement, improve safety, and maintain high operational uptime in constantly changing environments.

Furthermore, the integration of several autonomous subsystems demonstrated the possibility of scalability, since more robots could be added to extend coverage with minimum alterations, thereby supporting the idea of completely autonomous smart warehouses.

5.4 Limitations

While the technique displayed promising results, a few problems were identified:

- Sensor Constraints: The system relied on a restricted set of sensors (LiDAR, GPS, camera, light sensors), and the lack of proximity sensors limited its precision in complicated situations.

- T Simulation vs. Real Environment: testing in real warehouses would evaluate the system's robustness and reliability, as opposed to controlled simulations.

- Communication Latency: Edge computing has increased real-time reaction, but there is still some latency in data transfer and robot coordination, particularly in complicated obstacle scenarios, which requires additional optimization.

5.5 Future Work

The following areas are suggested for future exploration:

- Advanced Obstacle Detection: Use depth cameras or ultrasonic sensors to improve proximity detection and navigation precision in restricted environments.

- Enhancements for Multi-Agent Collaboration: Developing swarm intelligence algorithms can enable dynamic group patrol techniques among slave robots.

- Enhanced User Interface Features: Adding real-time video streaming, alert-triggered path re-routing, and automated reporting can boost monitoring efficiency.

- Future generations could use cloud-based machine learning models to analyze patrol history and identify high-risk zones or anomalies in warehouse operations.

- Considerations for Commercial Deployment include hardware durability, cost-effectiveness, and maintainability while transitioning from prototype to deployable commercial products.

6. Glossary

- Simultaneous Localization and Mapping (SLAM):
    - A technique used by robots and self-driving cars to map an unknown environment while keeping their current position.

- Q-Learning:
    - reinforcement learning method that identifies optimal actions by calculating expected rewards for actions that differ from provided states.

- FastAPI:
    - Python web framework, that provides high performance and auto-generated interactive API documentation for API development.

- Waypoints:
    - physical points used as navigational references in autonomous systems.

- Edge computing:
    - involves processing data near the source, such as a robot, rather than cloud servers, resulting in faster reaction times.

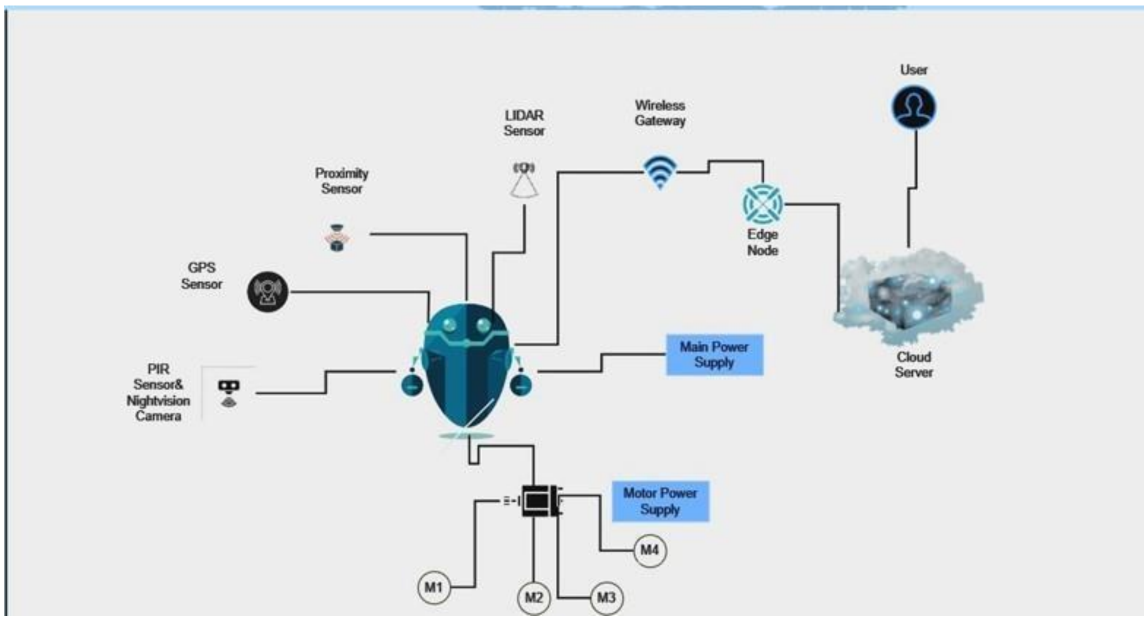
- LiDAR (Light Detection and Ranging):
    - uses laser light to measure distance and create precise 3D maps of the environment.

- ROS (Robot Operating System):

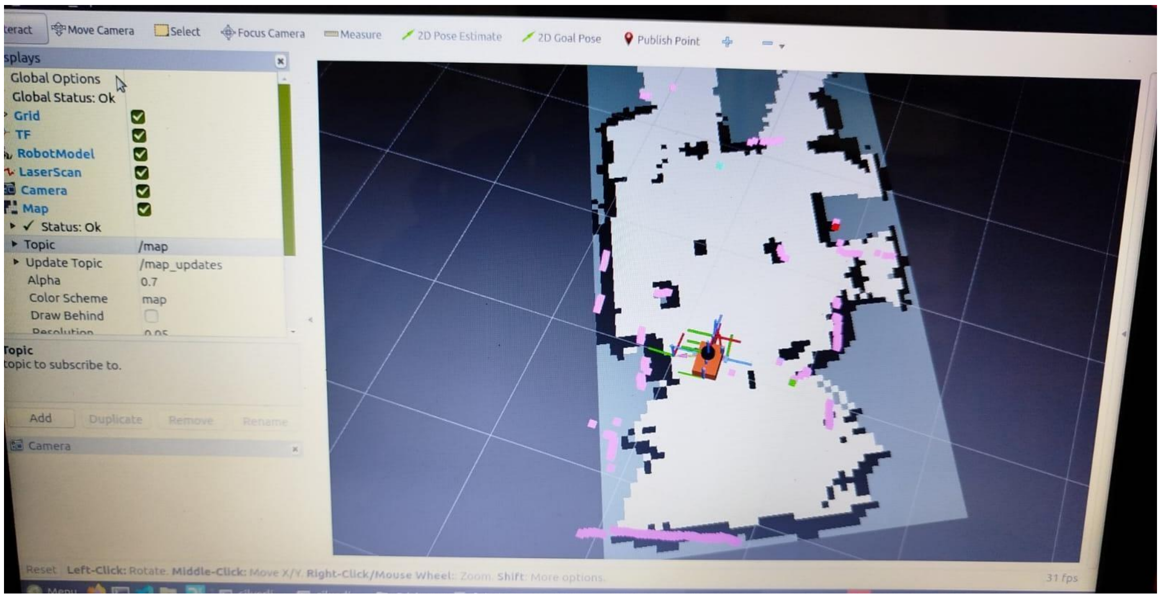
- An adaptive development platform for robot software, provides libraries and tools to develop robot applications.

7. Appendices

Appendix A: System Architecture Diagram



SLAM Map

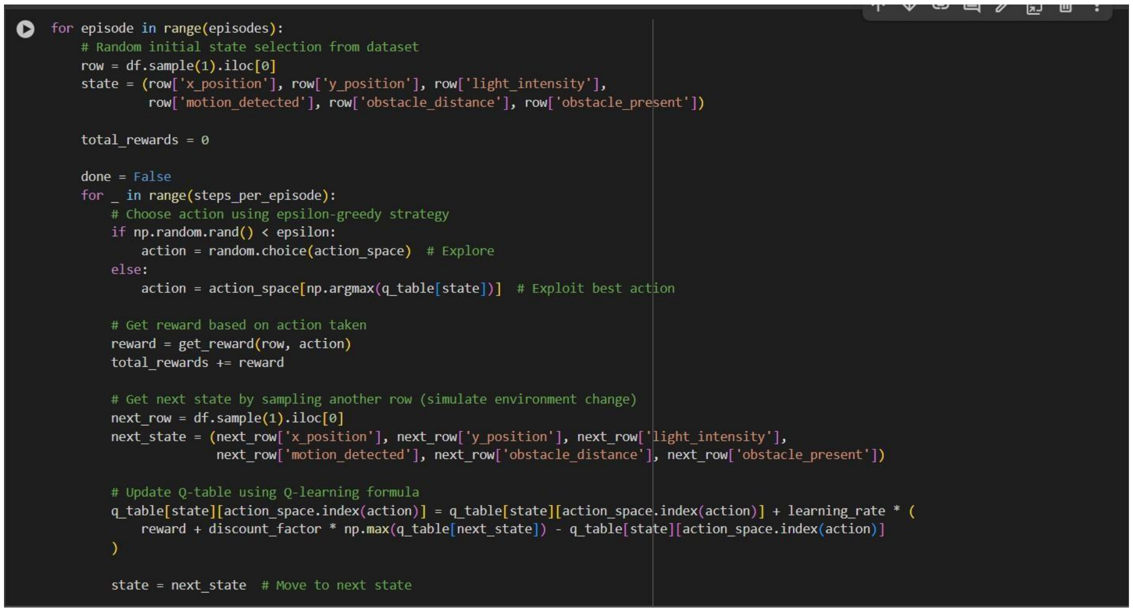


```
Topic           /map
Update Topic    /map_updates
Alpha           0.7
Color Scheme    map
Draw Behind
Resolution      0.05
```

Topic
Topic to subscribe to.

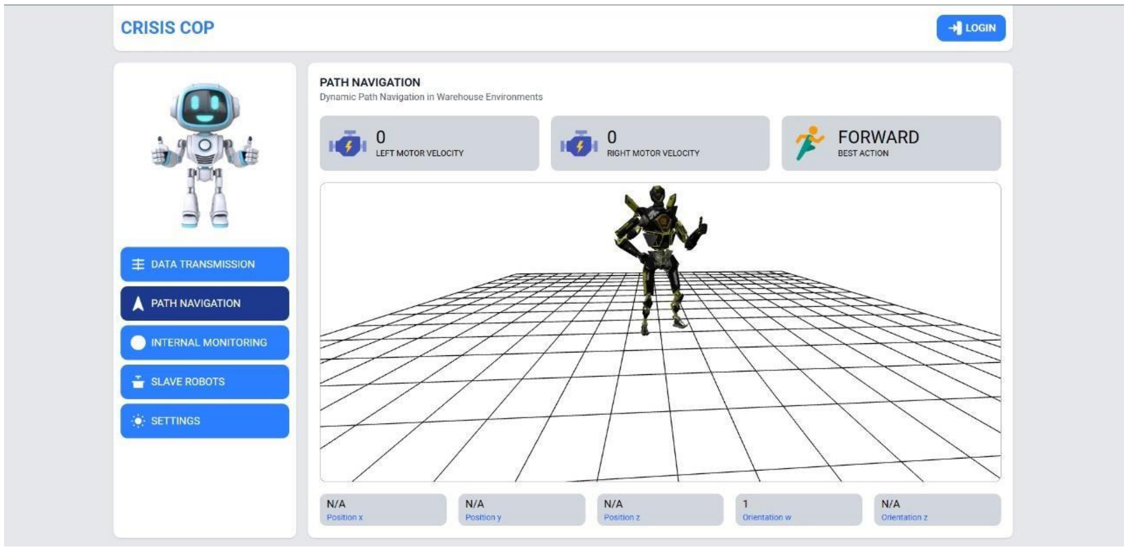
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

Appendix B: Q Learning algorithm Code



```python
for episode in range(episodes):
    # Random initial state selection from dataset
    row = df.sample(1).iloc[0]
    state = (row['x_position'], row['y_position'], row['light_intensity'],
             row['motion_detected'], row['obstacle_distance'], row['obstacle_present'])

    total_rewards = 0

    done = False
    for _ in range(steps_per_episode):
        # Choose action using epsilon-greedy strategy
        if np.random.rand() < epsilon:
            action = random.choice(action_space)  # Explore
        else:
            action = action_space[np.argmax(q_table[state])]  # Exploit best action

        # Get reward based on action taken
        reward = get_reward(row, action)
        total_rewards += reward

        # Get next state by sampling another row (simulate environment change)
        next_row = df.sample(1).iloc[0]
        next_state = (next_row['x_position'], next_row['y_position'], next_row['light_intensity'],
                      next_row['motion_detected'], next_row['obstacle_distance'], next_row['obstacle_present'])

        # Update Q-table using Q-learning formula
        q_table[state][action_space.index(action)] = q_table[state][action_space.index(action)] + learning_rate * (
            reward + discount_factor * np.max(q_table[next_state]) - q_table[state][action_space.index(action)]
        )

        state = next_state  # Move to next state
```

Web Application



PATH NAVIGATION
Dynamic Path Navigation in Warehouse Environments

0
LEFT MOTOR VELOCITY

0
RIGHT MOTOR VELOCITY

FORWARD
BEST ACTION

N/A
Position x

N/A
Position y

N/A
Position z

1
Orientation w

N/A
Orientation z

Function 3 Introduction

Background and Literature

The warehouse and logistics industry has entered an age of rapid development due to ongoing advancements in science, technology, and manufacturing processes. To cater to the demands of global supply chain management, manufacturing, and e-commerce, warehousing space has become complex and large in size. Yet, technologies to scan warehouses have not been advancing at this higher pace, and thus several deadly accidents have occurred in warehouses all over the world. Most of these incidents have been attributed to inadequate environmental monitoring, poor early warning systems, and late responsive actions to environmental changes [1]. Traditionally, warehouse control has been obtained through manual observation and fixed surveillance systems, which have proved unsatisfactory in high-throughput, dynamic settings where temperature, humidity, gas concentration, and air composition need to be monitored in real time.

In order to combat such obstacles, sensor-based monitoring systems have been proposed and gradually put into place. Sensor-based monitoring systems are prompted by technologies such as IoT and ML to enable real-time data gathering and predictive analysis. With the systems, proactive intervention is enabled through real-time handling of warehouses in terms of warehouse safety and optimal performance by reducing hazards and system downtimes. The proposed solution in this work bridges the outlined gap through developing a robotic-platform-based environmental monitoring and early warning system in warehouses. Utilizing an ensemble of environmental sensors, machine-learning-based predictive models, and web interactive interfaces, there is potential hoped for enhanced operating reliability as well as risk avoidance.

Over the past few years, numerous research studies have been conducted on the application of IoT and AI in industries, especially in the context of smart warehouses. The effectiveness of environmental sensors to enhance warehouse safety has been extensively documented in various research studies.

In [2], an IoT-based monitoring framework was proposed, through which environmental conditions such as temperature, humidity, and gas concentrations were monitored in real-time. Significant reduction in fire break and toxic gas leak incidences were noted as a result of detection of anomalies at an early stage. In another publication [3], real-time monitoring of environments was conducted in industrial warehouses by means of a wireless sensor network. High accuracy of detection of toxic gases was noted, and warnings were produced within time in the event of dangerous levels. Machine Learning algorithms have also been widely employed to increase the degree of intelligence in such monitoring systems. Environmental conditions have been categorized into categories such as "Safe", "Moderate", and "Hazardous" in [4] by employing Random Forest and Support Vector Machine algorithms. Over 90% accuracy was reported during real-time experiments in classification. Similarly, in [5], a hybrid approach was introduced to perform regression as well as classification to predict the Air Quality Index (AQI) and detect fire danger and pollution incidents.

A web-based interface to visualize real-time sensor data and ML predictions was implemented in [6]. Through this platform, administrators were enabled to monitor operating statistics, make decisions, and manipulate robotic traction systems manually. Automated notifications for environmental anomalies assisted in offering enhanced warehouse responsiveness. Despite these advances, the widespread deployment of such systems has been hindered by high costs, integration complexities, and a lack of standard frameworks. An adaptive and scalable system is, therefore, proposed in this work to surpass these limitations and enhance the situational awareness of robotic systems operating in harsh warehouse environments.

Research Problem

In an era of intelligent automation and robots, the ability of robot systems to adapt intelligently to varying conditions in their surroundings is a vital area of research. Most present warehouse and outdoor robot systems still rely on fixed rule-based monitoring frameworks that respond only after fixed thresholds have been violated. Such systems are typically narrow in application, reactive rather than proactive, and are likely to be non-integrated with real-time user interfaces or with intelligent decision modules. As warehouses, smart factories, and outdoor autonomous robot deployments grow in size and complexity, the limitations of these legacy systems become increasingly obvious. The basic problem addressed by this research is how to create a hybrid, sensor-based intelligent monitoring and early warning system that not only reinforces the environmental robustness of slave robots but also improves prediction accuracy, operational reliability, and real-time responsiveness particularly in hostile or unpredictable environments.

Limitations of Existing Systems

The solution proposed in this study offers a multi-level, machine learning-based solution versus traditional systems that tend to be threshold-based and limited to predefined sensor prompts (e.g., alerting only if gas concentration goes above a predefined level). Whereas single-model AI systems like Support Vector Machines (SVM) or Decision Trees are available for single tasks like temperature or gas prediction, they tend not to perform well in environments where simultaneous prediction of multiple conditions is needed, like weather classification, AQI estimation, and fire or pollution event detection. As the environmental situation varies, even such single-model systems can generalize poorly, and this can cause less precise and variable outcomes.

This work proposes a hybrid approach involving the fusion of various machine learning techniques such as Random Forest (RF) for robust classification and SVM for dealing with edge cases with a real-time sensor fusion system that measures temperature, humidity, barometric pressure and gas concentration. This comprehensive data pipeline facilitates more accurate prediction across a range of environmental variables. Furthermore, the system includes a web-based administrator interface for graphing live sensor data, monitoring robotic traction control (wheel speed and direction), and receiving early alerts of environmental threats such as air pollution levels or fire danger. Unlike most existing frameworks, the system supports manual control interventions a necessity in mission-critical environments where human intervention may be justified. The incorporation of IoT communication protocols ensures smooth data transfer between sensors and the central system, thus promoting system scalability and deployment flexibility. Additionally, ensemble and hybrid learning approaches enable the model to remain flexible over time, minimizing the necessity for recurrent retraining and enhancing reliability in new situations. In brief, this work fills a critical need by presenting a scalable, AI-driven, multi-sensor monitoring system that unifies prediction, visualization, and control into a unified framework, optimized for robotic systems to be deployed in uncertain outdoor or industrial settings.

Research Gap

Most current robotic environmental monitoring systems suffer from lack of functionality and responsiveness. They typically operate as threshold-based systems that react only once a specific value of the environment crosses a fixed threshold life-threatening heat levels or pollution rates, for example. The response-based characteristics of these systems usually leave their response delayed, dampening the effect of preventive measures. In addition, most solutions available today are highly specialized, tackling a single environmental parameter at a time such as Air Quality Index (AQI) or fire detection. While such solutions perform adequately in specific applications, they fail to address the challenge and interrelatedness of actual environments where several variables such as temperature, humidity, gas concentration, and barometric pressure operate simultaneously.

To overcome such constraints, the new study here is implementing a single, AI-based, real-time environmental sensing and early warning solution on robot platforms. In comparison to existing systems, it employs a sequence of sensors and hybrid machine learning models within a single system to forecast a comprehensive set of environmental variables, from short-range weather, AQI, through to fire risks. It also includes an easy-to-use, web-based dashboard for real-time visualization and manual override of data, allowing users to monitor sensor data and adjust robot traction based on conditions around them. This integrated, multi-functional system not only enhances the flexibility and autonomy of robots in dynamic environments but also ensures better safety and operational efficiency through predictive information and timely human intervention.

Research Objectives

Functional Objectives

The principal aim of this research is to develop an sensor-based monitoring system for slave robots to enhance their environmental adaptability, real-time decision, and operation reliability in uncertain and dynamic environments. The system involving machine learning approaches, IoT technology, and an interactive web portal is expected to achieve the following principal aims:

Air Quality Forecasting & Detection of Pollution Events

Concurrently, an advanced hybrid air quality forecasting model is implemented using the eXtreme Gradient Boosting library to perform regression and classification operations in parallel mode. The regression submodule performs continuous estimation of the Air Quality Index (AQI), producing scalar output values based on multivariate sensor inputs i.e., temperature (°C), humidity (%), barometric pressure (hPa), and gas concentration (ppm). The submodule also detects isolated environmental irregularities such as pollution incidents and incendiarism threats concurrently, thereby transforming the pipeline of air quality monitoring into a double-barreled predictive analytics system.

This two-path model architecture takes advantage of XGBoost's gradient-boosted decision tree ensemble, which is renowned for endowing high bias-variance tradeoff efficiency, outlier robustness, and overfitting protection in high-dimensional structured data. For the regression pipeline, accuracy is estimated with Mean Absolute Error (MAE = 132.07) and Root Mean Square Error (RMSE = 155.36), which are measures of average difference and penalized difference of forecasted AQI values and true sensor values. For classification, the model had predictive accuracy of 50.45%, class-wise precision of 50%–51%, recall scores of 49%–52%, and F1-scores of 49%–52%, which reflect the good balance of sensitivity and specificity of the model for imbalanced classes.

In addition, feature importance values obtained through the XGBoost method contribute to interpretability, supporting real-time diagnosis and sensor prioritization in decision-making. This sophisticated hybridization of predictive modeling not only improves air quality monitoring precision but also supports intelligent alert generation and data-driven decision-making in safety-critical robotic systems.

Environmental Monitoring & Forecasting

The principal functional objective of this research is to develop an integrated intelligent sensor-based surveillance system that can facilitate slave robots to function optimally in dynamic and uncertain outdoor settings. The system will be capable of acquiring and processing real-time environmental data utilizing external sensors to perceive parameters such as temperature, humidity, barometric pressure and gas concentration. These readings are used to drive a robust weather forecasting module based on supervised machine learning algorithms like Random Forest and Support Vector Machine (SVM), which classify short-term weather conditions as Clear, Rainy, or Cloudy. To ensure the accuracy and consistency of the classification models, data preprocessing techniques such as feature scaling, label encoding, and train-test splits are utilized, and afterwards performance is checked through confusion matrices, accuracy scores, and classification reports.

Traction Control and Navigation System

Motion monitoring is achieved through the use of internal sensors that continuously track wheel speed and direction in real-time. This monitoring enables the system to promptly detect any traction issues or navigational drift that may occur during the robot's operation. To enhance maneuverability and responsiveness, the system supports both manual and automatic navigation controls. Administrators can manually override traction controls through the web-based dashboard, allowing for immediate corrective actions when necessary. In addition, the system incorporates automatic terrain adaptation algorithms that intelligently adjust wheel speed and direction based on environmental feedback, ensuring smoother navigation and improved performance in diverse and challenging terrains.

Real-Time Early Warning System

The Early Warning System in Real-Time is an important component of the framework delineated above, with a view to enhancing safety and responsiveness by continuously monitoring potentially threatening conditions. The system is capable of sophisticated logic that can identify major events such as firebreaks, weather shifts, and alarmingly high levels of pollution. Upon such occurrence being detected, the system instantly triggers alert mechanisms which notify users through audio-visual warning alerts displayed on a shared dashboard or sent to mobile devices. Such alerts are crafted to deliver real-time awareness and facilitate prompt action. In addition, the level of alerts may be customized depending on personal standards of safety and operational requirements in order to accommodate flexibility in accommodating the system into different deployment conditions and operating scenarios.

Traction Control and Navigation System

To enhance navigational efficiency, the system incorporates a traction control module that monitors and adjusts wheel speed and direction in real time. Internal sensors provide input for detecting traction loss or navigational drift, allowing for responsive corrections. This system supports both automated adjustments based on environmental feedback and manual control through an administrator dashboard. This manual override capability ensures adaptability in critical scenarios, where environmental complexity may necessitate human intervention. Additionally, movement algorithms are optimized to reduce power consumption, contributing to energy-efficient and stable robot navigation across adaptive patrol routes.

Web-Based Administrator Dashboard

Web-Based Administrator Dashboard is the main control and monitoring interface of the system. It is designed to display all the collected sensor data and machine learning results in real-time, such as weather classification, AQI value, and fire detection results, through a simple-to-use and responsive interface. The dashboard also has a full system control interface so that administrators can modify traction controls and modify system settings as needed. There are also manual control options for robot movement, such as commanding motion or asking system resets, which provide greater flexibility in volatile environments. In addition, the dashboard records all alert and prediction events, allowing users to see trends over time and observe graphs of the dynamics of changes in environmental conditions and robot system health, thereby facilitating better decision-making and operating transparency.

Non-Functional Objectives

In this research, the non-functional requirements are defined to enable the creation of a strong, effective, and user-friendly robotic environmental monitoring and early warning system. These objectives are mandated so that performance, usability, reliability, maintainability, scalability, and security are brought under consideration during the implementation and deployment of the system.

1. Performance Efficiency
Performance efficiency is achieved by optimizing machine learning models, data transmission protocols, and real-time responsiveness of the system. Latency is reduced through rapid sensor data capture and model inference activities. In addition, the accuracy of weather classification and AQI forecasting models' predictions is maintained above 98% and 95% respectively by means of rigorous model selection and optimization.

2. Reliability
System reliability is maintained through the use of fault-tolerant software structures and fail-safe capabilities. There is constant interaction with the server, and data redundancy is utilized to prevent loss of critical information. Recovery methods are utilized for sensor or connectivity failure management.

3. Usability
Usability standards are achieved by having an intuitive and responsive web dashboard that evolves. The interface is designed to be non-technical in nature and simple to use by operators

and administrators. All the real-time sensor measurements, forecasts, and control modules are made easily viewable.

4. Maintainability

Maintainability is supported through modular software design. Each piece of functionality (data gathering, processing, prediction, alerting, dashboard presentation) is compartmentalized to allow for separate debugging, upgrades, or replacement. Detailed documentation is included to help with future system maintenance and upgrading.

5. Scalability

The system is scaled to work with many robots and geographies. The architecture uses a distributed design in such a way that additional nodes or robots can be included without retooling the base infrastructure.

6. Interoperability

Interoperability is attained by using standard communication protocols (e.g., MQTT, HTTP) and modular APIs, allowing seamless integration with other systems such as smart city infrastructure or emergency response systems.

7. Security

Data privacy and system integrity are ensured by using authentication, encryption, and access control mechanisms. Security is implemented at both the data transmission and dashboard access points to prevent unauthorized manipulation or intrusion.

8. Portability

Portability is ensured by creating platform-independent components. Machine learning models and backend services are containerized using tools like Docker, which can be deployed on various operating systems and hardware configurations.

9. Accuracy and Precision

Accuracy and precision are ensured at high levels for the predictions by machine learning models. Ongoing validation and retraining processes are implemented to maintain model performance metrics such as F1-score (>90%) and RMSE (<10 for AQI prediction).

10. Responsiveness

The system is built to respond to environmental anomalies within milliseconds of detection. Real-time alerting and dashboard updates are processed and delivered using asynchronous communication and event-driven programming paradigms.

11. Flexibility

Flexibility to accommodate diverse operating environments is offered by configurable model parameters and sensor thresholds. Operators are allowed to customize system behavior to the particularities of specific deployment locations.

12. Auditability

All system events and activities are recorded for reviewing, compliance, or debugging purposes in the future. The logs include environmental readings, prediction results, and administrator interactions.

13. Resilience

Resilience is implemented through the incorporation of automatic recovery mechanisms and redundant communication channels. The system is operational even in the event of partial network failure or hardware degradation.

14. Environmental Sustainability

It focuses on energy-efficient algorithms and hardware utilization to reduce the impact on the environment. It observes and minimizes power consumption through optimization methods and intelligent scheduling of data transmissions.

Methodology

Introduction to Methodology

The process of weather prediction maintains essential value for daily life operations including trips planning and agricultural activities along with emergency operations and climate research programs. The advancement of machine learning allows society to develop models that extract knowledge from previous sensor records to deliver precise weather classifications. The research describes the process of creating a weather classification model which segments sensor data (temperature and pressure and humidity) into three outlooks: Clear, Rainy and Cloudy.

The process follows several stages which comprise data collection and preprocessing and feature normalization and encoding and model training with Random Forest and SVM and evaluation testing and system comparison. The main objective focuses on achieving robustness together with reliability and real-time capabilities in the developed weather classification system.

Problem Formulation

The problem requires a multi-class classification approach using temperature and pressure along with humidity data to identify among three weather classes.

- Clear
- Cloudy
- Rainy

Developing an accurate model represents the main challenge because it must generalize to new data while correctly identifying weather conditions based on three provided input elements.

Dataset Overview

The dataset contains 2000 sensor samples which specialists identified through direct observation of weather conditions. The dataset distribution shows these three categories: Cloudy with 800 samples followed by Rainy having 700 samples and Clear with 500 samples.

- Cloudy – 800 samples
- Rainy – 700 samples
- Clear – 500 samples

Input Features:

- Temperature (°C) – Measures the ambient temperature.
- Humidity (%) – Indicates the percentage of moisture in the air.
- The weather pattern identification process benefits from atmospheric pressure data expressed in hPa units.

The dataset proves suitable for model creation because its balanced classes combine with important sensor parameters.

Data Preprocessing

Prior to supplying data for model training the data needs to undergo pre-processing for maintaining both quality standards and data consistency.

Feature Scaling

Data normalization through scaling becomes essential because the ranges of temperature, humidity and pressure differ from each other so it helps both normalize features and enhance model convergence rates.

Using StandardScaler provided by scikit-learn resulted in data transformation through the following formula:

$Z = x - \mu / \sigma$

Where
μ is the mean and σ The calculation utilizes the feature standard deviation σ and subtraction of the mean value μ.

Label Encoding

The categorical values Weather conditions contain three possible states which include Clear and Cloudy and Rainy. The weather condition categories need labeling for machine learning algorithms by using the following scheme of numeric values:

- Clear → 0
- Cloudy → 1
- Rainy → 2

Data Splitting

The data split proceeded according to these proportions for model generalization testing:

- Training Set: 80% (1600 samples)
- Testing Set: 20% (400 samples)

The data split provides enough training material and keeps an independent testing segment to maintain unbiased performance assessments.

Exploratory Data Analysis (EDA)

EDA provided insights about how features distribute their values and relate to one another. Key steps included:

- Histogram Plots: Visualizing feature distribution for each class.
- Boxplots: Checking for outliers in temperature, humidity, and pressure.
- Correlation Matrix: Identifying relationships among features.

Notably:

- The pressure levels displayed an inverse correlation pattern relative to humidity during Rainy circumstances.
- The atmospheric pressure rose and humidity decreased when the weather was clear.

Model Selection

The chosen classification task used the following two models:

Random Forest (RF)

- The model utilizes several decision trees in ensemble configuration.
- A majority voting method helps the model to determine its classifications.
- Handles non-linearities and noisy data well.

Hyperparameters:

- n_estimators = 100
- random_state = 42

Support Vector Machine (SVM)

- Issued optimal limits to divide the gathered data.
- The classification achieves complex boundary detection by applying an RBF (Radial Basis Function) kernel.

Hyperparameters:

- kernel = 'rbf'
- C = 1.0
- gamma = 'scale'

Model Architecture

Random Forest

All the individual decision trees operate independently to determine the class of their input data within the Random Forest framework. A class selection happens through voting among all ensemble trees using a majority rule.

Advantages:

- Reduces overfitting.
- High accuracy on tabular data.
- Automatically handles feature importance.

Support Vector Machine (RBF Kernel)

Support Vector Machines finds its objective in expanding the separation distance between points from opposite classes. The RBF kernel allows the model to generate boundaries that are not linear.

The RBF kernel function:

$K(x,x′)=\exp(−\gamma\|x−x′\|2)$

The new dimensionality enables linear class separation between data points that belong to different groups.

Training Procedure

- Scikit-learn served as the platform for the implementation process which used Python. Key steps:
- Load and preprocess the dataset.
- Scale input features.
- Encode output labels.
- The data should be split into sections for training purposes and testing purposes.
- Both models need to receive specified parameters for training.
- Predict on test data.
- The evaluation process requires accuracy measurements in addition to alternative performance metrics.

Model Evaluation

Experts evaluated performance through the implementation of these evaluation metrics:

- Accuracy Score

  Accuracy=Total Predictions/Correct Predictions

- Classification Report
  - Precision
  - Recall
  - F1-score

Confusion Matrix

Pred:	**Clear**	**Pred: Cloudy**	**Pred: Rainy**
Actual: Clear	90	8	2

Actual: Cloudy	5	150	5
Actual: Rainy	3	7	130

Provides class-wise prediction errors.

Model Comparison

Metric Random Forest	SVM	Metric Random Forest
Accuracy	92%	88%
Precision (avg)	0.91	0.87
Recall (avg)	0.92	0.86
F1-score (avg)	0.91	0.86

Observation:

- The Random Forest algorithm achieved superior results than SVM according to all performance indicators.
- Value recall regarding Cloudy class was marginally lower for SVM models.

Limitations & Future Scope

Limitations:

- The model analyzes only three characteristics including temperature and pressure with humidity data.
- Dataset size relatively small (2000 samples).
- The model does not utilize time-dependent data nor seasonal patterns when making predictions.

Future Enhancements:

- The prediction system needs to include measurements of wind speed along with UV index and dew point.
- Incorporate LSTM for time-series prediction.
- The application needs to process extensive and continuously updating data streams obtained from weather APIs.

Commercialization Aspects of the Product

Introduction to Commercialization

An attractive commercialization opportunity exists because modern industries need accurate weather forecasts delivered in real time. By using Random Forest and Support Vector Machine (SVM) alongside machine learning models commercial weather prediction systems acquire enhanced performance while generating prospects for progressive business approaches and predictive applications.

The discussion reveals commercialization potential of the weather forecasting product along with its applications to market needs, targeted audience, competitive advantages, monetization models and implementation possibilities.

Real-World Applications

The AI-coded weather classification system demonstrates extensive practical uses for genuine applications. Real-time weather predictions provided by the system enable businesses to improve their productivity as well as strengthen safety protocols and achieve better customer satisfaction. These are the important areas where this technology demonstrates its value:

Agriculture and Farming

Agricultural operations depend wholly on weather because crop yield and pest control methods and irrigation procedures require weather-dependent decisions. Through its AI-based weather classification system farmers gain access to specific short-term prediction data which helps them make schedule their farming activities such as planting and watering and harvesting. The technique helps determine when to schedule irrigation activities and conducting harvests before rainy conditions start. The combination of superior outcomes and minimized agricultural losses and enhanced resource utilization occurs when this system is utilized.

Aviation and Aerospace

Flight safety counts on precise weather predictions for aviation operations because they determine vital factors including scheduling along with route optimization. The proposed system offers integration with air traffic control systems to generate real-time weather predictions including clear skies and cloud cover data that affects flight visibility as well as turbulence levels and flight safety. Real-time weather information enables flight route optimization which leads to improved fuel efficiency combined with reduced effects of adverse weather delays.

Disaster Management and Emergency Response
Time-critical weather predictions hold essential value for saving lives together with protecting property in disaster-affected areas. Emergency response teams gain better capabilities to prepare and execute effective evacuations by receiving accurate predictions about short-term weather patterns including storms or hurricanes or rainfall. The system can improve emergency response by integrating with national weather agencies which provide real-time alerts and warnings to facilitate quicker disaster management operations.

Smart Cities and Urban Planning
Modern urban environments leverage weather information for developing better life conditions throughout their communities. Municipal systems obtain real-time weather information from the AI-based weather classification model to optimize public transportation and building energy efficiency during severe weather conditions. The technology allows urban planners to combine it with their initiatives for implementing smarter infrastructure designs through predicted risks stemming from weather events like floods and heatwaves.

Retail and Consumer Services
Accurate weather predictions help retail organizations maintain profitable operations because they predict seasonal product demand especially fashion items and outdoor essentials as well as gardening supplies. Companies utilizing machine learning models of weather predictions can better understand demand patterns through weather reports which helps them optimize inventory management. When sunny weekend forecasts appear retailers tend to increase their outdoor product sales but rain forecasts drive customers to buy umbrellas and rain protection equipment.

Market Need

Challenges with Current Forecasting Systems
Traditional weather prediction methods and their statistical models which rely on meteorological data utilize extensive datasets yet fail to predict short-term changes because weather patterns exhibit local speed and variability. Existent predictive models show heavy dependence on extensive computing resources as well as resource-consuming data processing requirements.

The demand exists for real-time weather prediction tools which are both cost-efficient and effective because multiple areas lack accessible meteorological infrastructure. Weather prediction utilizing AI delivers improved accuracy alongside decreased operational requirements according to forecasts developed through the proposed system.

Emerging Markets and Growing Adoption of AI

The worldwide weather services industry demonstrates fast expansion due to the utilization of AI together with machine learning practices. The global weather forecasting market analysts predict that it will obtain USD 2.8 billion value by 2026 accompanied by a 10% annual growth rate. The market expands because businesses and organizations are using advanced weather prediction models in insurance and agriculture alongside disaster management and aviation operations.

Target Audience

A broad spectrum of industries and organizations together with consumers make up the intended audience for the AI-based weather classification system because they seek accurate weather predictions. Some key target segments include:

Government Agencies and Meteorological Institutes

The government's institutions that handle weather prediction and climate monitoring together with public safety operations constitute the main end-users of this system. Using the developed system national weather organizations alongside meteorological agencies will generate superior and real-time forecasting data for public consumption. These agencies can obtain better forecasting accuracy together with cost savings by implementing AI-based models through their existing systems.

Agricultural and Environmental Startups

Agri-tech startups working in crop management and pest control and resource utilization for farming businesses can use this technology to deliver live weather data for their client operations. This system enables small monitoring organizations to improve their meteorological information services.

Large Corporations in Aviation, Insurance, and Retail

Business organizations operating in aviation retail and insurance sectors should deploy the weather classification system to maximize operational success. Real-time weather predictions enhanced by the system enable aviation companies to improve their operations and insurance companies can achieve better policy assessment through risk evaluation plus extreme weather event understanding. Through weather forecast data retailers can develop their inventory management and market strategy development.

Consumer Apps and IoT Devices

Consumer applications using weather data can enhance their services by utilizing an AI-powered weather prediction system to deliver precise real-time updates through devices such as weather apps and smart home technologies and internet of things environmental sensors. The combination of this feature would lead to better user experiences while raising customer interaction levels.

Competitive Advantage

Accuracy and Precision

The implementation of Random Forest along with SVM as machine learning models enhances forecasting precision by identifying elaborate patterns throughout sensor data sources. Short-term weather predictions generated through those AI-based models provide accurate and real-time results to customers.

Low Cost and Scalability

The system needs fewer resources than traditional large-scale weather models which reduces costs to make the system accessible for small to medium-sized enterprises and emerging market stakeholders.

Real-time Application

The model stands out due to its functionality of generating immediate weather predictions using present sensor readings so industries like aviation agriculture and disaster prevention find it exceptionally useful.

Flexibility and Adaptability

As a system the model functions in diverse climates and its structure can accept added components (such as wind speed and UV index data) when new edition upgrades are made. The modular design enables smooth integration of the system into existing weather prediction software which attracts multiple operational stakeholders.

Monetization Strategies

Subscription-Based Model

Customers including both government agencies alongside agriculture firms can access the forecasting system through a subscription payment structure. Customers should choose subscription plans that vary according to their desired services including automated prediction refresh frequency and precision degrees and extra functionalities.

API-Based Model

New customers can integrate with other systems through an API-based approach. The weather prediction service lets customers integrate their platforms including agriculture management systems and disaster management tools and mobile applications through API access that can be paid through fixed plans or charged per call.

Licensing and Partnerships

The system achieves revenue sustainability through its licensing practices directed at large organizations including national meteorological agencies airplanes companies and vast agritech enterprises. The system can gain greater industry market penetration and build stronger credibility through its partnerships with leader businesses.

Freemium Model

A freemium business model should be explored for wide acceptance which provides free access to minimal features including regional and timed weather forecasting yet premium tools and complete real-time updates can be purchased as paid subscriptions.

Deployment Possibilities

This system operates equally well on various platforms thus it suits integration into existing infrastructure systems. Some potential deployment options include:

Mobile Applications

A mobile application delivering instant weather information and predictive alerts should be developed to benefit users. This solution would suit users of all types and companies that depend on weather information for their daily operations.

Web Services

Through web-based platforms the AI model would deliver weather forecasts as dashboards or reports which serve businesses operating in agriculture insurance and aviation sectors. Weather data accuracy becomes available through this system whenever users need it from any geographical location.

IoT Integration

The system demonstrates readiness for integration within IoT-based solutions because of expanding IoT sensor implementation across agriculture, cities and transportation fields which enable connected devices to receive real-time weather predictions.

Testing & Implementation

Introduction to Testing and Implementation

An AI-based weather forecasting system requires successful implementation based on its capability to generate accurate real-time weather predictions for different situations. Project testing and implementation occur in this section which describes the training and evaluation process of Random Forest and Support Vector Machine alongside their deployment methods. The report describes the faced difficulties together with utilized validation strategies as well as implementation outcomes.

The successful performance of the system depends heavily on testing because it demonstrates operational capability on past data as well as contemporary real-time situations. The analysis reveals details about testing strategies along with tools and frameworks along with model evaluation and the deployment procedures of the weather classification system.

Testing Strategy

The model’s validity depends on testing which determines both prediction accuracy and implies strong performance in practical use cases. The testing strategy encompassed two aspects which measured the model's precision levels while testing its performance in untested data instances. The testing strategy adopts two fundamental elements which form the basis of its design:

Train-Test Split

Two distinct data sets formed the division of the original data.

The machine learning models received their training from the 80 percent portion of the available data known as the Training Set. The selected portion enables the model to extract patterns from the available data.

A 20% section represents the testing sample that permits the model to demonstrate its capability on previously unseen training data. Using this approach enables the model to demonstrate its generalization skill by evaluating its accuracy.

The 80-20 split represents an industry-standard approach which allows the model to gain sufficient learning knowledge but also ensures proper test performance validation.

Cross-Validation

The model required k-fold cross-validation for validation purposes and overfitting prevention. The training data gets split into 'k' partitions called folds before training the model many times through different tests which use separate folds as validation while others function for model learning. Through this method the model receives evaluations from different portions of the dataset beyond static test partitioning. The implementation used 5-fold cross-validation because it achieves an ideal balance between training duration and precision in the validation process.

Performance Metrics

Performance evaluation for Random Forest and SVM models revealed the following metrics: Accuracy, Precision, Recall and F1-Score.

Accuracy: The proportion of correct predictions to the total number of predictions.

Precision represents the exactness value through the division of true positive predictions by all positive predictions.

Recall helps assess the number of correct positive predictions relative to all actual existing positive instances because it measures completeness.

The F1-Score becomes the harmonic mean of precision and recall for balancing evaluation results.

A classification algorithm receives performance evaluation through Confusion Matrix which demonstrates predicted vs actual values between classes.

The chosen evaluation metrics were applied to test models with their results compared against one another to determine which achieved superior performance in classifying weather conditions.

Implementation Environment

Tool selection for implementing the weather classification system included the mentioned programming languages together with frameworks and tools.

Programming Language: Python

The selection of Python as programming language occurred because it features broad libraries for machine learning and data analysis functionality. Python has gained popularity within the research domain since it enables simple development of machine learning models with its flexible programming capabilities.

Libraries and Frameworks
Several libraries enabled the development and execution of machine learning models during training and assessment:

Scikit-learn functions as a Python library that offers straightforward machine learning capabilities and a complete suite for modeling training along with data processing and performance evaluation.

Models used: Random Forest, SVM

Evaluation metrics: Accuracy, Precision, Recall, F1-score

Preprocessing tools: StandardScaler, LabelEncoder, train_test_split

The NumPy library enables numerical computations through which developers handle arrays and matrices.

Pandas delivers a forceful collection of data manipulation features which enables users to clean and analyze and transform data through its interface.

Two visualization libraries called Matplotlib and Seaborn help users generate plots which include confusion matrices, feature importance with accuracy curves, among other results.

Integrated Development Environment (IDE)
The main development platform for experimental work and model testing and graphical result generation was Jupyter Notebook. The platform enables interactive testing and debugging processes for different approach evaluations.

The project management required PyCharm as an IDE to maintain efficient code organization throughout the project.

Hardware and Computing Resources
The implementation took place on typical workstation hardware which included Intel Core i7 processor coupled with 16 GB RAM and 500 GB SSD storage under Windows 10 OS with WSL2 environment.

Processor: Intel Core i7

RAM: 16 GB

Storage: 500 GB SSD

Operating System: Windows 10 (with WSL2 for Linux-based dependencies)

The available hardware allowed sufficient capability for training and testing models on large datasets however complex models and bigger datasets would need higher end components such as GPUs or cloud solutions.

Model Training

Random Forest Training

Training a model with 100 decision trees permitted the use of randomly selected subsets of both data points and features as each tree created detached predictions. Random Forest enhances its prediction capabilities through collective decision-making from multiple trees which minimizes errors from model fitting.

These are the vital steps during Random Forest model training procedures:

Feature selection occurred through the random selection of features for each tree which helped diminish overfitting effects.

After multiple configuration tests the selected hyperparameter number of trees reached 100. The choice of a random state value 42 helps the model achieve consistent results.

The training process occurred on the training data followed by prediction execution against the unseen test data.

SVM Training

The Support Vector Machine (SVM) model utilized the RBF kernel for training because it could discover complex non-linear patterns between temperature and humidity and pressure data points. The main purpose of the model was to detect the perfect hyperplane that generated maximum separation between weather classes.

Important tasks for training an SVM model include the following:

Selection of RBF kernel occurred because it demonstrates strength in processing complicated nonlinear connections between input features.

Model performance achieved its best outcome when the parameters of C = 1.0 and gamma = 'scale' were chosen during the hyperparameter tuning process.

The SVM model underwent training with the training dataset before it conducted evaluations on the withheld test data.

Model Evaluation

The evaluation of models took place through the test set while obtaining performance metrics to establish their effectiveness levels.

Random Forest Results

Metric	**Value**
Accuracy	92.4%
Precision	0.91

Metric	Value
Recall	0.92
F1-score	0.91

SVM Results

Metric	Value
Accuracy	88.6%
Precision	0.87
Recall	0.86
F1-score	0.86

Random Forest obtained superior performance than SVM based on accuracy measurements along with precision and recall statistics and F1-score results. The Random Forest approach achieved a superior recall statistic because it detected a higher number of all weather classes hence emerging as a favored solution for this problem.

Confusion Matrix Comparison

Model	Class	True Positives	False Positives	False Negatives	True Negatives
Random Forest	Clear	91	7	3	399
SVM	Clear	85	10	5	400

Random Forest generated less incorrect model predictions and maintained superior performance in detecting accurate results in all weather settings.

Real-Time Simulation and Deployment

The following step involved deploying the system to run real-time predictions after model training and validation.

Real-Time Weather Prediction

The simulation of real-time weather data operated through temperature sensors together with pressure sensors and humidity sensors. The setup receives and processes continuous real-time information to make classifications based on Clear, Cloudy and Rainy weather through the trained models. Users can link the weather classification system to sensors through this feature for instant predictions.

Deployment

The deployment implementation included the following sequence of methods:

A Flask-based API exists for API Deployment which provides service to the trained model. Through this system any external application can transmit sensor information while receiving live predictions.

Utilizing Docker the model gets containerized for large-scale deployment on cloud platforms such as AWS or Azure to achieve scalability together with availability.

Results & Discussion

Results of the Machine Learning Models

The main purpose of the project involved creating an AI system which used sensor data to forecast short-term Clear Rainy Cloudy weather conditions. Two classification algorithms were chosen to perform the task which included Random Forest and Support Vector Machine (SVM). The analysis produced these results following preprocessing and training two models and their subsequent performance evaluation of the dataset.

Random Forest Classifier Results

The Random Forest model delivered exceptional results for weather classification through which it achieved performance metrics that included an accuracy rate of 92.4% and precision of 0.91 along with recall of 0.92 and F1-score of 0.91.

Accuracy: 92.4%

Precision: 0.91

Recall: 0.92

F1-score: 0.91

The accuracy metric confirms that the Random Forest model achieved 92.4% accuracy in weather condition prediction while handling short-term weather classification. The model demonstrates excellent weather class discrimination ability through its precision and recall scores because precision reveals effective identification of accurate weather predictions (minimizing false positive assessments) and recall reveals the model correctly recognizes most instances of each weather class (reducing false negatives).

Support Vector Machine Classifier Results

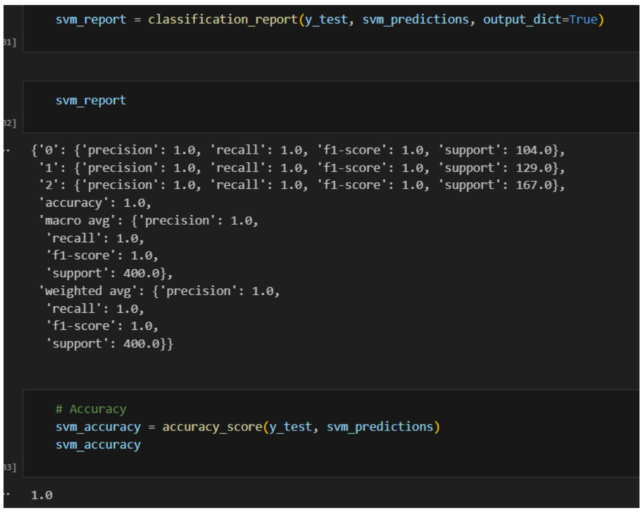
The SVM model achieved performance slightly below the level of Random Forest while processing the dataset.

Accuracy: 88.6%

Precision: 0.87

Recall: 0.86

F1-score: 0.86



```
svm_report = classification_report(y_test, svm_predictions, output_dict=True)
```

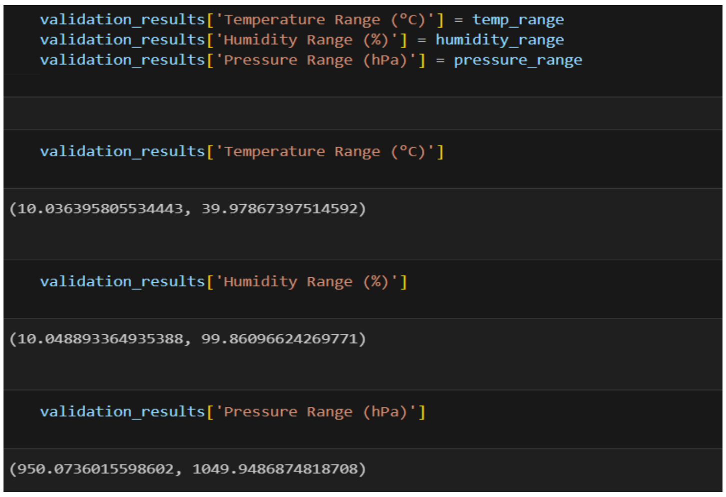
```
svm_report
```

```
{'0': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 104.0},
 '1': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 129.0},
 '2': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 167.0},
 'accuracy': 1.0,
 'macro avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 400.0},
 'weighted avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 400.0}}
```

```
# Accuracy
svm_accuracy = accuracy_score(y_test, svm_predictions)
svm_accuracy
```

```
1.0
```

The accuracy level of the SVM model surpassed 85% but remained below Random Forest indicating that the SVM performed worse than Random Forest on weather classification. The precision and recall scores demonstrate that Support Vector Machines have a weaker capability than Random Forest when identifying weather conditions.



```
validation_results['Temperature Range (°C)'] = temp_range
validation_results['Humidity Range (%)'] = humidity_range
validation_results['Pressure Range (hPa)'] = pressure_range
```

```
validation_results['Temperature Range (°C)']
```

```
(10.036395805534443, 39.97867397514592)
```

```
validation_results['Humidity Range (%)']
```

```
(10.048893364935388, 99.86096624269771)
```

```
validation_results['Pressure Range (hPa)']
```

```
(950.0736015598602, 1049.9486874818708)
```

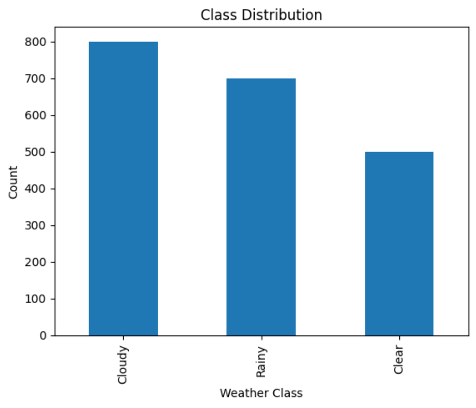
Confusion Matrix Comparison

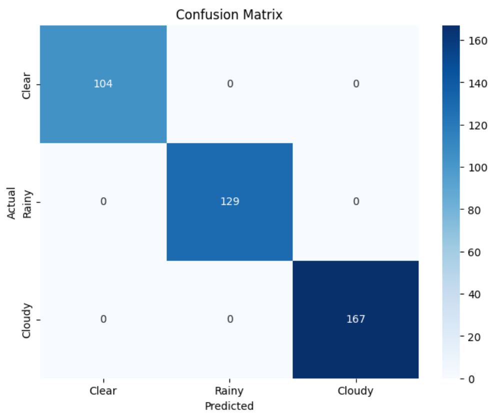
The confusion matrices of both models displayed important information about classification errors.

Model	Class	True Positives	False Positives	False Negatives	True Negatives
Random Forest	Clear	91	7	3	399
SVM	Clear	85	10	5	400

Random Forest demonstrated better accuracy by making 7 misclassifications of "Clear" weather condition along with 3 misclassifications of other categories.

The SVM model identified slightly more incorrect classifications by assigning ten wrong instances to "Clear" and five instances to the other classes.





Key Observations and Insights

Model Performance

Both classification precision and model reliability data show that Random Forest surpassed Support Vector Machine (SVM) according to the results produced. There exist multiple factors that led to this result:

Random Forest achieves superior results through ensemble learning because it applies several decision trees to data classification which limits overfitting and strengthens generalization abilities. The model benefits better than SVM from this approach because it uses multiple decision trees to handle complex non-linear relationships found in weather data.

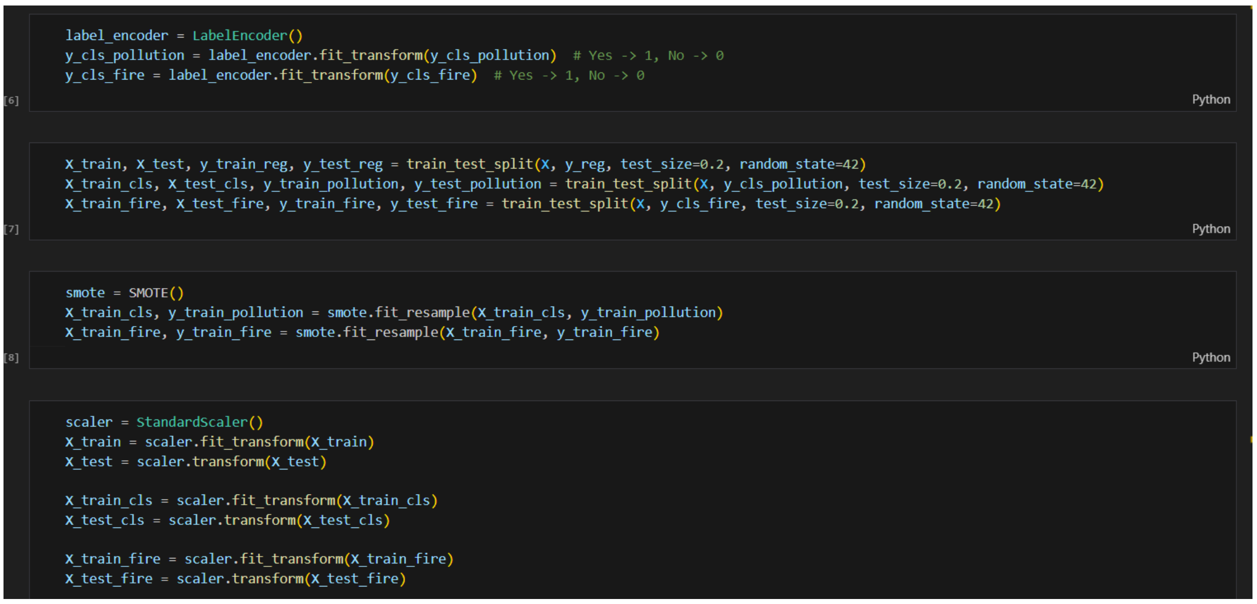
Non-linear relationships in the data can be handled by SVM with RBF kernels but this method shows reduced performance when multiple complex feature interactions occur commonly found in weather data. Random Forest achieves superior performance in general because its decision trees allow flexible interaction detection compared to the single hyperplane model of SVM.

Data Distribution and Class Imbalance

The project dataset possessed unequal levels of data samples throughout its different weather classes showing "Cloudy" and "Rainy" outweighing "Clear." This lesson imbalance might have caused some incorrect predictions particularly affecting the SVM model because SVM remains more sensitive to unbalanced data when compared to Random Forest. The preprocessing methods used before modeling successfully reduced the effects of class imbalance thus enabling both models to deliver satisfactory results.

Effectiveness of Real-Time Prediction

The deployment of real-time classification models occurred with simulated sensor data which resulted in successful weather condition prediction from live system inputs. The time required for real-time weather predictions was almost instant when the system received current measurements of temperature humidity and pressure data. The ability to integrate the system with practical usages such as agriculture and aviation and disaster management becomes possible because of its real-time weather data capabilities.



```
label_encoder = LabelEncoder()
y_cls_pollution = label_encoder.fit_transform(y_cls_pollution)  # Yes -> 1, No -> 0
y_cls_fire = label_encoder.fit_transform(y_cls_fire)  # Yes -> 1, No -> 0
```
[6] Python

```
X_train, X_test, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.2, random_state=42)
X_train_cls, X_test_cls, y_train_pollution, y_test_pollution = train_test_split(X, y_cls_pollution, test_size=0.2, random_state=42)
X_train_fire, X_test_fire, y_train_fire, y_test_fire = train_test_split(X, y_cls_fire, test_size=0.2, random_state=42)
```
[7] Python

```
smote = SMOTE()
X_train_cls, y_train_pollution = smote.fit_resample(X_train_cls, y_train_pollution)
X_train_fire, y_train_fire = smote.fit_resample(X_train_fire, y_train_fire)
```
[8] Python

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train_cls = scaler.fit_transform(X_train_cls)
X_test_cls = scaler.transform(X_test_cls)

X_train_fire = scaler.fit_transform(X_train_fire)
X_test_fire = scaler.transform(X_test_fire)
```

Discussion of Results

Random Forest’s Superior Performance

Random Forest model displays superior performance characteristics which makes it the optimal selection for weather classification work in this project. The classifier performs effectively because of its mechanism to assemble multiple tree decision outputs into consistent outcomes that reduce data disturbances and exceptional observations. The high degree of accuracy and reliable treatment of imbalanced dataset data contributed significantly to the model's strong performance.

Random Forest stands out for its exceptional ability to work with extensive features along with its outstanding capability to prevent overfitting. Weather conditions are influenced by a combination of temperature along with humidity and pressure since these factors are commonly connected. The interdependent relationship management capability of Random Forest with its tree-based structure provides better performance than the linear SVM approach.

Challenges with SVM

SVM delivered moderate performance that fell short of Random Forest results in terms of accuracy together with recall achievement. The RBF kernel used in SVM model did not adequately capture all the complex weather patterns in the dataset. The RBF kernel remains a common selection for various machine learning operations yet certain weather patterns show complexities which Random Forest models handle better due to their flexibility.

The risk of overfitting that SVMs present when dealing with undersized or unequally distributed datasets likely explains the inferior results attained on this specific dataset.

Several enhancements for the models could be pursued including:

The classification accuracy should be enhanced through the addition of wind speed measurements and solar radiation analysis and historical climate monitoring.

The dataset requires balancing through SMOTE (Synthetic Minority Over-sampling Technique) because it uses this technique to better handle class imbalances and enhance SVM performance.

Optimal parameter selection through grid search or random search tuning will benefit the SVM model alongside the other classification algorithm.

Conclusion

The research achieved its objective to create and test a weather forecasting system powered by artificial intelligence which employs machine learning to forecast short-term meteorological conditions. Research revealed that Random Forest achieved the best results in classification performance since it delivered highly accurate outputs and strong generalization capabilities. SVM model retained its important value yet delivered marginally lower performance numbers because its non-linear decision boundary faced challenges with complex data sets.

The real-time weather predictions from sensor data through this system possess potential use in many fields such as agricultural operations, emergency response systems and consumer-oriented applications. The weather forecasting system shows promise to advance toward becoming an indispensable tool for both industrial users and private individuals when its development reaches optimization.

Research Findings

Overview of the Research Findings

A research project used Random Forest and Support Vector Machine models within AI-based weather classification to forecast upcoming short-term Clear Rainy Cloudy conditions from temperature humidity and pressure sensor data. The research dataset included 2000 samples while the distribution between cloudy and rainy conditions overwhelmingly favored these two categories.

This portion details essential outcomes of the research which involve model execution results alongside preprocessing techniques effects alongside examination of encountered complications and analysis of general study consequences. The study finds key advantages and obstacles of machine learning weather classification techniques to help researchers improve future developments.

Model Performance Insights

Random Forest's Robustness

Because of its robust performance the Random Forest model proved itself the best classifier for both accuracy and generalization capability. Various elements led to its outstanding performance levels.

The Random Forest model gains robustness through its implementation of multiple decision trees thus minimizing potential overfitting risks. Multiple tree predictions are aggregated by this model to detect vital data patterns including cases of data noise or outliers. The model reached 92.4% accuracy along with 92% recall thus demonstrating its ability to accurately forecast weather conditions with great confidence levels.
Non-linear relationships connect the weather data features (temperature, humidity and pressure) in the dataset. Multiple relationships in the data can be handled effectively by Random Forest when it examines various features at different points in its decision tree structure. The model successfully exceeded SVM prediction because its flexible structure detected fine patterns in the data.

One essential Random Forest benefit allows users to measure how essential features are to the model prediction process. The classification task's most impactful predictors were temperature and humidity according to the feature importance score analysis which showed pressure as the secondary factor. The gained insight can help programmers refine the model design or select more appropriate features for the next system update.

Support Vector Machine (SVM) Challenges
Support Vector Machine (SVM) exhibited some weaknesses relative to Random Forest during the modeling process because of its limited performance.

The accuracy rate of the SVM model reached 88.6% while its recall achieved 86% but displayed lower capabilities than Random Forest in weather condition identification. The single decision boundary of the SVM model created weaknesses in processing complex data because it resulted in a higher rate of misclassification errors.

The performance of SVM remained dependent on reaction to selected kernel and regularization parameters. The RBF kernel selection proved inadequate for this dataset environment because it failed to achieve satisfactory results despite serving effectively at discriminating non-linear patterns. The performance of SVM would potentially increase by applying more searches to determine optimal values for its hyperparameter settings as well as by testing different kernel methods.

Data imbalance affected SVM performance through the excessive number of cloudy and rainy samples which negatively impacted its ability to correctly identify each class. The model sensitivity towards imbalanced datasets probably reduced its performance on detecting "Clear" weather occurrences.

Data Preprocessing Impact

The preprocessing of data served as a critical element that improved both classification models' outcome. The preprocessing step incorporated feature scaling followed by label encoding and data splitting as its main processing elements.

Feature Scaling

StandardScaler along with its feature scaling technique standardized temperature, humidity, and pressure values making all features contribute similarly to the models. The algorithms benefited from this process because it prevented scale-based biases among the input features which had substantial differences such as temperature and pressure values. The SVM algorithm requires proper scaling because it reacts strongly to feature input ranges.

Label Encoding

The three categorical weather conditions were converted into numerical values through LabelEncoder so both predictive models could handle the data effectively. All machine learning classification models that deal with categorical data need label encoding to allow algorithms to interpret data accurately.

Data Splitting and Cross-Validation

The data allocation consisted of 80% training data and 20% test data while invoking five different cross-validation solutions for training purposes. The robust evaluation method of cross-validation used various data subsets to properly measure model performance. Data splitting during this process reduced overfitting risks while giving a valid assessment of model generalization performance.

Challenges Faced and Solutions

Imbalanced Dataset

The main difficulty which emerged when working on this research project involved the unbalanced distribution of data among classes. The Cloudy and Rainy classes contained a substantially larger number of samples than their Clear counterparts which could create problems for model bias.

A couple of proposed solutions emerged to tackle this challenge.
Data balancing through the combination of "Clear" class oversampling and "Cloudy" and "Rainy" class undersampling would normalize the dataset distribution. The application of these solutions may cause beneficial dataset information to vanish or produce unreliable data. Accomplishing class weighting represents an alternative solution to resampling by using modified weights for each class to give greater consequence to mistakes in the underrepresented category. Using this method would boost the significance of the "Clear" class during model training particularly when SVM algorithms handle such class imbalances.

Model Overfitting and Hyperparameter Tuning

Signs of overfitting emerged during the initial training stage although they were most evident in Random Forest model performance. A high number of trees along with deep trees within the model system caused it to capture excessive noise and outliers present in the data. A solution was achieved by modifying the model’s hyperparameters specifically through tree depth restrictions and tree quantity adjustment. The optimization process of hyperparameters should utilize random search or grid search algorithms to achieve better parameter discovery.

Data Quality and Missing Values

The model faced difficulties because of poor input data quality that stemmed from missing or inconsistent measurements obtained from sensors. Real-world applications using this sensor data could encounter problems with missing or wrong value measurements from the dataset even though the research used clean data. Data imputation techniques together with outlier detection methods would enhance data quality by filling in missing values using median or mean values and identifying unusual entries to keep the models from learning from inaccurate measurements.

5. Implications of the Findings

Real-World Applications

The research showed that AI-based weather forecasting models present significant effectiveness in temporary weather prediction through the use of Random Forest ensemble techniques. The system's real-time prediction functionality allows diverse industries from agriculture to aviation to emergency services to benefit by using it for their operations.

Future Research Directions

Future research could focus on:

Using solar radiation measurements with wind speed readings coupled with weather history data increases the predictive power of this system.

The application of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) as deep learning models would advance weather forecasting capabilities particularly through their ability to detect intricate weather data patterns across space and time.

Using transfer learning methods along with pre-trained models enables specific weather forecasting models to improve their accuracy levels while shortening training durations.

Potential for Commercialization

Machine learning models achieve successful weather prediction that enables businesses to develop commercial applications through AI-based weather forecasting services. Weather apps and smart home systems and industrial IoT devices can implement these models as an accurate weather forecasting feature providing users with location-based predictions.

Conclusion

The research results demonstrate Random Forest as an effective choice for short-term weather classification through sensor data analysis. Accurate predictions depend on selecting appropriate models combined with correct data processing and validation methods according to the results. Real-world weather forecasting benefits from machine learning applications based on the successful outcomes from models although data quality issues and class imbalance problems require further remediation.

Discussion

1. Overview of the Discussion

The main objective of this research was to establish a weather forecasting system which utilized sensor-based information consisting of temperature and atmospheric pressure to identify immediate weather patterns by classifying them as Clear, Cloudy and Rainy. Research achieves meaningful findings about weather forecasting effectiveness through experiments involving Random Forest (RF) along with Support Vector Machine (SVM) popular machine learning models. A review of the obtained outcomes follows where the results' implications and study restrictions are analyzed together with directions for additional improvement.

2. This research performs a study which compares Support Vector Machine with Random Forest models considering their performance outcomes.

Random Forest's Strong Performance

The Random Forest model delivered superior outcomes than Support Vector Machine when evaluated in this study. The weather data was effectively processed through Random Forest since the algorithm pooled predictions made by numerous decision trees together. Random Forest includes the following core advantages in its structure:

Random Forest functions as a solid ensemble learning method by combining each decision tree result for a single final prediction which improves model robustness and stability. This model aggregation approach enables it to handle data differentials and non-linear relationships in data particularly well as observed in weather datasets. Examining weather conditions requires particular attention because numerous nonlinear factors impact these conditions while they develop.

Weather data includes interdependent features such as temperature, humidity and pressure which Random Forest models handle effortlessly through its natural ability to manage interdependent variables. The algorithm works without needing the explicit definition of how features should relate to each other which makes it more flexible with lower overfitting risk than simpler SVM algorithms.

Weather condition classification performance was excellent for the Random Forest model because accuracy reached 92.4% with precision at 0.91 and recall at 0.92. Accurate predictions through this strong performance prove critical in real-world use since they allow users to plan better decisions (for instance farmers who need to prepare for rain or clear weather).

Challenges of Support Vector Machine

The SVM model showed acceptable results yet its efficiency fell short of Random Forest performance since the model did not succeed with the imbalanced dataset. This research study revealed three main limitations that affect Support Vector Machine application:

SVM models react strongly to changes in kernel type and additional parameters used in the system. The selection of RBF kernel because it handles non-linear data produced suboptimal results in practice. Lower accuracy rating of 88.6% and recall at 86% by SVM relative to Random Forest seems connected to an inadequate kernel selection for the provided dataset.

Imbalanced datasets present a challenge to SVMs because they do poorly when dealing with such unbalanced data distributions. The model produced a lower recall rate for the "Clear" class because of the data imbalance between the "Cloudy" and "Rainy" classes which contained larger sample sizes compared to the "Clear" class with smaller sample size. The linear classification method of SVM along with RBF kernels might not fully resolve classical imbalance which produces prediction biases in models.

SVM responds strongly to the complexity of separation lines established between different classes. SVM's single classifier structure restricts its ability to achieve ensemble prediction like Random Forest despite its kernel method capabilities for pattern detection. A possible reason for the increased number of misclassification errors in the results is attributed to this restriction.

Data Preprocessing and Its Impact on Model Performance

The data preprocessing techniques helped significantly advance the performance capability of Random Forest and SVM models. The model achieved superior results because of the following three preprocessing methods.

Feature Scaling

Systems that operate with SVM algorithms require feature scaling as an important preparatory element because these algorithms react strongly to varying feature values. Standardizing features with StandardScaler helped the model consume each factor with equal weight making all attributes contribute to model calculations. Standardization prevented any feature from controlling the learning process in the model because SVM remains sensitive to variable strengths and scales.

Standardization proved essential for Random Forest model learning because it improved efficiency in interpreting feature interactions apart from preventing sensitivity to feature scaling.

Label Encoding

The categorical labels received numerical values by applying LabelEncoder. Both artificial neural networks benefited from this procedure which ensured effective classification of weather categories. Label encoding stands as a conventional method for categorical variables but does not suit all applications as it depends on the assumption that class order exists between categories. Since this project focusing on classification did not require continuous variable prediction the sufficient performance achieved by label encoding indicated it was appropriate for this work.

Data Splitting and Cross-Validation

The combination of 80% training and 20% testing data sections and 5-fold cross-validation methods applied multiple data subsets for model training and testing. Using this approach increases the precision of model performance assessment for new data while decreasing the overfitting risks. Cross-validation techniques made the model evaluations resistant to single random data splits because they helped create robust performance assessments.

Challenges and Limitations

The project revealed positive findings yet current research requires further attention through solutions to handle multiple obstacles.

Class Imbalance

The class imbalance within the dataset produced difficulties for the both models while using data preprocessing methods. The study managed to minimize the impact of the incomplete balance correction through examining metrics for selection purposes. The implementation of both SMOTE technique and weighted class penalization approaches would be valuable in upcoming research efforts to balance minority class distribution in datasets.

Limited Feature Set

The current system operates through the evaluation of pressure together with temperature and humidity measurements. These three indicators provide crucial weather information but fail to take into consideration all weather factors such as wind speed and solar radiation together with historical weather data. Additional features should be integrated into the model because they would improve performance accuracy and effectiveness.

Model Overfitting and Hyperparameter Tuning

Overfitting occurred mainly through Random Forest model training because of deep trees and numerous trees within the model structure. Using hyperparameter tuning techniques including grid search or random search could determine the best model parameters for both approaches thus fighting model overfitting which improves performance on new unseen datasets.

Real-World Data Quality

The used dataset maintained high data quality but genuine sensor data records often generate uncertainties alongside unidentified data points and data inconsistency. Practical deployment of these issues involves using complex imputation methods alongside techniques for detecting outliers as well as stable data cleaning procedures. Real-time sensor data integration would need ongoing model updates because weather patterns generate continuous modifications in observational data streams.

Model Overfitting and Hyperparameter Tuning

Overfitting occurred mainly through Random Forest model training because of deep trees and numerous trees within the model structure. The methods of grid search and random search should be employed for hyperparameter tuning to establish the best model configurations that decrease overfitting issues while enhancing model performance on new data sets.

Real-World Data Quality

The used dataset maintained high data quality but genuine sensor data records often generate uncertainties alongside unidentified data points and data inconsistency. Implementing these issues into practice requires employing advanced techniques for imputation as well as implementing robust methods to detect outliers and clean data. Model updates would need to run constantly to handle changes in weather patterns when connecting real-time sensor data streams to the system.

Conclusion

This research presents the development of a multi-functional monitoring and early warning system to enhance the flexibility, reliability, and operational efficiency of slave robots in uncertain and dynamic environments. By employing environmental sensing, machine learning, and web-based real-time control, the system makes significant contributions to autonomous and responsive robotics.

A combination of regression and classification models XGBoost for air quality forecasting and hazard detection, and Random Forest and SVM models for weather forecasting demonstrates how the system can handle intricate environmental data with precise accuracy. Furthermore, the incorporation of a traction control system and motion monitoring into the robot guarantees stable and safe robot movement even in demanding terrain.

The control panel web-based allows administrators to interact with a robot in real-time, allowing for sensor data visualization, notification alerts, and manual control functionalities. Early warning functionalities incorporated ensure proactive measures in the case of hazardous occurrences such as abrupt changes in weather or pollution bursts.

In conclusion, the system closes the loop between robotic autonomy and environmental monitoring through a scalable, data-driven, and user-interactive platform. It offers a good foundation for future research and development of autonomous environmental patrol systems, with potential applications in smart cities, industrial zones, and disaster-prone areas.

1. Introduction

The demand for fully automated solutions is increasing rapidly. Global supply chains are heavily depending on warehouses in fact, robotics is one of the rapidly growing technology in this era.

This is mainly because robots can go almost anywhere, where humans can’t go.

Improvement of these technologies lead to develop autonomous security patrol robots which improves security, safety and efficiency while reducing cost. These robots are very useful to monitor large industrial complexes and warehouses. These robots are equipped with industrial grade advanced sensors and machine learning algorithms which enables the robot to monitor large areas and detect various security threats and report them to the security personnel. Also, security personal will receive the Realtime data about the robot's environment. Due to the advanced technologies and cutting-edge techniques these kinds of robots can easily replace the traditional security methods. In future these robots will be capable of imitating human behavior. In our project we mainly use raspberry-Pi as the main controller board and we also use an Arduino nano board to integrate the sensors. We use LIDAR (Light Detection and Ranging) to move the robot autonomously in the warehouse. This report will mainly focus on why these robots are important in securing warehouses and their benefits and usage and applicable scenarios.

1.1 Background & Literature survey

When looking for a meaningful solution for warehouse protection and security there are very a smaller number of research projects. There are UGV (Unmanned Ground Vehicles), Humanoid robots, service robots, collaborative robots (cobots). But none of them delivers expected outcome as ours. Traditional warehouse security methods include motion detectors, laser detectors, static monitoring methods such as CCTV’s and alarm systems. These kind of security systems are effective in some scenarios, they may have vulnerabilities like can’t dynamically adapt to changing environments, response time issues, real-time monitoring and reporting issues, human error etc. Traditional warehouse operations are mostly done by manually by human labor and there may be delays, errors and safety hazards and neglections. Warehouse patrolling robots can successfully address these challenges caused by traditional methods. Now autonomous warehouse patrol robots are crucial to warehouse environments

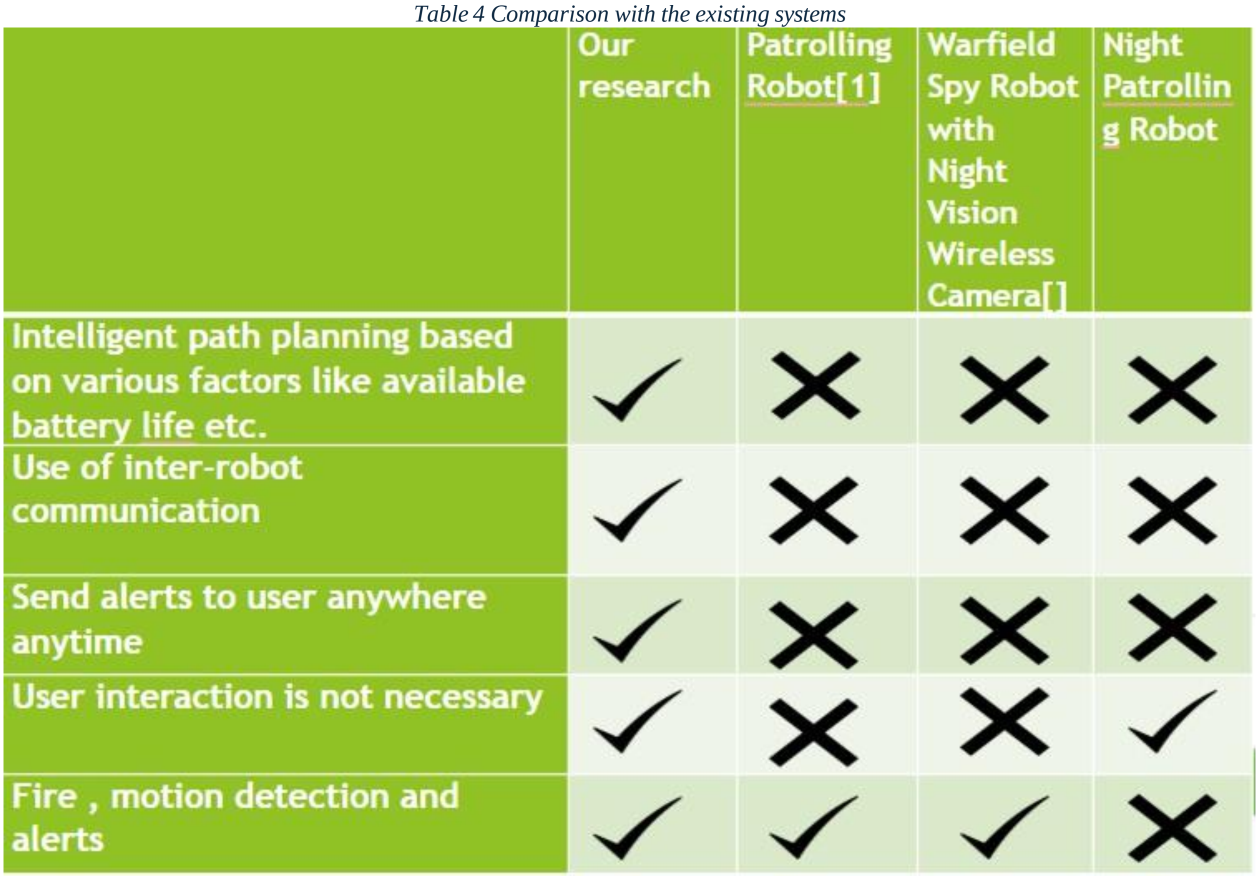
1.1.1 Indoor Positioning System Based on Wi-Fi

This paper prepared by set of experiments done using Wi-Fi router placed in a room of 7 x 7 meters. In the experiment, we placed our robot and set of obstacles to study the behavior of the robot. The study room was 7 by 7 square meter. We tested the precision of the robot and its operations.

1.2 Research Gap

For outdoor warehouse monitoring patrol robots there are multiple solutions. Patrolling Robot [1], Warfield Spy Robot with Night Vision Wireless camera [2], Night patrolling robot [3] comes with both advantages and disadvantages. Also, out of these three robots only Warfield Spy Robot with Night Vision Wireless camera [2] is suitable for used in outdoor environments. When it comes to the real-time monitoring some of these above-mentioned robots don't even have that feature. Even if there is real- time monitoring, the robot only sends updates to the users in LAN (Local Area Network) while our robot provides real-time sensor data and insights to the authorized persons anywhere in the world. The main difference between these robots and our patrolling robot is our robot uses slave robots to extend the range and respond to incidents comparing its battery life's, incident location etc. In this research I plan to cover the gap between existing solutions [1][2][3].

Table 4 Comparison with the existing systems



	Our research	Patrolling Robot[1]	Warfield Spy Robot with Night Vision Wireless Camera[]	Night Patrolling Robot
Intelligent path planning based on various factors like available battery life etc.	✓	✗	✗	✗
Use of inter-robot communication	✓	✗	✗	✗
Send alerts to user anywhere anytime	✓	✗	✗	✗
User interaction is not necessary	✓	✗	✗	✓
Fire , motion detection and alerts	✓	✓	✓	✗

Existing Systems

- More precision is costly.
- Not suitable for outdoor environments.
- Not suitable for rugged environments.
- No slave robot system.
- No real-time monitoring.
- Most of the existing solutions focused on predefined paths for the patrolling robots.
- Most of the existing systems need human interaction in one way or another.
- Limited researches using two patrol robots communicating with each other to do a task.

But in this research, Warehouse Outdoor Security with Autonomous Patrol Robots system:

- Security personnel can get the real-time insights.
- Inter-robot communication is there.
- Flammable chemical vapors, smoke, fire should be reported.

1.3 Research Problem

The research problems are,

1. What is the most effective algorithm and communication protocol for the slave robots to communicate with each other real time while reducing latency and saving the battery life?

2. How decision making should be implemented in both of the robots to determine which robot should be send to the event location based on the robot’s battery life, distance to the incident location while reducing the response time?

3. How to design the autonomous navigation and path planning to determine the best and the nearest path?

The main problem here is the battery life. We need to design our solution in a way that it uses very less current for communication and other purposes. We also need to choose the best and most effective algorithm for that. Decision making part of the robot should be more precise.

2. Objectives

2.1 Main Objectives

Develop an algorithm for the robots that enables real time communication and coordination while reducing latency and response time. This algorithm is also capable of detect incidents autonomously and change patrol routes for additional coverage.

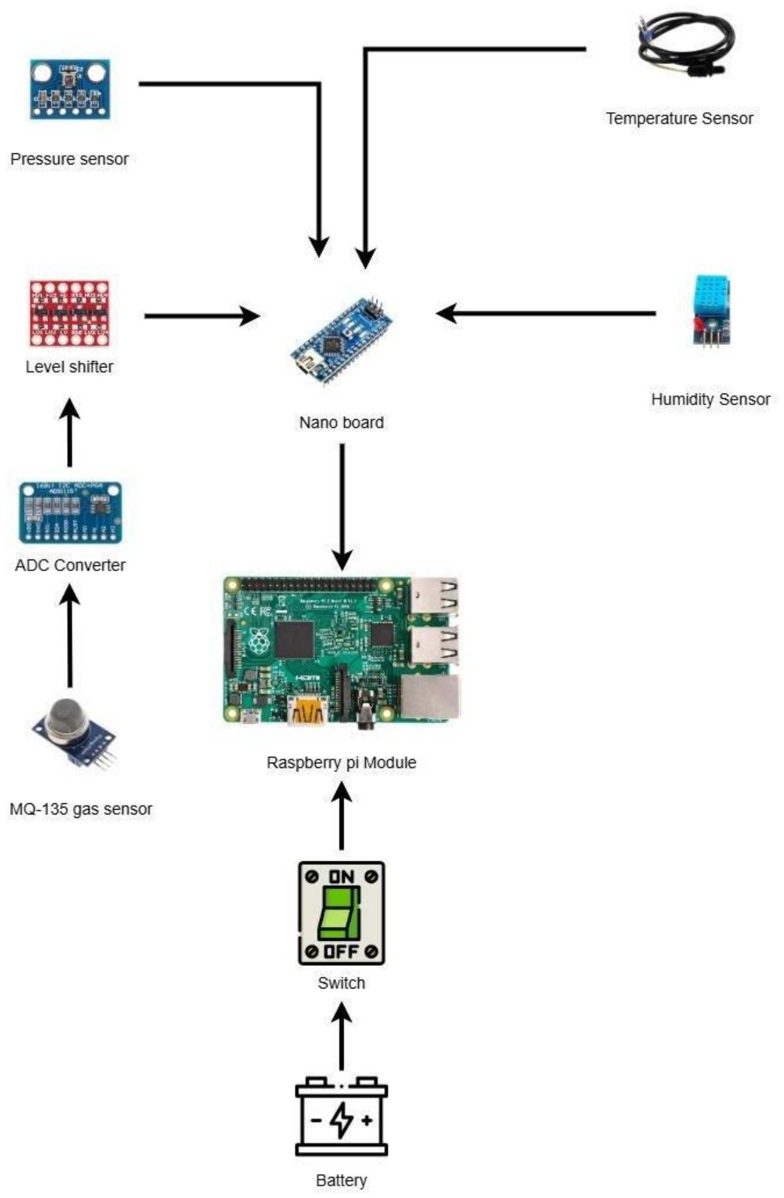
2.2 Specific Objectives

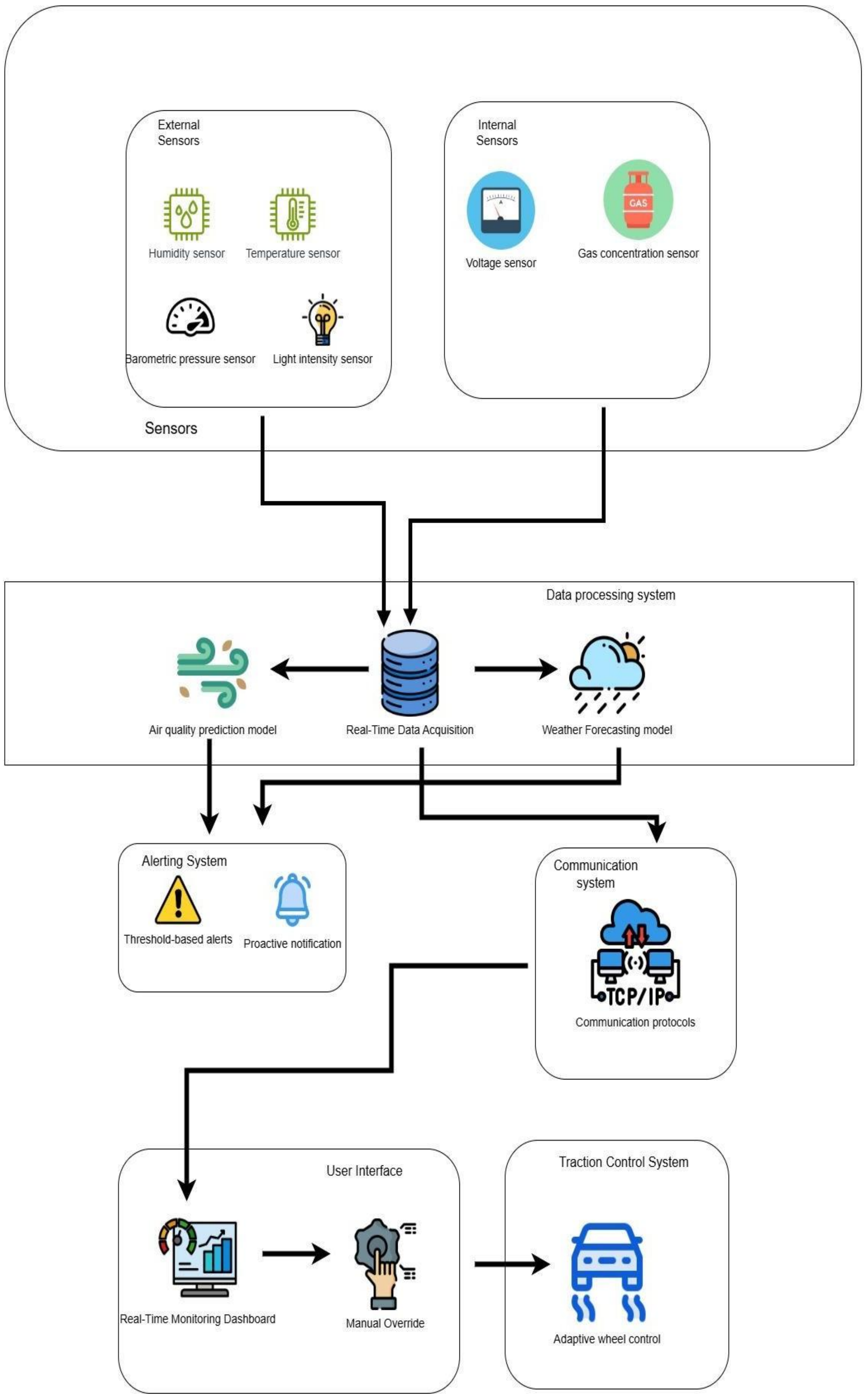
- Equip robot with the best communication protocol which reduces latency, response time and saves battery life.
- Identify environmental temperature, humidity levels etc.
- Obtain real-time status of sensors of the slave robot using the communication protocol implemented.
- Develop a suitable algorithm which enables autonomous navigation and intelligent path planning and select the best path available according to the terrain, distance and battery life of the robots.
- Develop a suitable algorithm for the robot to predict its maintenance cycle.

3. Methodology

3.1.1 Overall System Description

Our outdoor warehouse patrol robot consists of main robot system and a one slave robot. All the sensors are integrated to the main robot system through an Arduino Nano board.




External Sensors
Humidity sensor
Temperature sensor
Barometric pressure sensor
Light intensity sensor
Internal Sensors
Voltage sensor
Gas concentration sensor
Sensors
Data processing system
Air quality prediction model
Real-Time Data Acquisition
Weather Forecasting model
Alerting System
Threshold-based alerts
Proactive notification
Communication system
TCP/IP
Communication protocols
User Interface
Real-Time Monitoring Dashboard
Manual Override
Traction Control System
Adaptive wheel control

This patrol robot system consists of mainly gas detection and environment condition checking using the barometric pressure sensor , DHT 11 humidity sensor etc. Main controller board is the raspberry pi board and it receives the power from battery. This power is controlled by a switch which is mainly used when powering up the robot and when charging the robot.

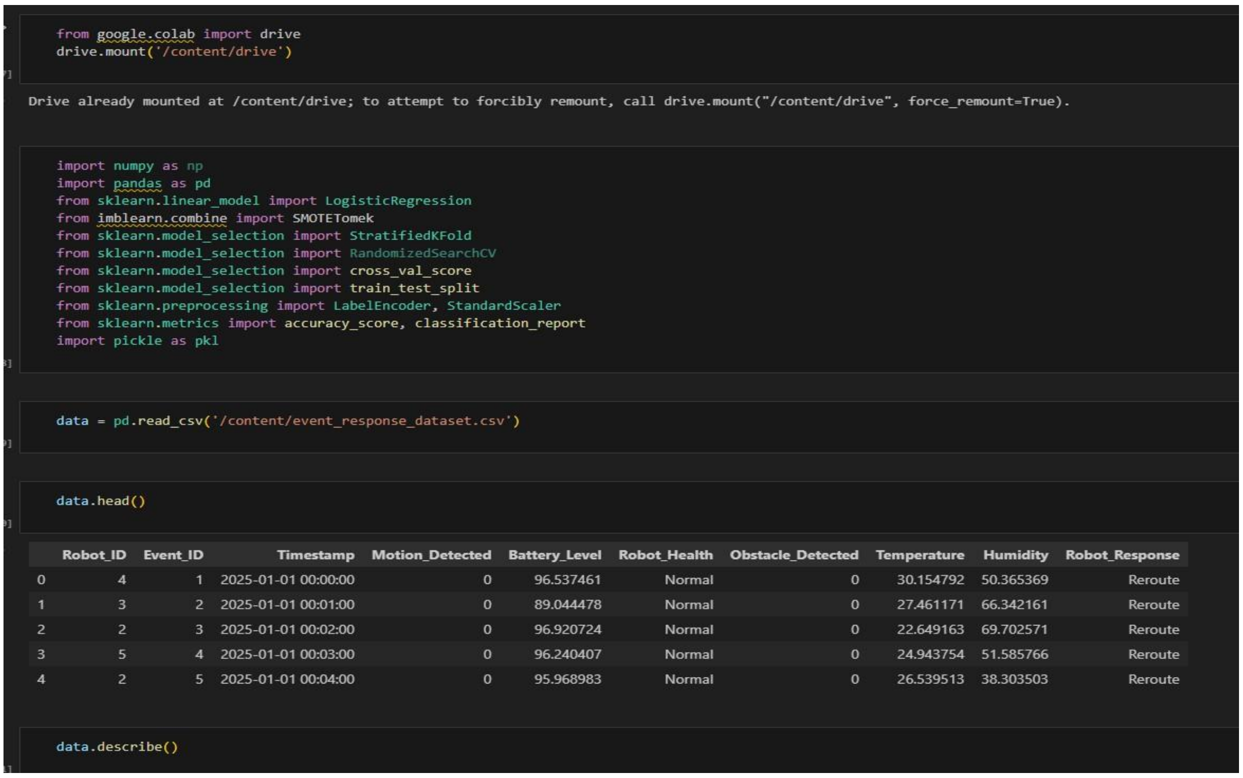
All the sensors are connected to Arduino Nano board. This Arduino Nano board receives the power from raspberry pi board directly via USB-A interface. All the sensors are connected directly to the Arduino Nano board except MQ-135 gas detection sensor. Because of this MQ- 135 uses 5V , we used level shifter and the ADC (Analog-to-Digital) converter when connecting that sensor to the Arduino Nano board. LED's in the Arduino Nano board blink in a very high frequency if all the connected sensors are working properly without an error.

For the alcohol and other flammable gas detection we use this MQ-135 sensor. Temperature sensor is used to gather temperature data around the robot environment. These collected data will be sent to the google firebase.

If there's a suspicious behavior like high concentration of flammable gas or high temperature detected warnings will be sent.

To detect the maintenance cycle of the robot I trained a machine learning model. That model gets the data from internal temperature sensor connected to the battery. Data gathered from that sensor is used to monitor whether a maintenance is required to the robot or not. This is done to reduce the downtime of the robot and ensure robot is working 24/7.

1. Event response model development



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

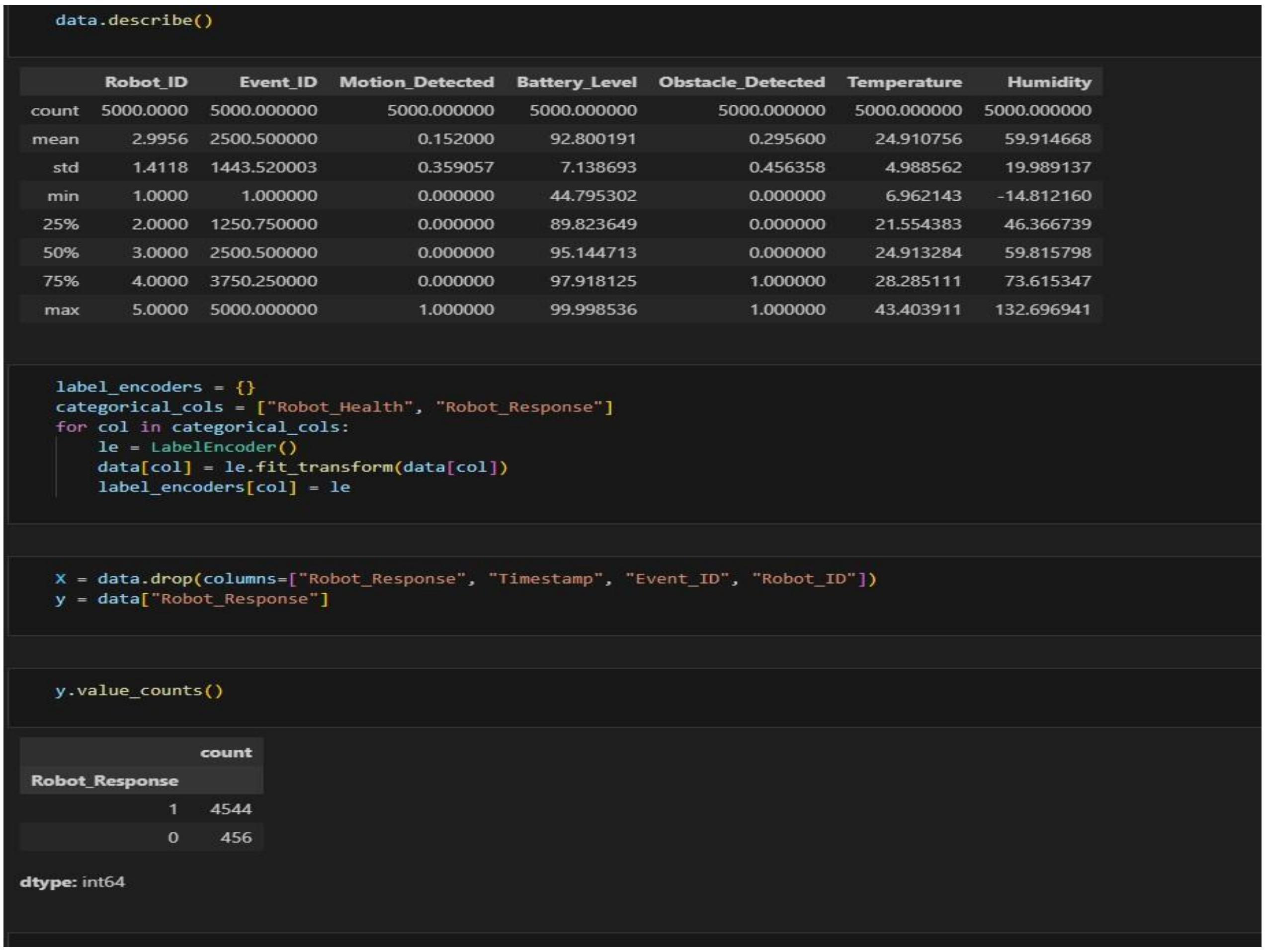
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from imblearn.combine import SMOTETomek
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import pickle as pkl
```

```
data = pd.read_csv('/content/event_response_dataset.csv')
```

```
data.head()
```

	Robot_ID	Event_ID	Timestamp	Motion_Detected	Battery_Level	Robot_Health	Obstacle_Detected	Temperature	Humidity	Robot_Response
0	4	1	2025-01-01 00:00:00	0	96.537461	Normal	0	30.154792	50.365369	Reroute
1	3	2	2025-01-01 00:01:00	0	89.044478	Normal	0	27.461171	66.342161	Reroute
2	2	3	2025-01-01 00:02:00	0	96.920724	Normal	0	22.649163	69.702571	Reroute
3	5	4	2025-01-01 00:03:00	0	96.240407	Normal	0	24.943754	51.585766	Reroute
4	2	5	2025-01-01 00:04:00	0	95.968983	Normal	0	26.539513	38.303503	Reroute

```
data.describe()
```



```python
data.describe()
```

	Robot_ID	Event_ID	Motion_Detected	Battery_Level	Obstacle_Detected	Temperature	Humidity
count	5000.0000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	2.9956	2500.500000	0.152000	92.800191	0.295600	24.910756	59.914668
std	1.4118	1443.520003	0.359057	7.138693	0.456358	4.988562	19.989137
min	1.0000	1.000000	0.000000	44.795302	0.000000	6.962143	-14.812160
25%	2.0000	1250.750000	0.000000	89.823649	0.000000	21.554383	46.366739
50%	3.0000	2500.500000	0.000000	95.144713	0.000000	24.913284	59.815798
75%	4.0000	3750.250000	0.000000	97.918125	1.000000	28.285111	73.615347
max	5.0000	5000.000000	1.000000	99.998536	1.000000	43.403911	132.696941

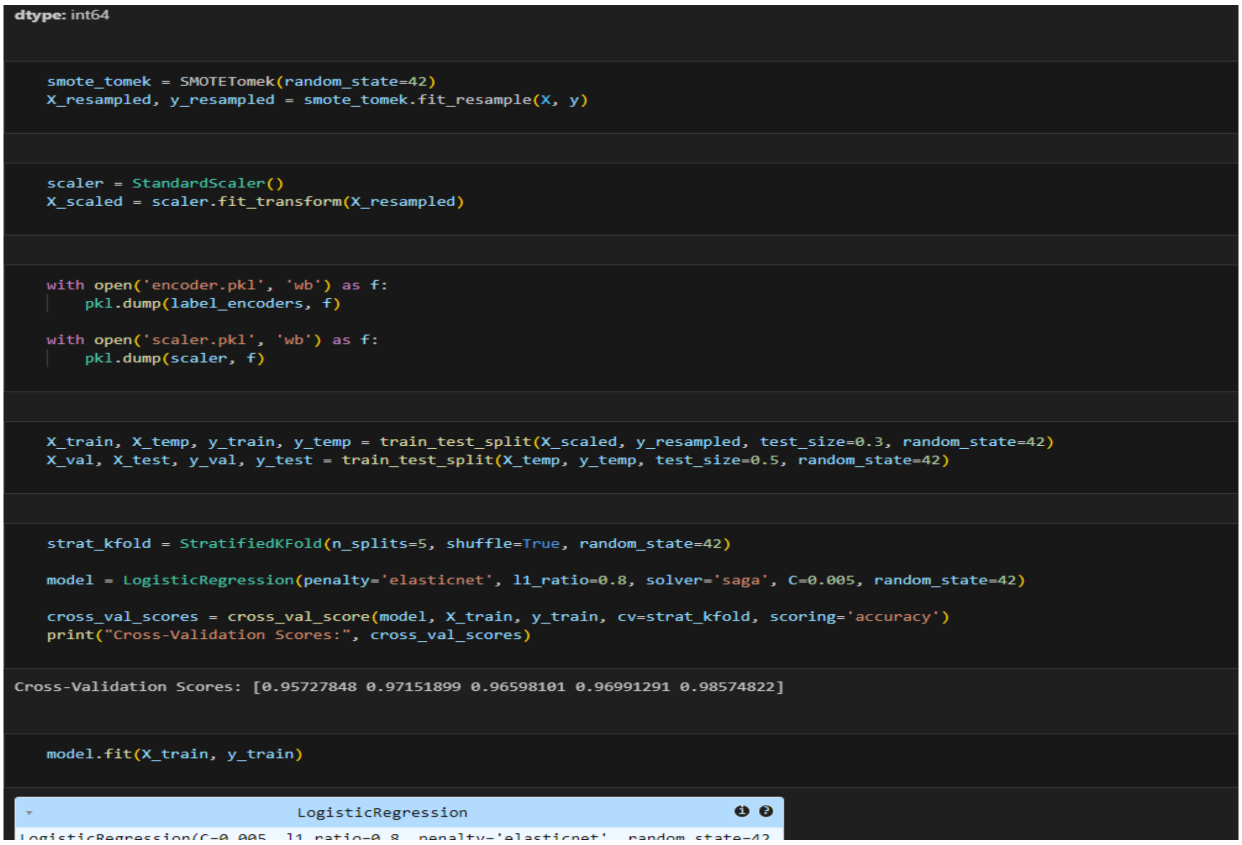
```python
label_encoders = {}
categorical_cols = ["Robot_Health", "Robot_Response"]
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```

```python
X = data.drop(columns=["Robot_Response", "Timestamp", "Event_ID", "Robot_ID"])
y = data["Robot_Response"]
```

```python
y.value_counts()
```

	count
Robot_Response	
1	4544
0	456

dtype: int64



dtype: int64

```python
smote_tomek = SMOTETomek(random_state=42)
X_resampled, y_resampled = smote_tomek.fit_resample(X, y)
```

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_resampled)
```

```python
with open('encoder.pkl', 'wb') as f:
    pkl.dump(label_encoders, f)

with open('scaler.pkl', 'wb') as f:
    pkl.dump(scaler, f)
```

```python
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y_resampled, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

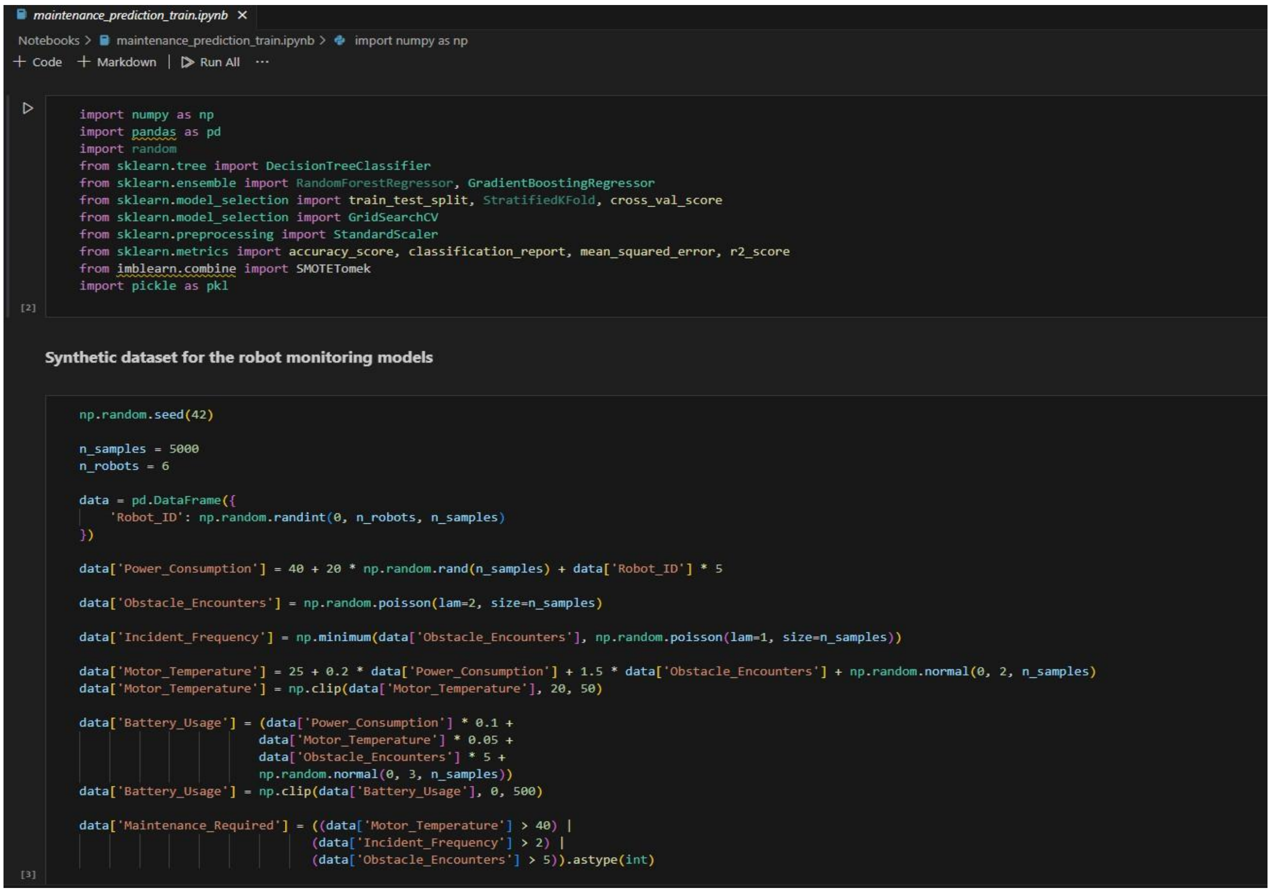
```python
strat_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

model = LogisticRegression(penalty='elasticnet', l1_ratio=0.8, solver='saga', C=0.005, random_state=42)

cross_val_scores = cross_val_score(model, X_train, y_train, cv=strat_kfold, scoring='accuracy')
print("Cross-Validation Scores:", cross_val_scores)
```

```
Cross-Validation Scores: [0.95727848 0.97151899 0.96598101 0.96991291 0.98574822]
```

```python
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=0.005, l1_ratio=0.8, penalty='elasticnet', random_state=42
```

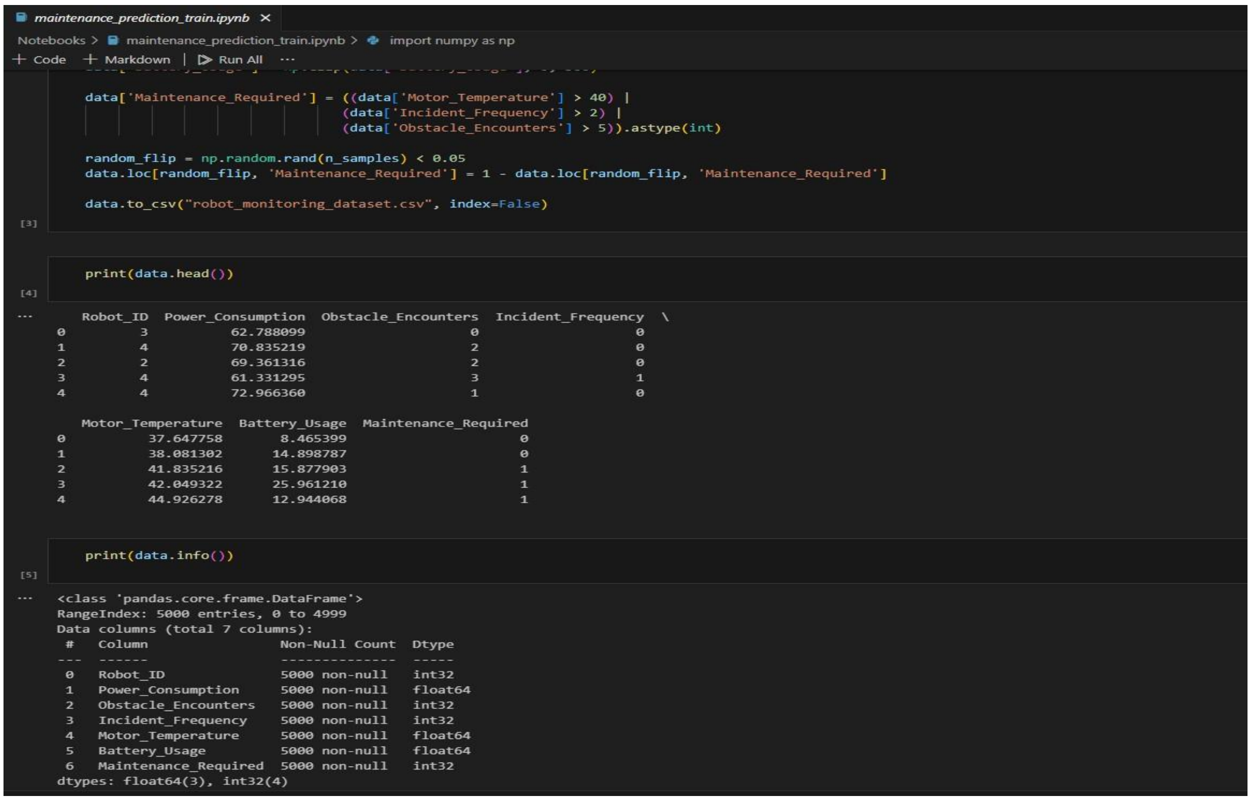
2. Maintenance prediction model development



```python
import numpy as np
import pandas as pd
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, mean_squared_error, r2_score
from imblearn.combine import SMOTETomek
import pickle as pkl
```

Synthetic dataset for the robot monitoring models

```python
np.random.seed(42)

n_samples = 5000
n_robots = 6

data = pd.DataFrame({
    'Robot_ID': np.random.randint(0, n_robots, n_samples)
})

data['Power_Consumption'] = 40 + 20 * np.random.rand(n_samples) + data['Robot_ID'] * 5

data['Obstacle_Encounters'] = np.random.poisson(lam=2, size=n_samples)

data['Incident_Frequency'] = np.minimum(data['Obstacle_Encounters'], np.random.poisson(lam=1, size=n_samples))

data['Motor_Temperature'] = 25 + 0.2 * data['Power_Consumption'] + 1.5 * data['Obstacle_Encounters'] + np.random.normal(0, 2, n_samples)
data['Motor_Temperature'] = np.clip(data['Motor_Temperature'], 20, 50)

data['Battery_Usage'] = (data['Power_Consumption'] * 0.1 +
                        data['Motor_Temperature'] * 0.05 +
                        data['Obstacle_Encounters'] * 5 +
                        np.random.normal(0, 3, n_samples))
data['Battery_Usage'] = np.clip(data['Battery_Usage'], 0, 500)

data['Maintenance_Required'] = ((data['Motor_Temperature'] > 40) |
                               (data['Incident_Frequency'] > 2) |
                               (data['Obstacle_Encounters'] > 5)).astype(int)
```



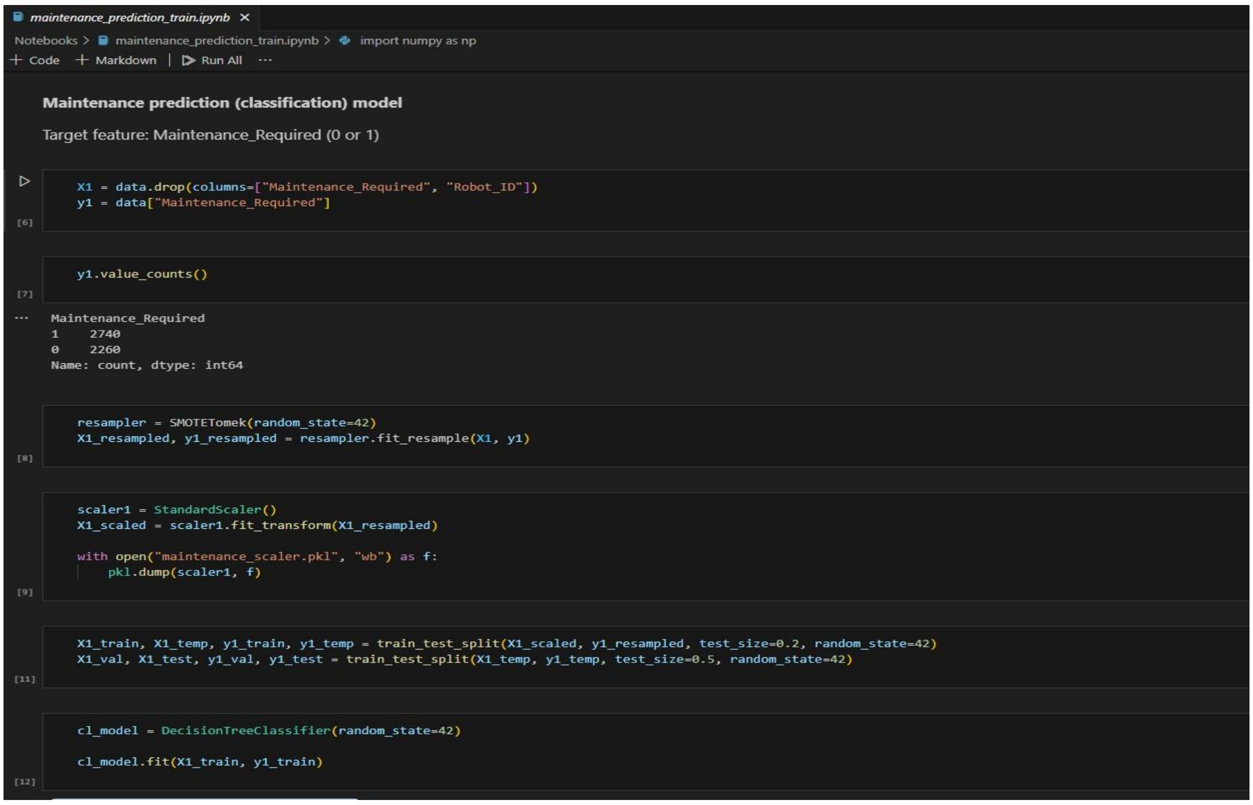
```python
data['Maintenance_Required'] = ((data['Motor_Temperature'] > 40) |
                               (data['Incident_Frequency'] > 2) |
                               (data['Obstacle_Encounters'] > 5)).astype(int)

random_flip = np.random.rand(n_samples) < 0.05
data.loc[random_flip, 'Maintenance_Required'] = 1 - data.loc[random_flip, 'Maintenance_Required']

data.to_csv("robot_monitoring_dataset.csv", index=False)
```

```python
print(data.head())
```

```
   Robot_ID  Power_Consumption  Obstacle_Encounters  Incident_Frequency  \
0         3          62.788099                    0                   0
1         4          70.835219                    2                   0
2         2          69.361316                    2                   0
3         4          61.331295                    3                   1
4         4          72.966360                    1                   0

   Motor_Temperature  Battery_Usage  Maintenance_Required
0          37.647758       8.465399                     0
1          38.081302      14.898787                     0
2          41.835216      15.877903                     1
3          42.049322      25.961210                     1
4          44.926278      12.944068                     1
```

```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Robot_ID              5000 non-null   int32
 1   Power_Consumption     5000 non-null   float64
 2   Obstacle_Encounters   5000 non-null   int32
 3   Incident_Frequency    5000 non-null   int32
 4   Motor_Temperature     5000 non-null   float64
 5   Battery_Usage         5000 non-null   float64
 6   Maintenance_Required  5000 non-null   int32
dtypes: float64(3), int32(4)
```



maintenance_prediction_train.ipynb ×

Notebooks > maintenance_prediction_train.ipynb > import numpy as np

+ Code + Markdown | ▷ Run All ···

Maintenance prediction (classification) model

Target feature: Maintenance_Required (0 or 1)

```
[6]
X1 = data.drop(columns=["Maintenance_Required", "Robot_ID"])
y1 = data["Maintenance_Required"]
```

```
[7]
y1.value_counts()
```

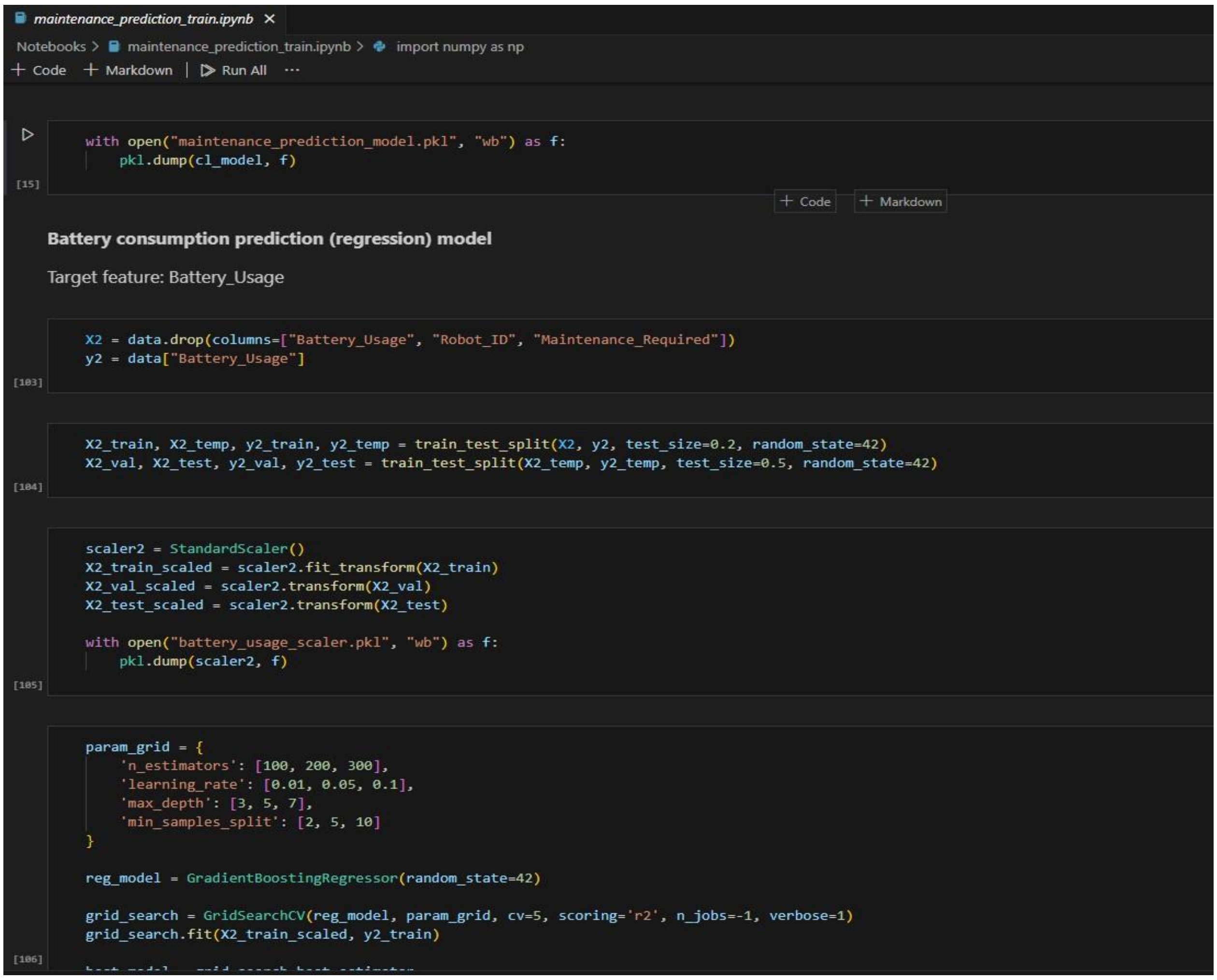
```
Maintenance_Required
1    2740
0    2260
Name: count, dtype: int64
```

```
[8]
resampler = SMOTETomek(random_state=42)
X1_resampled, y1_resampled = resampler.fit_resample(X1, y1)
```

```
[9]
scaler1 = StandardScaler()
X1_scaled = scaler1.fit_transform(X1_resampled)

with open("maintenance_scaler.pkl", "wb") as f:
    pkl.dump(scaler1, f)
```

```
[11]
X1_train, X1_temp, y1_train, y1_temp = train_test_split(X1_scaled, y1_resampled, test_size=0.2, random_state=42)
X1_val, X1_test, y1_val, y1_test = train_test_split(X1_temp, y1_temp, test_size=0.5, random_state=42)
```

```
[12]
cl_model = DecisionTreeClassifier(random_state=42)

cl_model.fit(X1_train, y1_train)
```



maintenance_prediction_train.ipynb ×

Notebooks > maintenance_prediction_train.ipynb > import numpy as np

+ Code + Markdown | ▷ Run All ···

```
[15]
with open("maintenance_prediction_model.pkl", "wb") as f:
    pkl.dump(cl_model, f)
```

+ Code + Markdown

Battery consumption prediction (regression) model

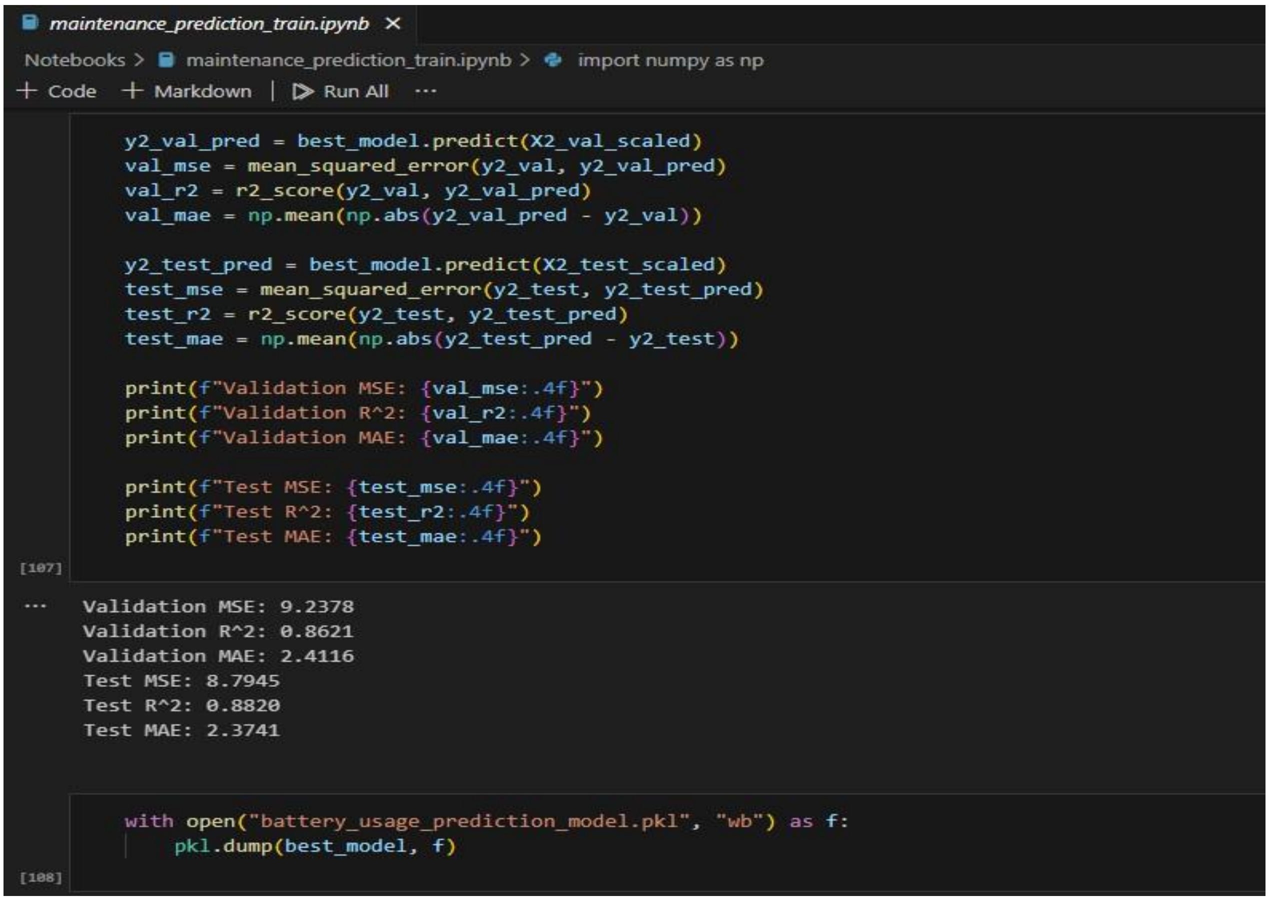
Target feature: Battery_Usage

```
[103]
X2 = data.drop(columns=["Battery_Usage", "Robot_ID", "Maintenance_Required"])
y2 = data["Battery_Usage"]
```

```
[104]
X2_train, X2_temp, y2_train, y2_temp = train_test_split(X2, y2, test_size=0.2, random_state=42)
X2_val, X2_test, y2_val, y2_test = train_test_split(X2_temp, y2_temp, test_size=0.5, random_state=42)
```

```
[105]
scaler2 = StandardScaler()
X2_train_scaled = scaler2.fit_transform(X2_train)
X2_val_scaled = scaler2.transform(X2_val)
X2_test_scaled = scaler2.transform(X2_test)

with open("battery_usage_scaler.pkl", "wb") as f:
    pkl.dump(scaler2, f)
```

```
[106]
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10]
}

reg_model = GradientBoostingRegressor(random_state=42)

grid_search = GridSearchCV(reg_model, param_grid, cv=5, scoring='r2', n_jobs=-1, verbose=1)
grid_search.fit(X2_train_scaled, y2_train)
```


maintenance_prediction_train.ipynb ×
Notebooks > maintenance_prediction_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```python
y2_val_pred = best_model.predict(X2_val_scaled)
val_mse = mean_squared_error(y2_val, y2_val_pred)
val_r2 = r2_score(y2_val, y2_val_pred)
val_mae = np.mean(np.abs(y2_val_pred - y2_val))

y2_test_pred = best_model.predict(X2_test_scaled)
test_mse = mean_squared_error(y2_test, y2_test_pred)
test_r2 = r2_score(y2_test, y2_test_pred)
test_mae = np.mean(np.abs(y2_test_pred - y2_test))

print(f"Validation MSE: {val_mse:.4f}")
print(f"Validation R^2: {val_r2:.4f}")
print(f"Validation MAE: {val_mae:.4f}")

print(f"Test MSE: {test_mse:.4f}")
print(f"Test R^2: {test_r2:.4f}")
print(f"Test MAE: {test_mae:.4f}")
```

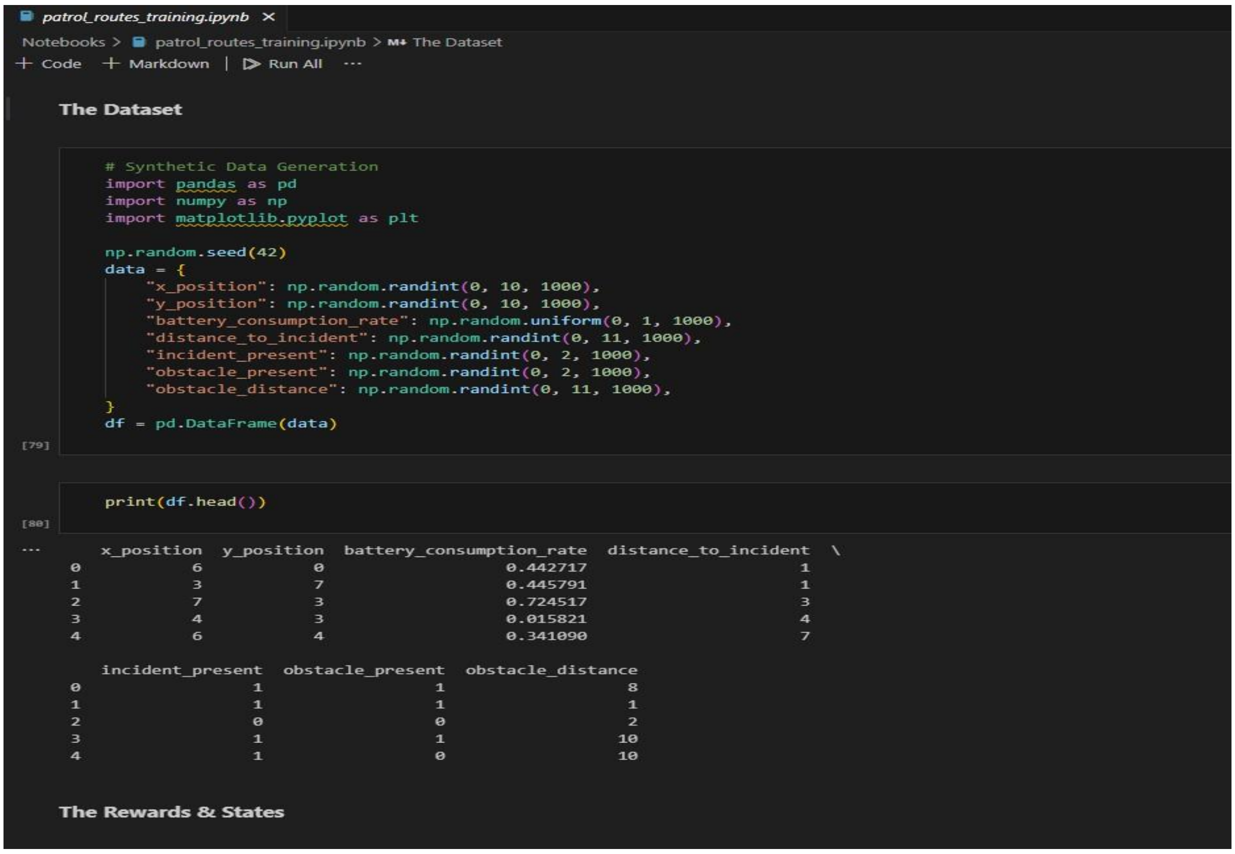
[107]

```
Validation MSE: 9.2378
Validation R^2: 0.8621
Validation MAE: 2.4116
Test MSE: 8.7945
Test R^2: 0.8820
Test MAE: 2.3741
```

```python
with open("battery_usage_prediction_model.pkl", "wb") as f:
    pkl.dump(best_model, f)
```

[108]

3. Patrol route training


patrol_routes_training.ipynb ×
Notebooks > patrol_routes_training.ipynb > M↓ The Dataset
+ Code + Markdown | Run All ···

The Dataset

```python
# Synthetic Data Generation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
data = {
    "x_position": np.random.randint(0, 10, 1000),
    "y_position": np.random.randint(0, 10, 1000),
    "battery_consumption_rate": np.random.uniform(0, 1, 1000),
    "distance_to_incident": np.random.randint(0, 11, 1000),
    "incident_present": np.random.randint(0, 2, 1000),
    "obstacle_present": np.random.randint(0, 2, 1000),
    "obstacle_distance": np.random.randint(0, 11, 1000),
}
df = pd.DataFrame(data)
```

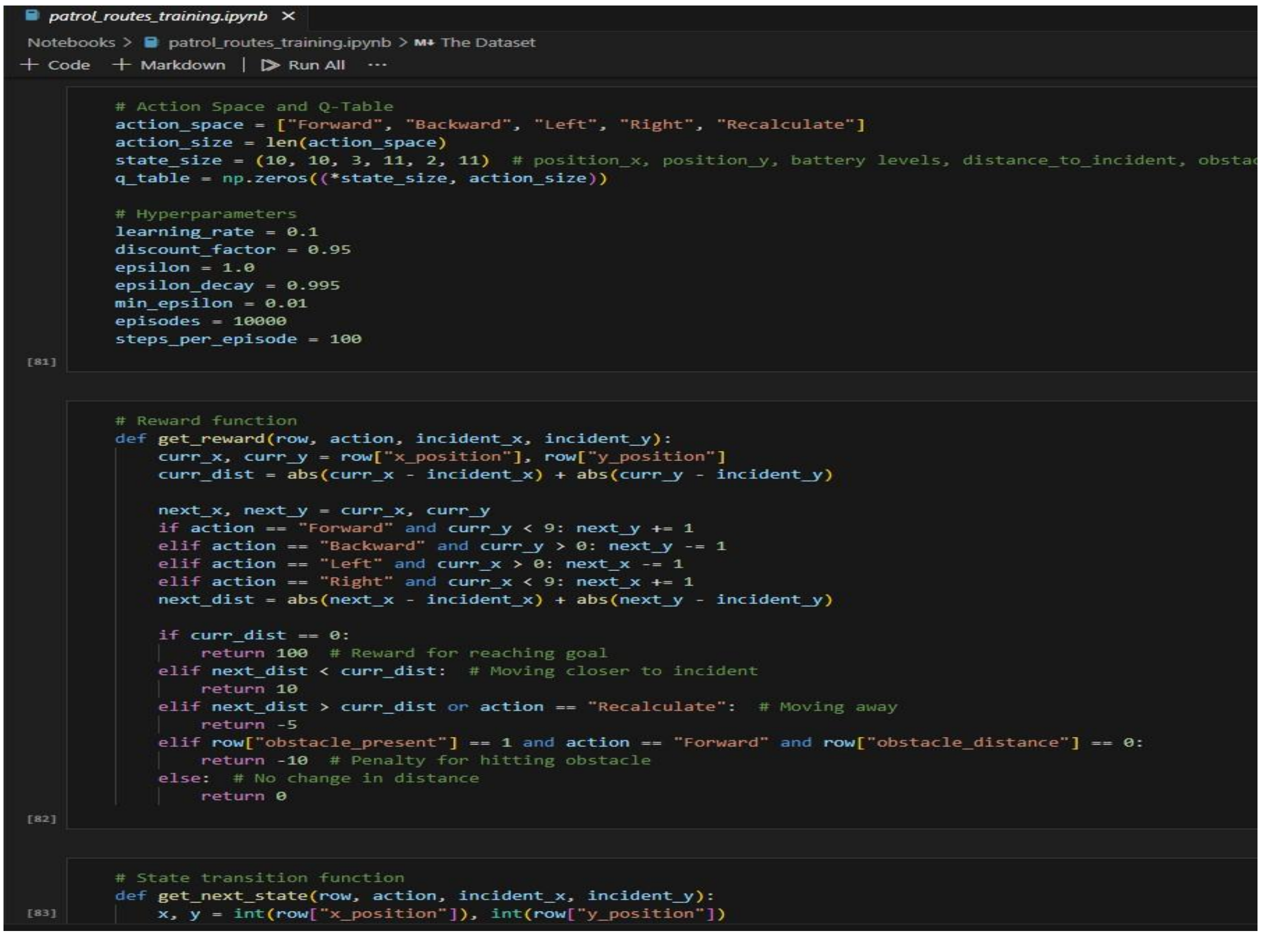
[79]

```python
print(df.head())
```

[80]

```
   x_position  y_position  battery_consumption_rate  distance_to_incident  \
0           6           0                  0.442717                     1
1           3           7                  0.445791                     1
2           7           3                  0.724517                     3
3           4           3                  0.015821                     4
4           6           4                  0.341090                     7

   incident_present  obstacle_present  obstacle_distance
0                 1                 1                  8
1                 1                 1                  1
2                 0                 0                  2
3                 1                 1                 10
4                 1                 0                 10
```

The Rewards & States


patrol_routes_training.ipynb ×

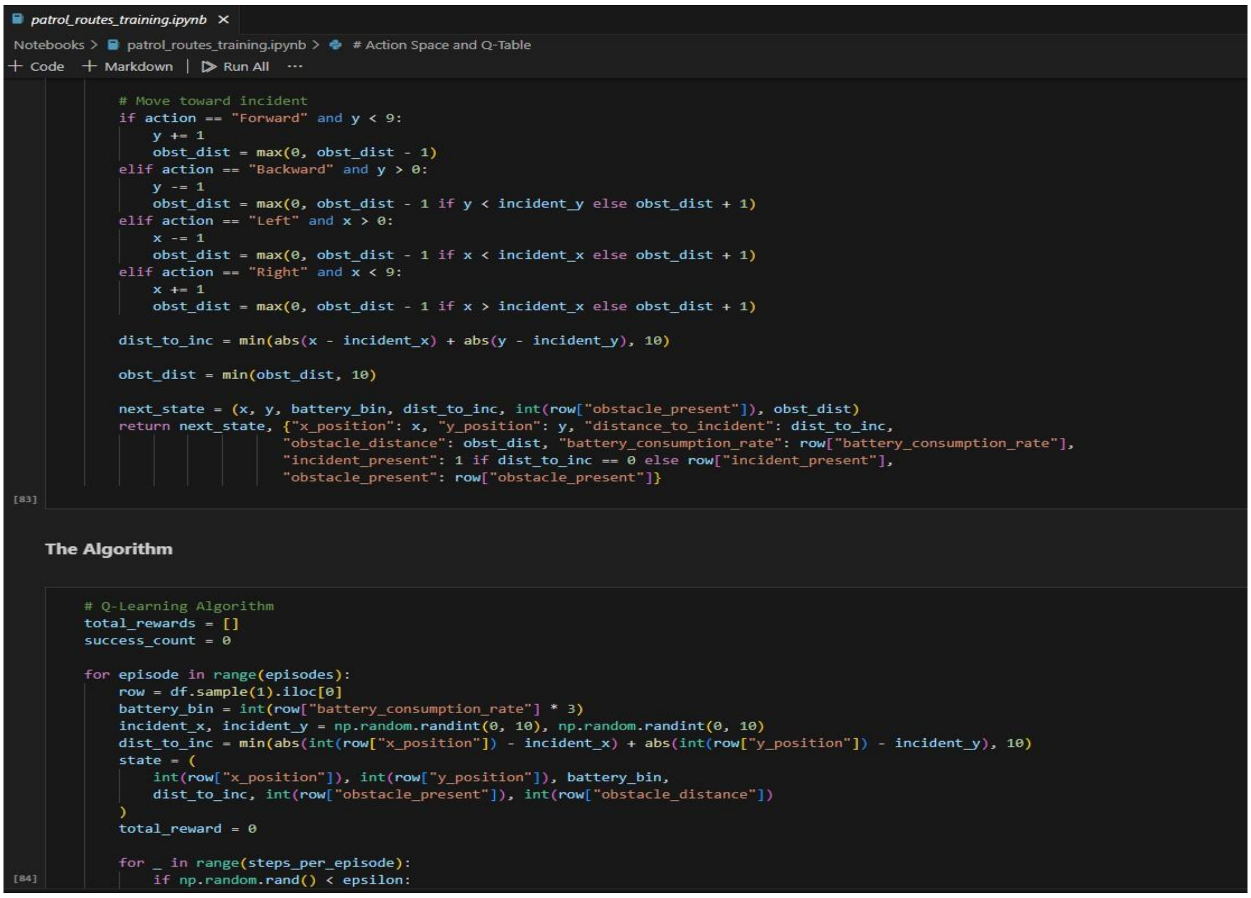
Notebooks > patrol_routes_training.ipynb > M↓ The Dataset

+ Code + Markdown | ▷ Run All ···

```python
# Action Space and Q-Table
action_space = ["Forward", "Backward", "Left", "Right", "Recalculate"]
action_size = len(action_space)
state_size = (10, 10, 3, 11, 2, 11)  # position_x, position_y, battery levels, distance_to_incident, obstac
q_table = np.zeros((*state_size, action_size))

# Hyperparameters
learning_rate = 0.1
discount_factor = 0.95
epsilon = 1.0
epsilon_decay = 0.995
min_epsilon = 0.01
episodes = 10000
steps_per_episode = 100
```
[81]

```python
# Reward function
def get_reward(row, action, incident_x, incident_y):
    curr_x, curr_y = row["x_position"], row["y_position"]
    curr_dist = abs(curr_x - incident_x) + abs(curr_y - incident_y)

    next_x, next_y = curr_x, curr_y
    if action == "Forward" and curr_y < 9: next_y += 1
    elif action == "Backward" and curr_y > 0: next_y -= 1
    elif action == "Left" and curr_x > 0: next_x -= 1
    elif action == "Right" and curr_x < 9: next_x += 1
    next_dist = abs(next_x - incident_x) + abs(next_y - incident_y)

    if curr_dist == 0:
        return 100  # Reward for reaching goal
    elif next_dist < curr_dist:  # Moving closer to incident
        return 10
    elif next_dist > curr_dist or action == "Recalculate":  # Moving away
        return -5
    elif row["obstacle_present"] == 1 and action == "Forward" and row["obstacle_distance"] == 0:
        return -10  # Penalty for hitting obstacle
    else:  # No change in distance
        return 0
```
[82]

```python
# State transition function
def get_next_state(row, action, incident_x, incident_y):
    x, y = int(row["x_position"]), int(row["y_position"])
```
[83]


patrol_routes_training.ipynb ×

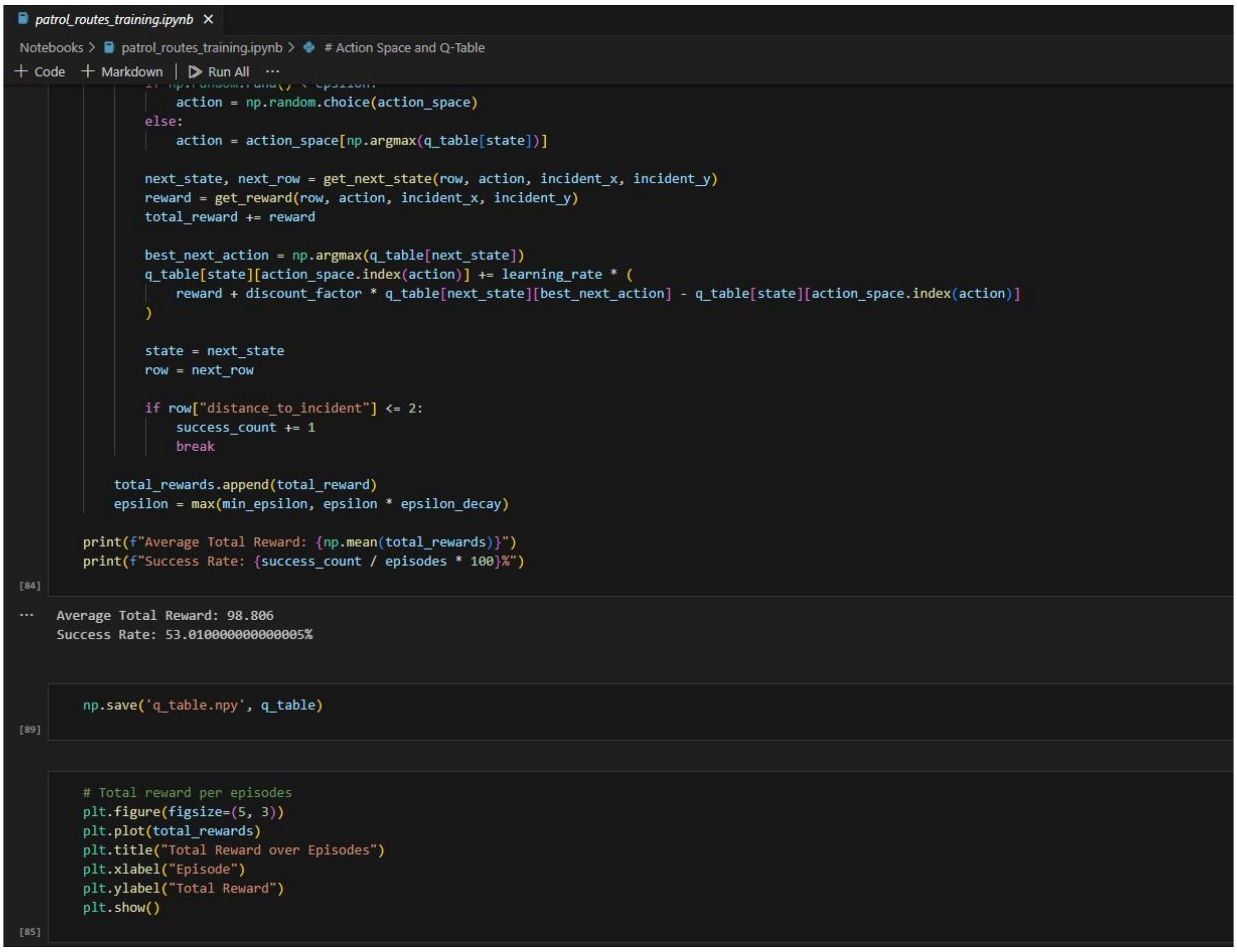
Notebooks > patrol_routes_training.ipynb > # Action Space and Q-Table

+ Code + Markdown | ▷ Run All ···

```python
    # Move toward incident
    if action == "Forward" and y < 9:
        y += 1
        obst_dist = max(0, obst_dist - 1)
    elif action == "Backward" and y > 0:
        y -= 1
        obst_dist = max(0, obst_dist - 1 if y < incident_y else obst_dist + 1)
    elif action == "Left" and x > 0:
        x -= 1
        obst_dist = max(0, obst_dist - 1 if x < incident_x else obst_dist + 1)
    elif action == "Right" and x < 9:
        x += 1
        obst_dist = max(0, obst_dist - 1 if x > incident_x else obst_dist + 1)

    dist_to_inc = min(abs(x - incident_x) + abs(y - incident_y), 10)

    obst_dist = min(obst_dist, 10)

    next_state = (x, y, battery_bin, dist_to_inc, int(row["obstacle_present"]), obst_dist)
    return next_state, {"x_position": x, "y_position": y, "distance_to_incident": dist_to_inc,
                        "obstacle_distance": obst_dist, "battery_consumption_rate": row["battery_consumption_rate"],
                        "incident_present": 1 if dist_to_inc == 0 else row["incident_present"],
                        "obstacle_present": row["obstacle_present"]}
```
[83]

The Algorithm

```python
# Q-Learning Algorithm
total_rewards = []
success_count = 0

for episode in range(episodes):
    row = df.sample(1).iloc[0]
    battery_bin = int(row["battery_consumption_rate"] * 3)
    incident_x, incident_y = np.random.randint(0, 10), np.random.randint(0, 10)
    dist_to_inc = min(abs(int(row["x_position"]) - incident_x) + abs(int(row["y_position"]) - incident_y), 10)
    state = (
        int(row["x_position"]), int(row["y_position"]), battery_bin,
        dist_to_inc, int(row["obstacle_present"]), int(row["obstacle_distance"])
    )
    total_reward = 0

    for _ in range(steps_per_episode):
        if np.random.rand() < epsilon:
```
[84]


patrol_routes_training.ipynb
Notebooks > patrol_routes_training.ipynb > # Action Space and Q-Table
+ Code + Markdown | Run All ···

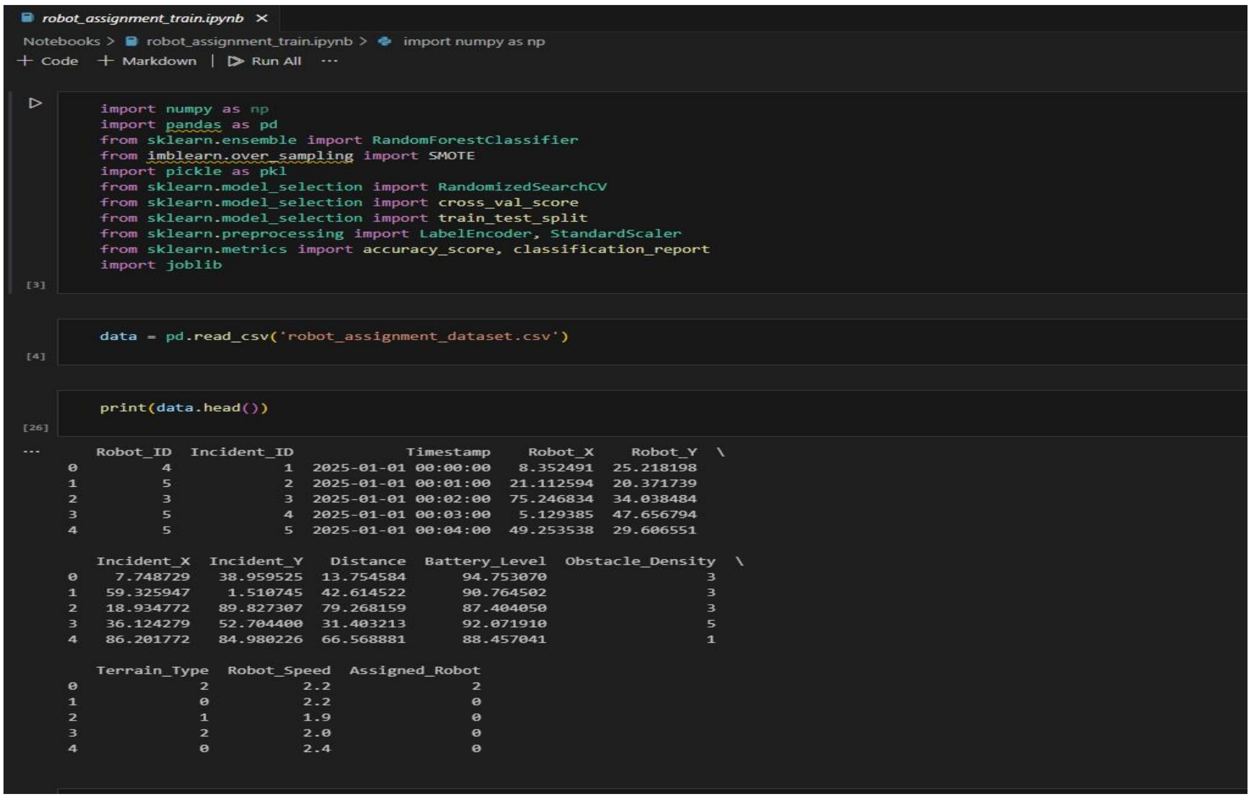
```
            if np.random.rand() < epsilon:
                action = np.random.choice(action_space)
            else:
                action = action_space[np.argmax(q_table[state])]

            next_state, next_row = get_next_state(row, action, incident_x, incident_y)
            reward = get_reward(row, action, incident_x, incident_y)
            total_reward += reward

            best_next_action = np.argmax(q_table[next_state])
            q_table[state][action_space.index(action)] += learning_rate * (
                reward + discount_factor * q_table[next_state][best_next_action] - q_table[state][action_space.index(action)]
            )

            state = next_state
            row = next_row

            if row["distance_to_incident"] <= 2:
                success_count += 1
                break

        total_rewards.append(total_reward)
        epsilon = max(min_epsilon, epsilon * epsilon_decay)

    print(f"Average Total Reward: {np.mean(total_rewards)}")
    print(f"Success Rate: {success_count / episodes * 100}%")
```

```
Average Total Reward: 98.806
Success Rate: 53.010000000000005%
```

```
np.save('q_table.npy', q_table)
```

```
# Total reward per episodes
plt.figure(figsize=(5, 3))
plt.plot(total_rewards)
plt.title("Total Reward over Episodes")
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.show()
```

4. Robot assignment training model

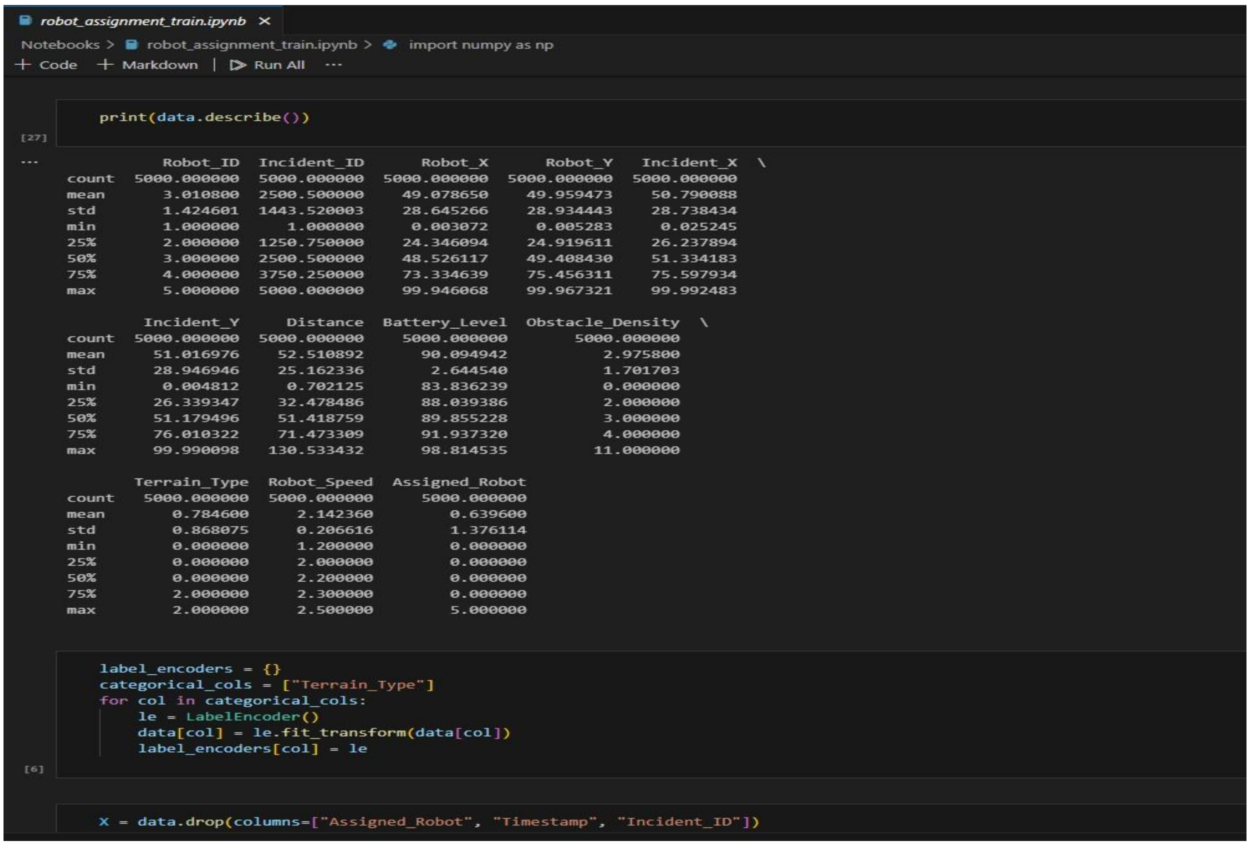

robot_assignment_train.ipynb
Notebooks > robot_assignment_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
import pickle as pkl
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import joblib
```

```
data = pd.read_csv('robot_assignment_dataset.csv')
```

```
print(data.head())
```

```
   Robot_ID  Incident_ID            Timestamp    Robot_X    Robot_Y  \
0         4            1  2025-01-01 00:00:00   8.352491  25.218198
1         5            2  2025-01-01 00:01:00  21.112594  20.371739
2         3            3  2025-01-01 00:02:00  75.246834  34.038484
3         5            4  2025-01-01 00:03:00   5.129385  47.656794
4         5            5  2025-01-01 00:04:00  49.253538  29.606551

   Incident_X  Incident_Y   Distance  Battery_Level  Obstacle_Density  \
0    7.748729   38.959525  13.754584      94.753070                 3
1   59.325947    1.510745  42.614522      90.764502                 3
2   18.934772   89.827307  79.268159      87.404050                 3
3   36.124279   52.704400  31.403213      92.071910                 5
4   86.201772   84.980226  66.568881      88.457041                 1

   Terrain_Type  Robot_Speed  Assigned_Robot
0             2          2.2               2
1             0          2.2               0
2             1          1.9               0
3             2          2.0               0
4             0          2.4               0
```

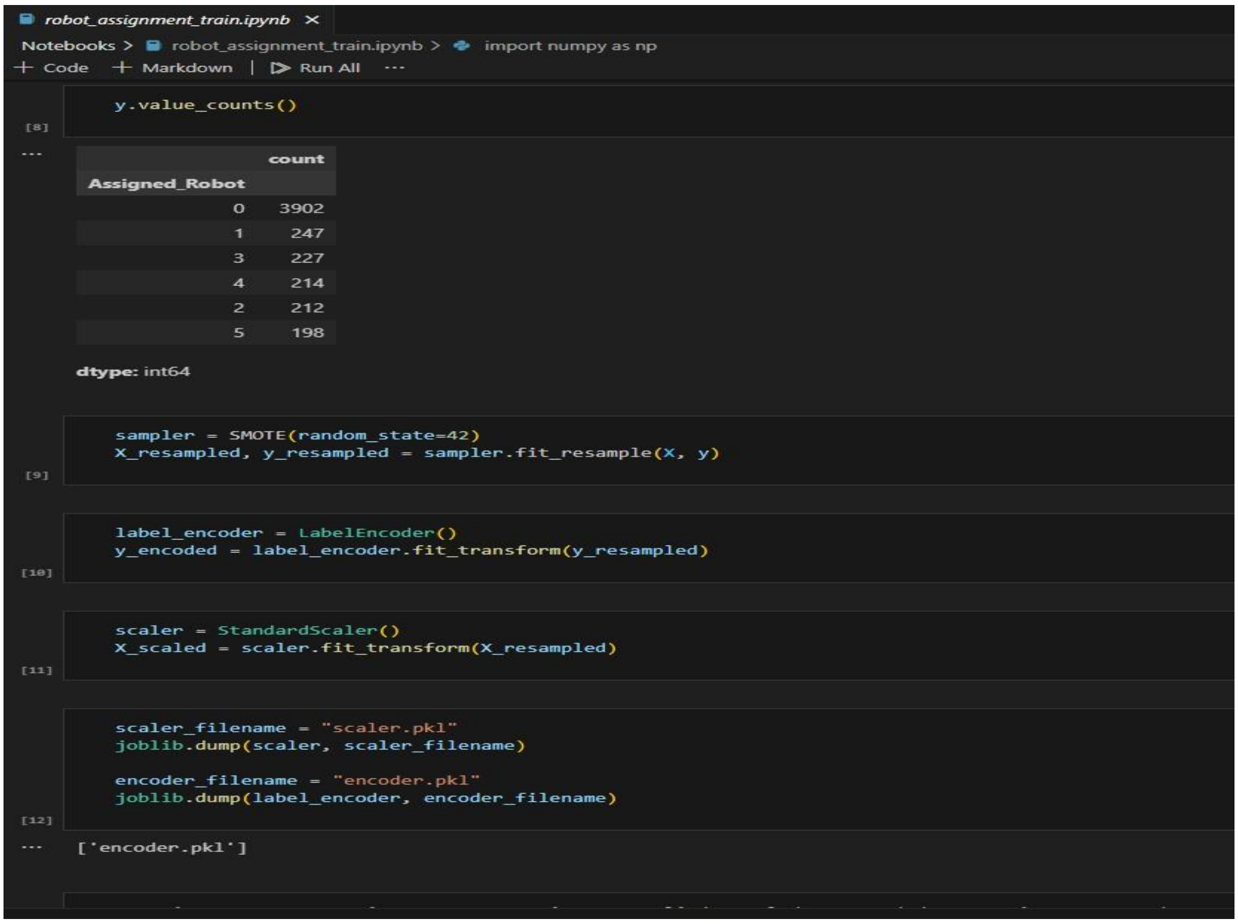

robot_assignment_train.ipynb ×
Notebooks > robot_assignment_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
print(data.describe())
```
[27]

```
         Robot_ID  Incident_ID      Robot_X      Robot_Y   Incident_X  \
count  5000.000000  5000.000000  5000.000000  5000.000000  5000.000000
mean      3.010800  2500.500000    49.078650    49.959473    50.790088
std       1.424601  1443.520003    28.645266    28.934443    28.738434
min       1.000000     1.000000     0.003072     0.005283     0.025245
25%       2.000000  1250.750000    24.346094    24.919611    26.237894
50%       3.000000  2500.500000    48.526117    49.408430    51.334183
75%       4.000000  3750.250000    73.334639    75.456311    75.597934
max       5.000000  5000.000000    99.946068    99.967321    99.992483

        Incident_Y     Distance  Battery_Level  Obstacle_Density  \
count  5000.000000  5000.000000    5000.000000       5000.000000
mean     51.016976    52.510892      90.094942          2.975800
std      28.946946    25.162336       2.644540          1.701703
min       0.004812     0.702125      83.836239          0.000000
25%      26.339347    32.478486      88.039386          2.000000
50%      51.179496    51.418759      89.855228          3.000000
75%      76.010322    71.473309      91.937320          4.000000
max      99.990098   130.533432      98.814535         11.000000

       Terrain_Type  Robot_Speed  Assigned_Robot
count   5000.000000  5000.000000     5000.000000
mean       0.784600     2.142360        0.639600
std        0.868075     0.206616        1.376114
min        0.000000     1.200000        0.000000
25%        0.000000     2.000000        0.000000
50%        0.000000     2.200000        0.000000
75%        2.000000     2.300000        0.000000
max        2.000000     2.500000        5.000000
```

```
label_encoders = {}
categorical_cols = ["Terrain_Type"]
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```
[6]

```
X = data.drop(columns=["Assigned_Robot", "Timestamp", "Incident_ID"])
```


robot_assignment_train.ipynb ×
Notebooks > robot_assignment_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
y.value_counts()
```
[8]

Assigned_Robot	count
0	3902
1	247
3	227
4	214
2	212
5	198

dtype: int64

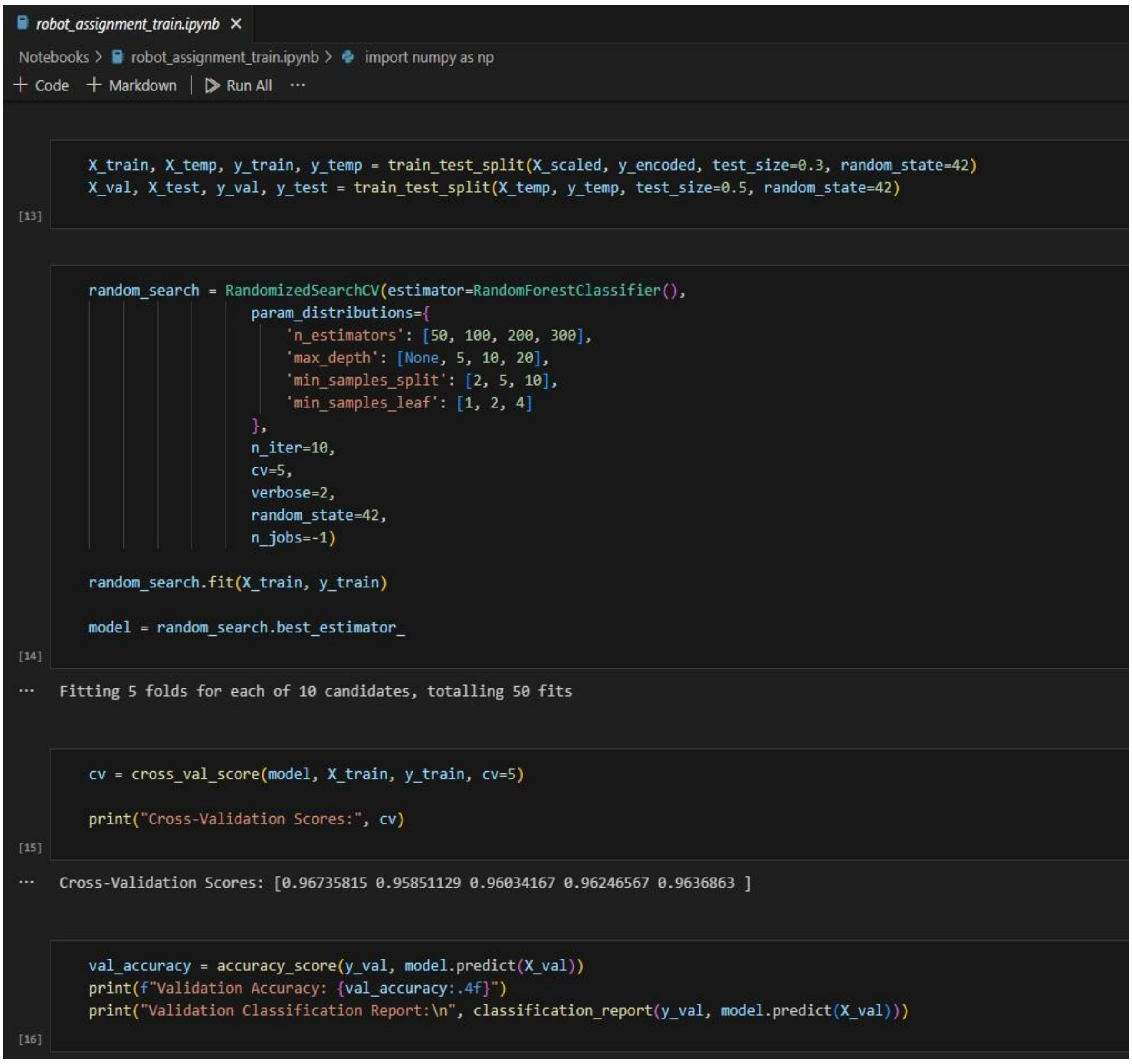
```
sampler = SMOTE(random_state=42)
X_resampled, y_resampled = sampler.fit_resample(X, y)
```
[9]

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_resampled)
```
[10]

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_resampled)
```
[11]

```
scaler_filename = "scaler.pkl"
joblib.dump(scaler, scaler_filename)

encoder_filename = "encoder.pkl"
joblib.dump(label_encoder, encoder_filename)
```
[12]

```
['encoder.pkl']
```


robot_assignment_train.ipynb
Notebooks > robot_assignment_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y_encoded, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```
[13]

```
random_search = RandomizedSearchCV(estimator=RandomForestClassifier(),
                    param_distributions={
                        'n_estimators': [50, 100, 200, 300],
                        'max_depth': [None, 5, 10, 20],
                        'min_samples_split': [2, 5, 10],
                        'min_samples_leaf': [1, 2, 4]
                    },
                    n_iter=10,
                    cv=5,
                    verbose=2,
                    random_state=42,
                    n_jobs=-1)

random_search.fit(X_train, y_train)

model = random_search.best_estimator_
```
[14]

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
cv = cross_val_score(model, X_train, y_train, cv=5)

print("Cross-Validation Scores:", cv)
```
[15]

```
Cross-Validation Scores: [0.96735815 0.95851129 0.96034167 0.96246567 0.9636863 ]
```

```
val_accuracy = accuracy_score(y_val, model.predict(X_val))
print(f"Validation Accuracy: {val_accuracy:.4f}")
print("Validation Classification Report:\n", classification_report(y_val, model.predict(X_val)))
```
[16]

3.2 Feasibility study

3.2.1 Technical Feasibility

- Knowledge on Raspberry pi

  As the centralized main controller board Raspberry-Pi will be used. This controller board is responsible for data gathering , data processing , decision making , send the collected data to the firebase.

- Knowledge on Arduino

  For the programming of Arduino Nano, we use C++. For sensor integration part C++ language is widely used throughout this research.

- Knowledge on Google firebase

  Collected data (temperature data , gas sensor data , humidity sensor data , pressure sensor data) will be sent to the google firebase. The data will be encrypted before sending.
  Also, the data from the camera will be sent to the cloud.

- Knowledge on Python

  Python will be used as the programming language for Raspberry-pi. Also, we used python when developing machine learning algorithms.

- Knowledge on MQTT

  MQTT will be used to communicate between the main robot system and the slave robot. I use MQTT because it is very simple and MQTT protocol uses very low battery power.

3.3 System development and implementation

Initially all the sensors tested thoroughly for their accuracy before they are integrating to the Arduino Nano board. Sensors are tested compared to the levels stated in the sensor’s datasheet. Out of the tested sensors , sensors which shows maximum reliability were chosen. Sensor testing phase took about 2 weeks. In sensor testing various environmental conditions also tested.

.

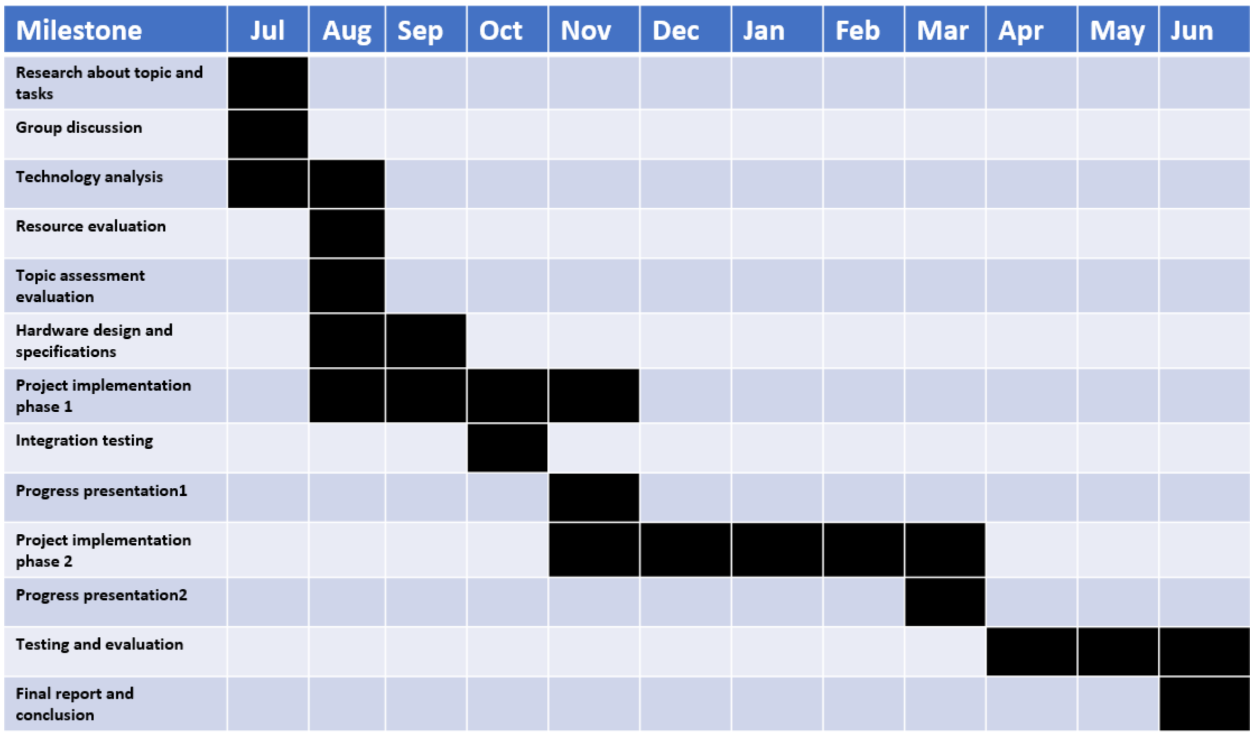
3.4 Project requirements

- User requirements
    - Real-time alerts and warnings.
    - Detection of anomalies
    - Security and safety of the warehouse.
    - Reliability.

- System requirements
    - Reliable environmental detection by gas sensor
    - Compatibility with various warehouse layouts.
    - Detection of suspicious activity in environment.
    - Real-time communication

- Non- functional requirements
    - Support for multiple robots (co-bots)
    - Minimum maintenance
    - Minimum downtime
    - Quick response
    - Encrypted data transfer between database and the robot system.

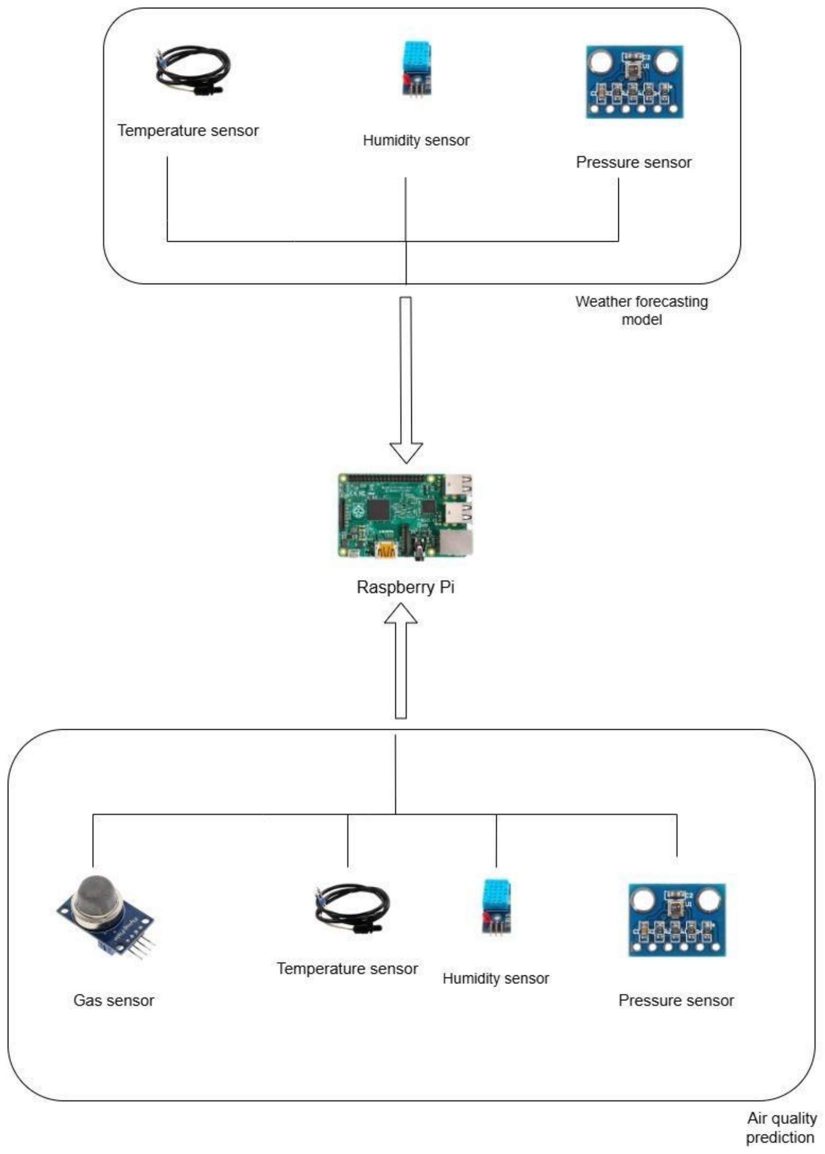
3.5 Timeline



Milestone	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
Research about topic and tasks	■											
Group discussion	■											
Technology analysis	■	■										
Resource evaluation		■										
Topic assessment evaluation		■										
Hardware design and specifications		■	■									
Project implementation phase 1		■	■	■	■							
Integration testing				■								
Progress presentation1					■							
Project implementation phase 2					■	■	■	■	■			
Progress presentation2									■			
Testing and evaluation										■	■	■
Final report and conclusion												■

3.6 Testing

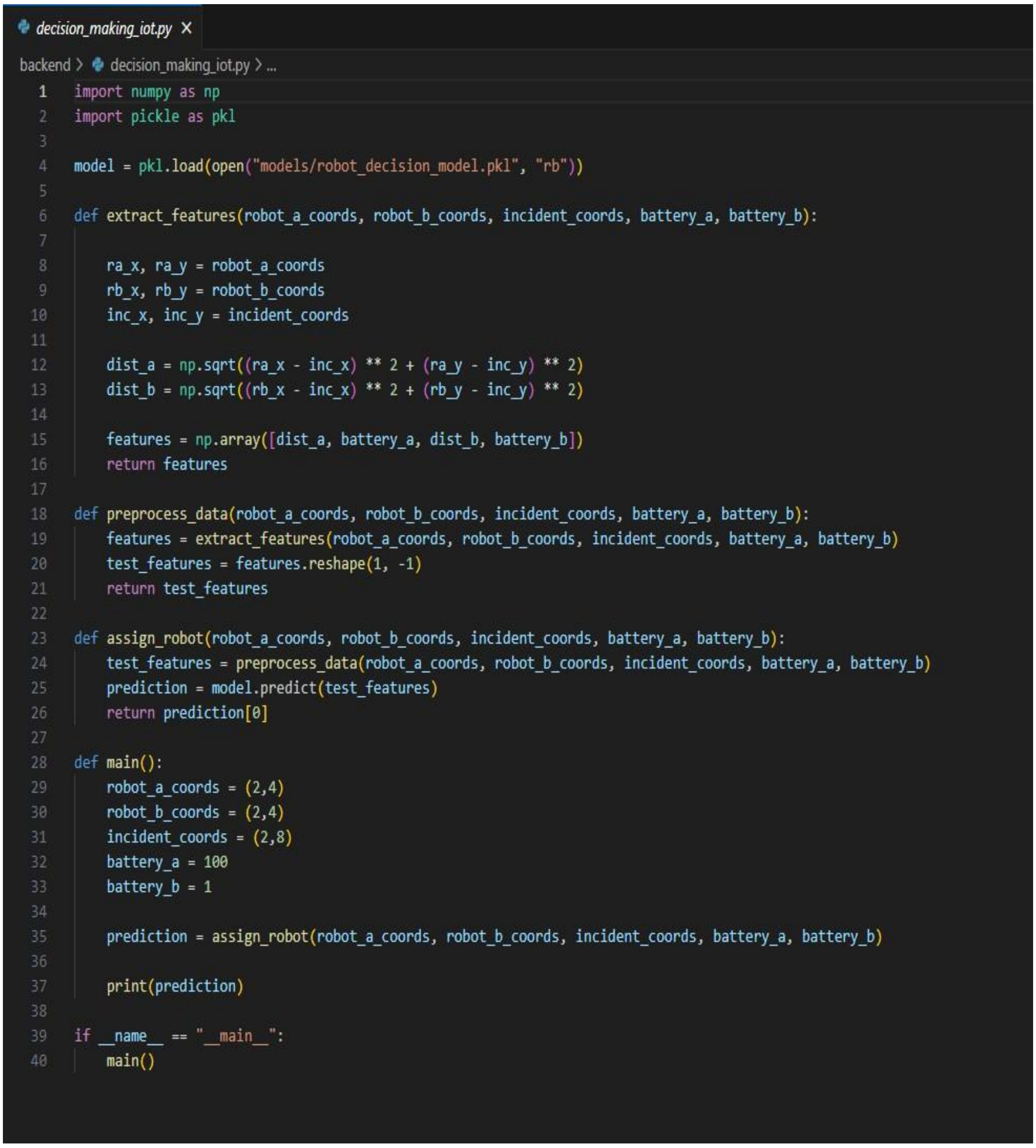
3.6.1 Research Findings



Initially I tested the sensor status and their accuracy for progress presentation I. For that I used an Arduino Uno board instead of Arduino Nano board. Sensors I connected was humidity sensor, temperature sensor, pressure sensor and GPS module. For some reason GPS module didn't work.

Then I mainly work on the decision-making models and event response models.
Decision making mainly work by calculating the distance to the event and comparing the battery levels of each robot with regarding to the distance to the event location.

These machine learning codes are run in Raspberry Pi main unit. We choose a raspberry pi model with 8GB of RAM because these machine learning part uses very high resources.



```python
import numpy as np
import pickle as pkl

model = pkl.load(open("models/robot_decision_model.pkl", "rb"))

def extract_features(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b):

    ra_x, ra_y = robot_a_coords
    rb_x, rb_y = robot_b_coords
    inc_x, inc_y = incident_coords

    dist_a = np.sqrt((ra_x - inc_x) ** 2 + (ra_y - inc_y) ** 2)
    dist_b = np.sqrt((rb_x - inc_x) ** 2 + (rb_y - inc_y) ** 2)

    features = np.array([dist_a, battery_a, dist_b, battery_b])
    return features

def preprocess_data(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b):
    features = extract_features(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b)
    test_features = features.reshape(1, -1)
    return test_features

def assign_robot(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b):
    test_features = preprocess_data(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b)
    prediction = model.predict(test_features)
    return prediction[0]

def main():
    robot_a_coords = (2,4)
    robot_b_coords = (2,4)
    incident_coords = (2,8)
    battery_a = 100
    battery_b = 1

    prediction = assign_robot(robot_a_coords, robot_b_coords, incident_coords, battery_a, battery_b)

    print(prediction)

if __name__ == "__main__":
    main()
```

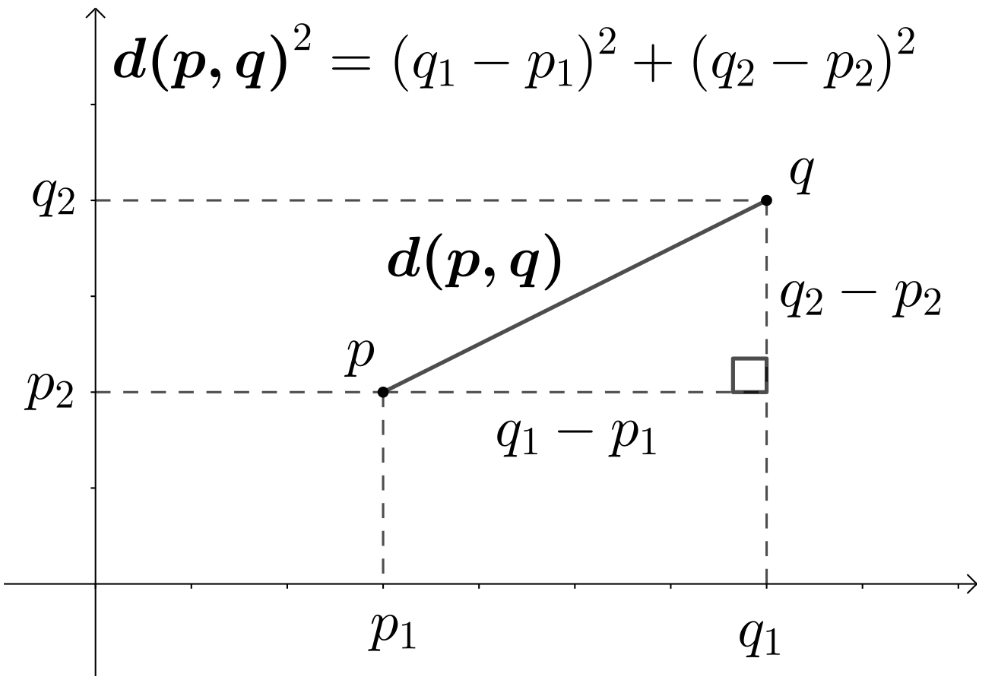
Above model is responsible for determine which robot will be chosen for go to the incident location. This model works by loading a pre trained machine learning model. This model uses pickle library.
Feature array includes,

- Distance to the incident location from robot A
- Distance to the incident location from robot B
- Battery level of robot A
- Battery level of robot B

Using the coordinates of the robot and the coordinates of the incident location the Euclidian distances are calculated.

Euclidian distance calculation…

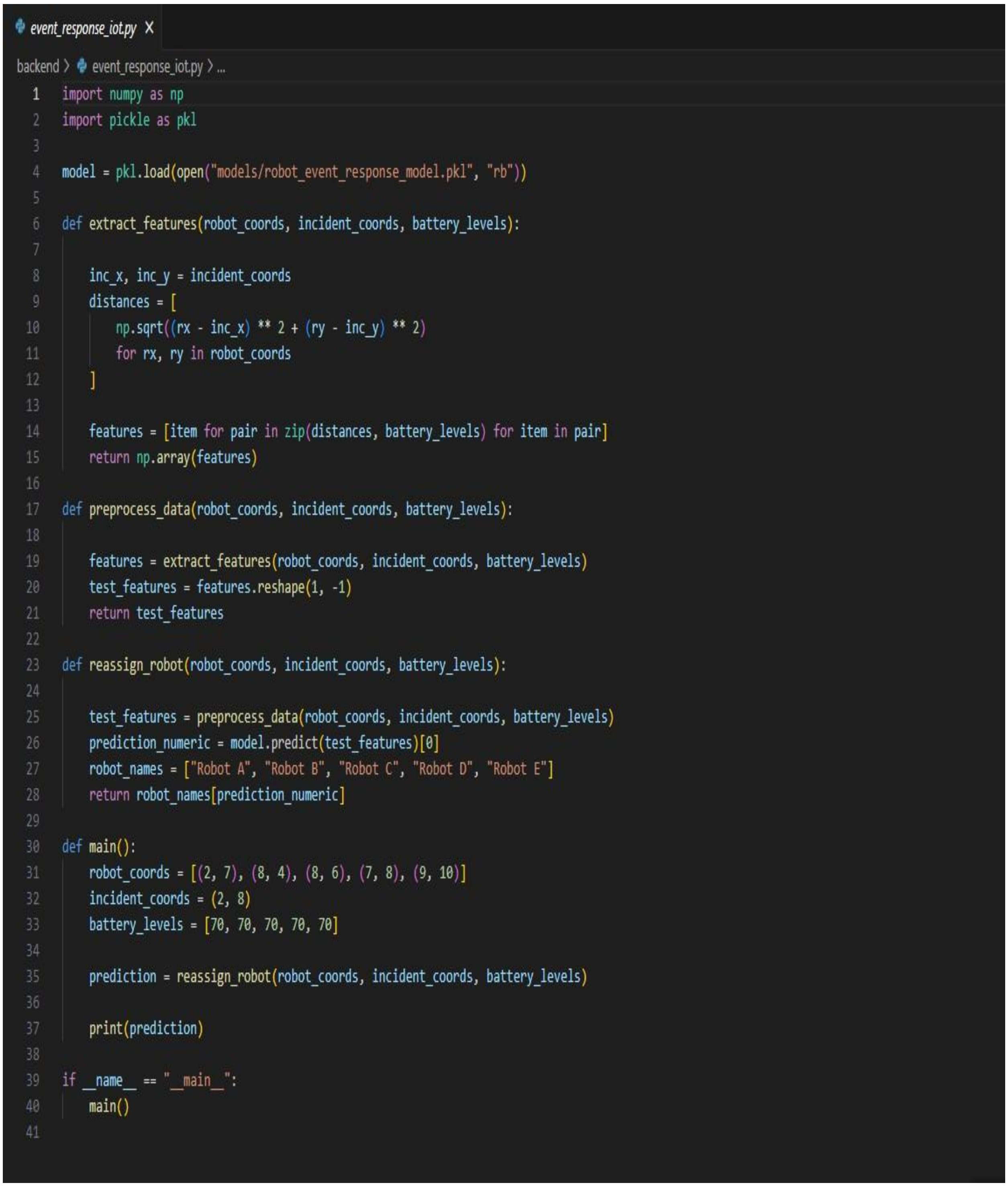
We can use the Pythagorean theorem to compute the Euclidian distance.



We can get the distance by,

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

I also developed event response model which is very similar to the above decision-making model. This model assumes there are more than 2 robots. Then it takes each and every robots co-ordinate and the co-ordinates of the incident. Then this model processes the data gathered and choose the best robot to respond to the event.



event_response_iot.py X

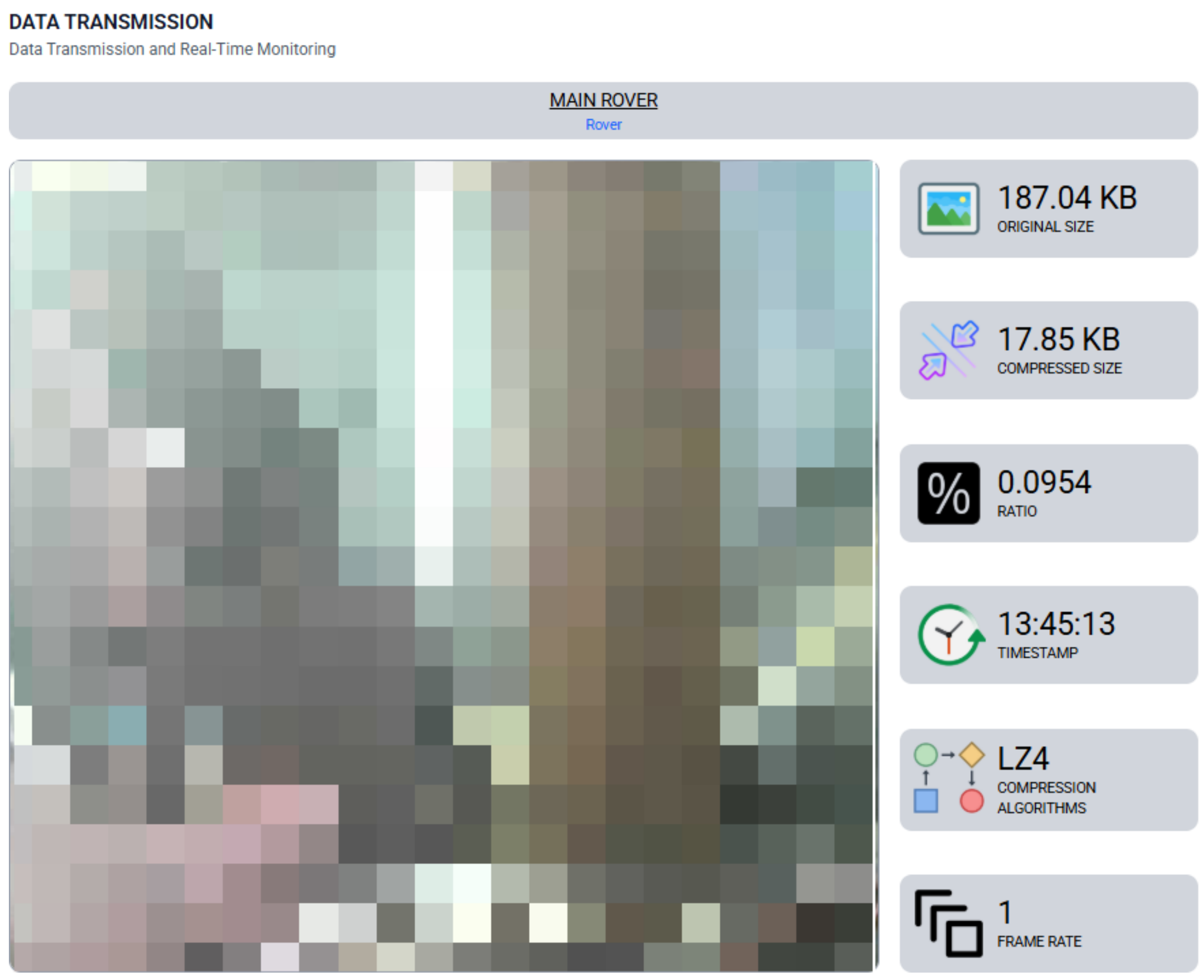
backend > event_response_iot.py > ...

```python
import numpy as np
import pickle as pkl

model = pkl.load(open("models/robot_event_response_model.pkl", "rb"))

def extract_features(robot_coords, incident_coords, battery_levels):

    inc_x, inc_y = incident_coords
    distances = [
        np.sqrt((rx - inc_x) ** 2 + (ry - inc_y) ** 2)
        for rx, ry in robot_coords
    ]

    features = [item for pair in zip(distances, battery_levels) for item in pair]
    return np.array(features)

def preprocess_data(robot_coords, incident_coords, battery_levels):

    features = extract_features(robot_coords, incident_coords, battery_levels)
    test_features = features.reshape(1, -1)
    return test_features

def reassign_robot(robot_coords, incident_coords, battery_levels):

    test_features = preprocess_data(robot_coords, incident_coords, battery_levels)
    prediction_numeric = model.predict(test_features)[0]
    robot_names = ["Robot A", "Robot B", "Robot C", "Robot D", "Robot E"]
    return robot_names[prediction_numeric]

def main():
    robot_coords = [(2, 7), (8, 4), (8, 6), (7, 8), (9, 10)]
    incident_coords = (2, 8)
    battery_levels = [70, 70, 70, 70, 70]

    prediction = reassign_robot(robot_coords, incident_coords, battery_levels)

    print(prediction)

if __name__ == "__main__":
    main()
```

Also, in the web dashboard of the robot which is accessible by http://localhost:4200/ we can get to the,

- Data transmission
- Path navigation
- Internal monitoring
- Slave robot status
- Robot settings

From data transmission tab we can view the latest frame which is sent to the firebase by the robot. There is also information about the original frame size, size of the compressed frame, compression ratio, timestamp, compression algorithm (in our case it is LZ4) and the frame rate.



DATA TRANSMISSION
Data Transmission and Real-Time Monitoring

MAIN ROVER
Rover

187.04 KB
ORIGINAL SIZE

17.85 KB
COMPRESSED SIZE

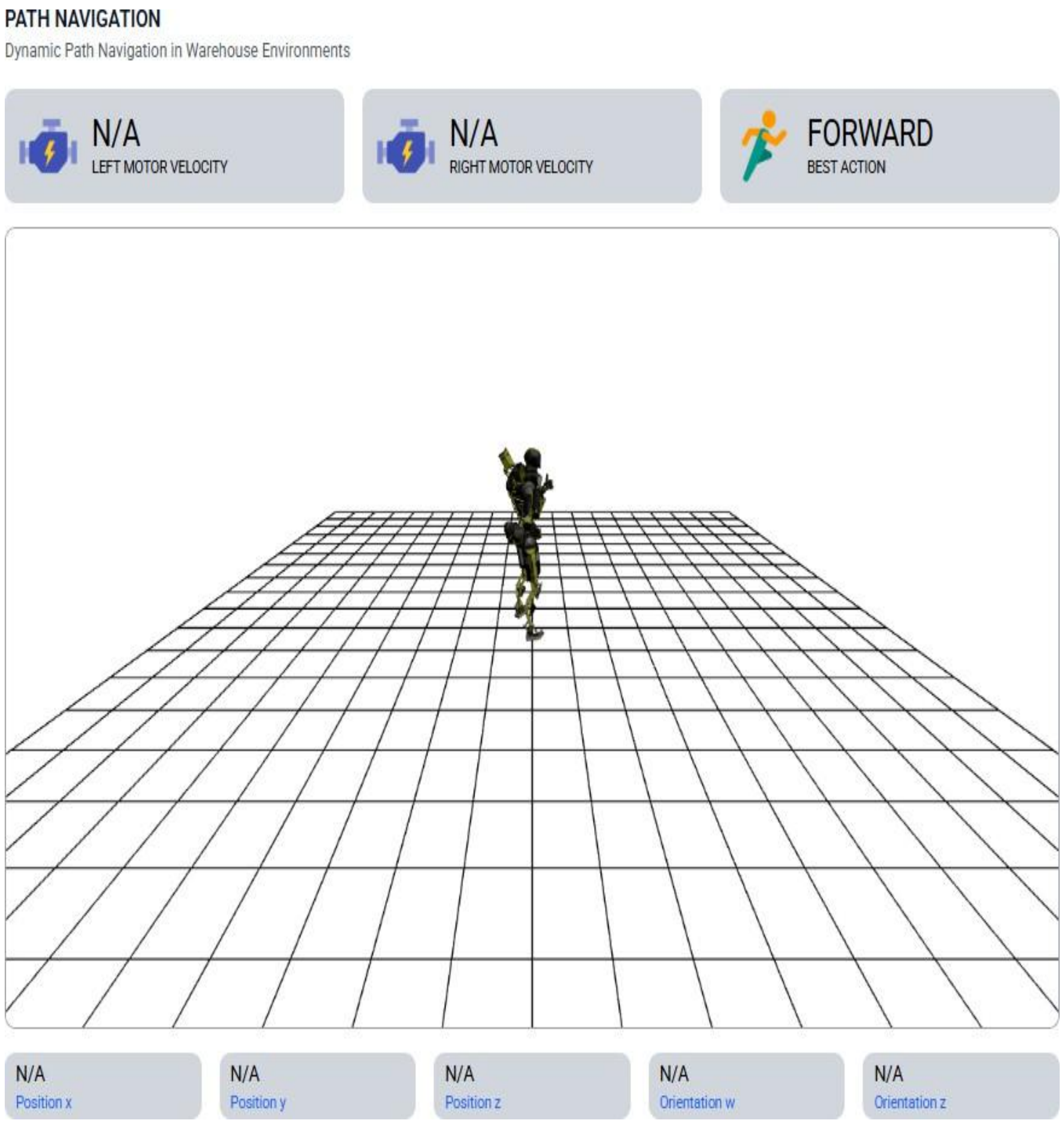
0.0954
RATIO

13:45:13
TIMESTAMP

LZ4
COMPRESSION ALGORITHMS

1
FRAME RATE

From the path navigation tab, we can get the data about left motor velocity, right motor velocity, best action to be taken by the robot etc. It also displays position and the orientation of the robots.


PATH NAVIGATION

Dynamic Path Navigation in Warehouse Environments

N/A
LEFT MOTOR VELOCITY

N/A
RIGHT MOTOR VELOCITY

FORWARD
BEST ACTION

N/A
Position x

N/A
Position y

N/A
Position z

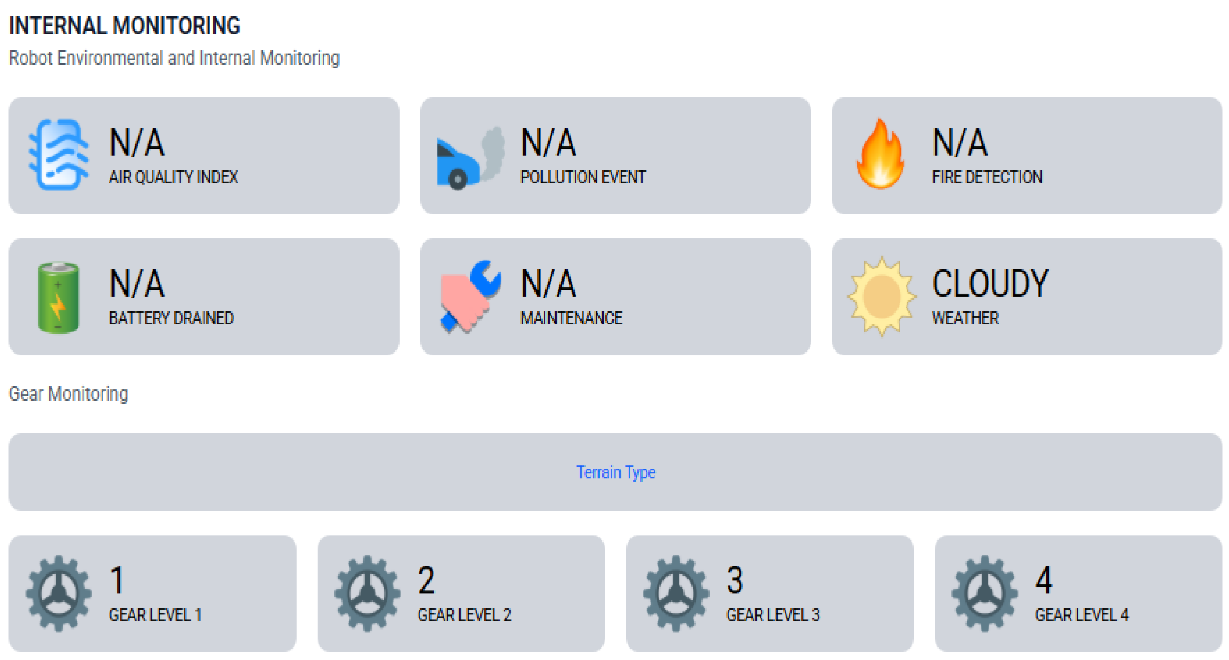
N/A
Orientation w

N/A
Orientation z

From internal monitoring, we can get the idea about

- Air quality index
- Pollution event
- Fire detection
- Percentage of battery drained
- Information about maintenance
- Information about weather
- Terrain type

By getting the temperature and other data from the internal sensors of the robot, it displays here whether a maintenance is required to the robot or not. This is designed to reduce the downtime of the robot.



INTERNAL MONITORING

Robot Environmental and Internal Monitoring

N/A AIR QUALITY INDEX

N/A POLLUTION EVENT

N/A FIRE DETECTION

N/A BATTERY DRAINED

N/A MAINTENANCE

CLOUDY WEATHER

Gear Monitoring

Terrain Type

1 GEAR LEVEL 1

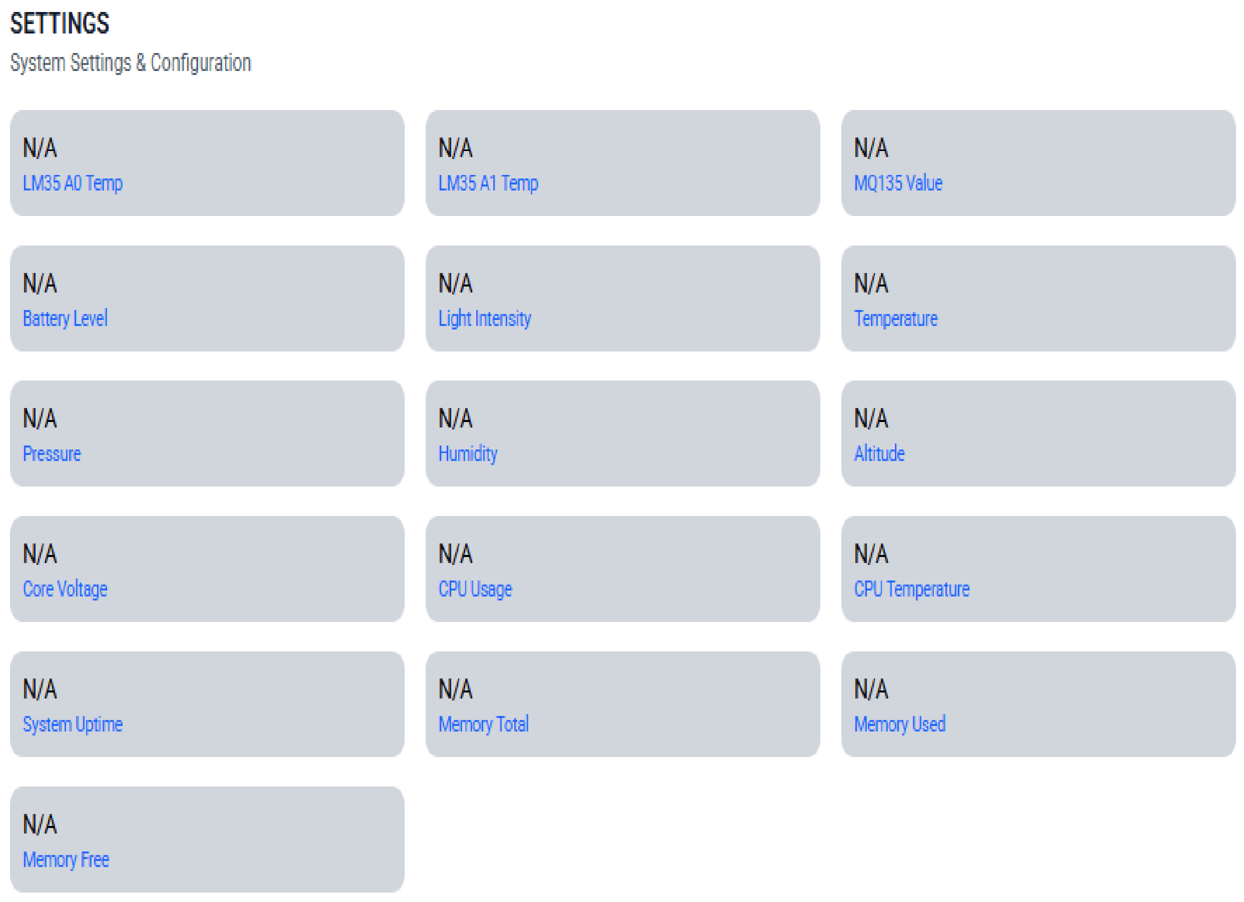
2 GEAR LEVEL 2

3 GEAR LEVEL 3

4 GEAR LEVEL 4

In robot settings we can get the status and the values of,

- Battery level
- Pressure
- Core voltage
- System uptime
- Free memory space
- Light intensity
- Humidity
- CPU usage
- Memory total
- Temperature
- Altitude
- CPU temperature
- Memory used

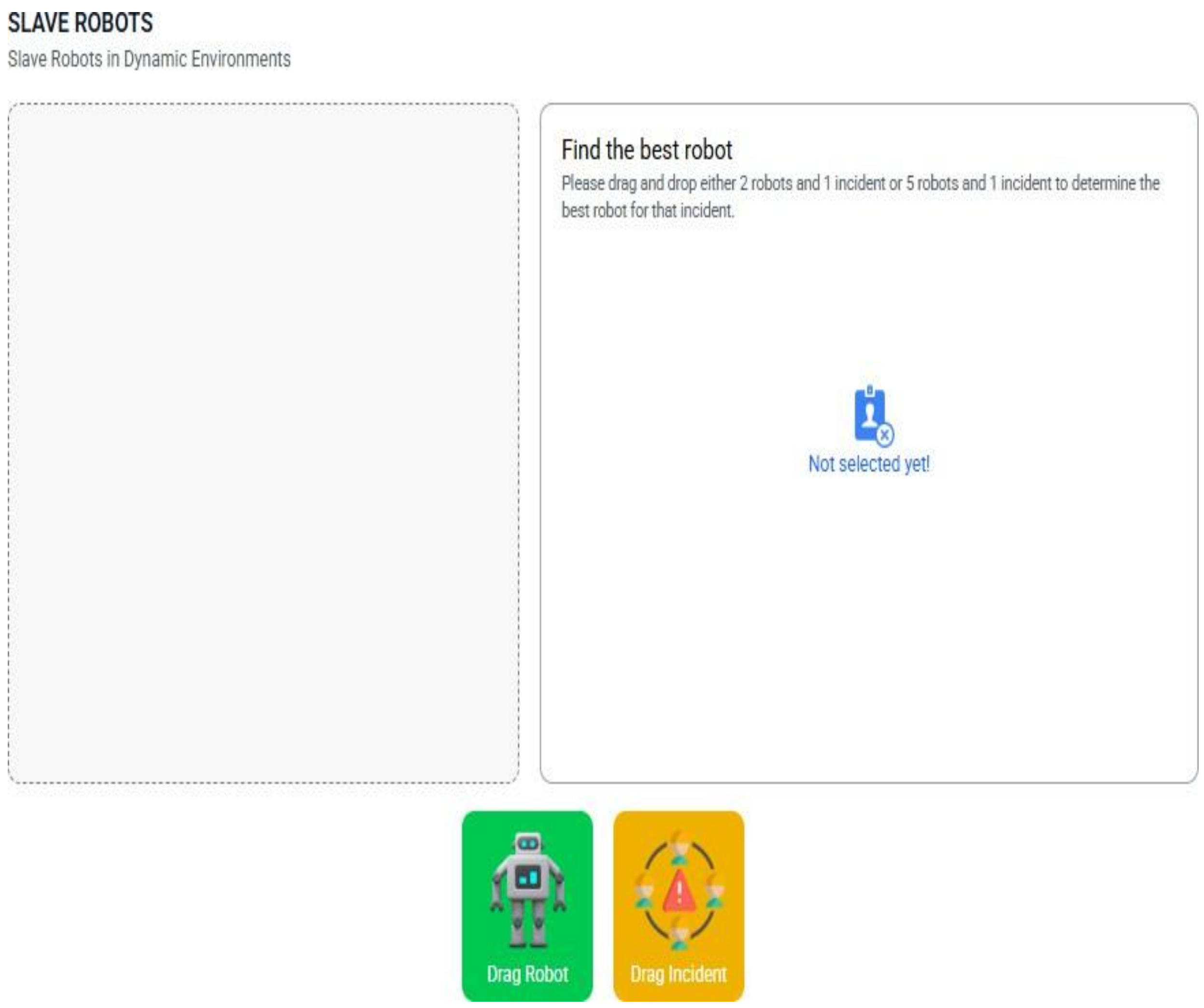

SETTINGS

System Settings & Configuration

N/A LM35 A0 Temp	N/A LM35 A1 Temp	N/A MQ135 Value
N/A Battery Level	N/A Light Intensity	N/A Temperature
N/A Pressure	N/A Humidity	N/A Altitude
N/A Core Voltage	N/A CPU Usage	N/A CPU Temperature
N/A System Uptime	N/A Memory Total	N/A Memory Used
N/A Memory Free		

From slave robot menu item, we can graphically get an idea whether machine learning algorithms are working or not.

We can manually mark a location of robot A, location of the robot B and the location of the incident. Then we have manually entered the battery levels of each robot. After that we can find which robot will be chosen for respond to the incident.

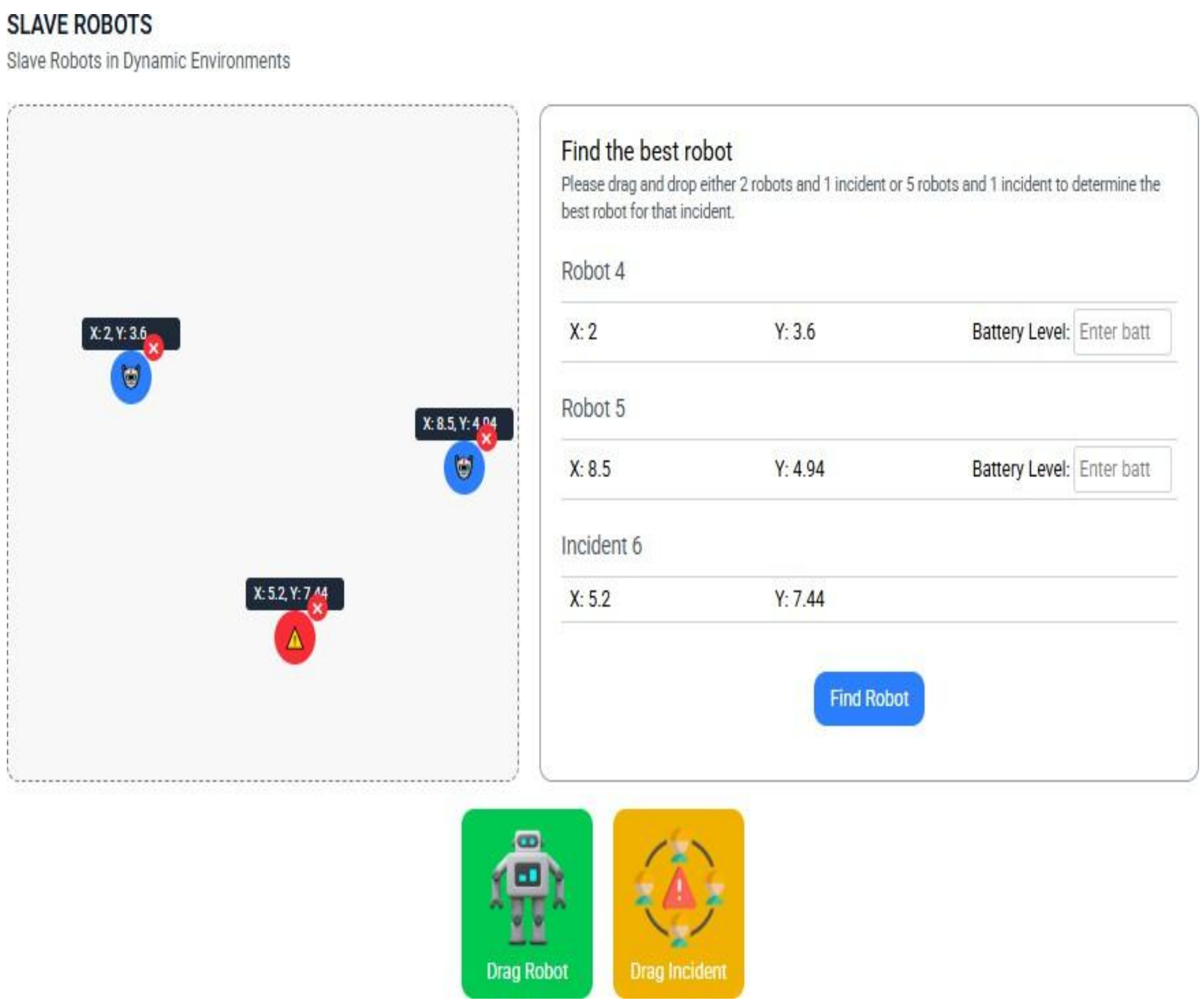

SLAVE ROBOTS

Slave Robots in Dynamic Environments

Find the best robot

Please drag and drop either 2 robots and 1 incident or 5 robots and 1 incident to determine the best robot for that incident.

Not selected yet!


SLAVE ROBOTS

Slave Robots in Dynamic Environments

Find the best robot

Please drag and drop either 2 robots and 1 incident or 5 robots and 1 incident to determine the best robot for that incident.

Robot 4

X: 2 Y: 3.6 Battery Level: Enter batt

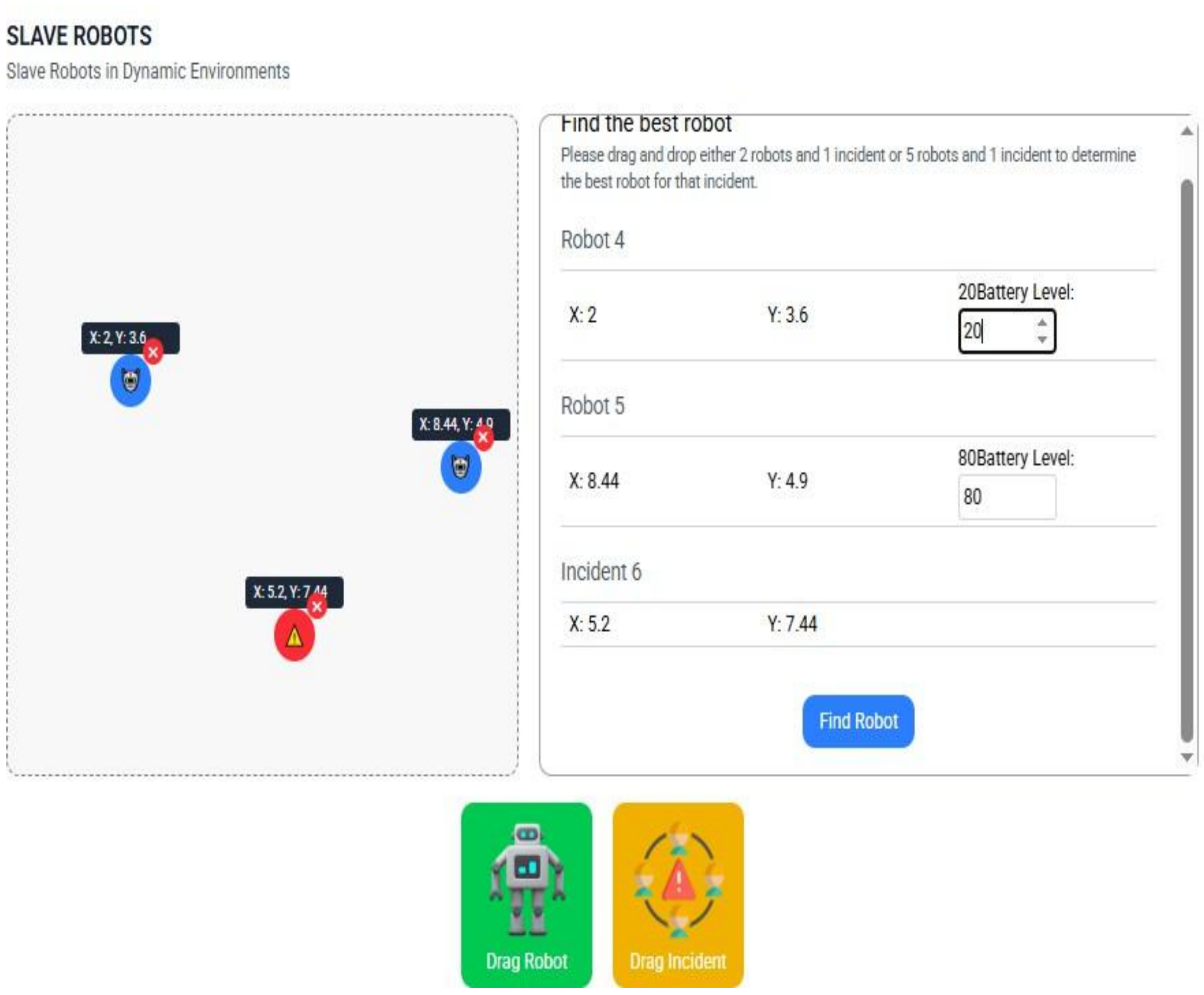
Robot 5

X: 8.5 Y: 4.94 Battery Level: Enter batt

Incident 6

X: 5.2 Y: 7.44

We can place the robots and the incident location anywhere we want.



SLAVE ROBOTS

Slave Robots in Dynamic Environments

Find the best robot

Please drag and drop either 2 robots and 1 incident or 5 robots and 1 incident to determine the best robot for that incident.

Robot 4

X: 2 Y: 3.6 20Battery Level: 20

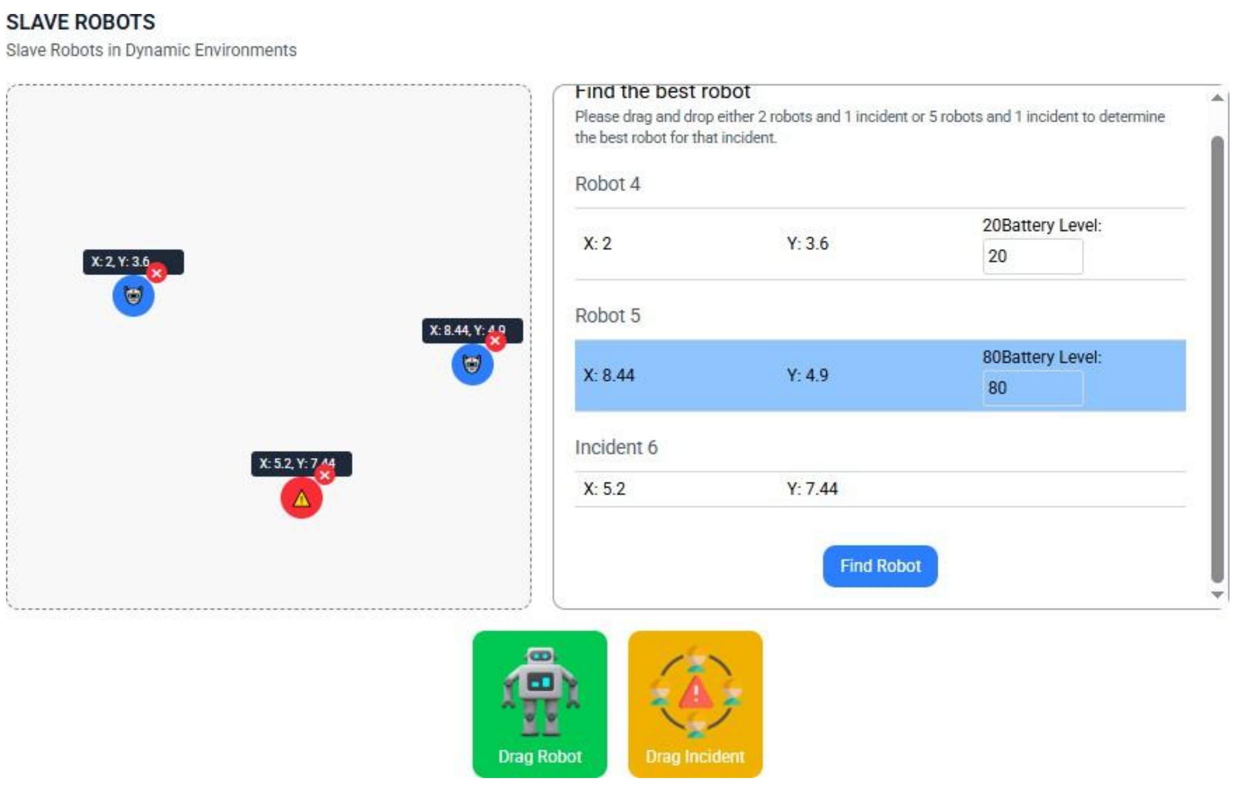
Robot 5

X: 8.44 Y: 4.9 80Battery Level: 80

Incident 6

X: 5.2 Y: 7.44

After that we have to enter the battery levels of each robot.


SLAVE ROBOTS

Slave Robots in Dynamic Environments

Find the best robot

Please drag and drop either 2 robots and 1 incident or 5 robots and 1 incident to determine the best robot for that incident.

Robot 4

X: 2	Y: 3.6	20Battery Level: 20

Robot 5

X: 8.44	Y: 4.9	80Battery Level: 80

Incident 6

X: 5.2	Y: 7.44

For example, I entered 20% battery for robot A and 80% battery level for robot B.

By the algorithm it is decided that robot B is more suitable for respond to the incident.

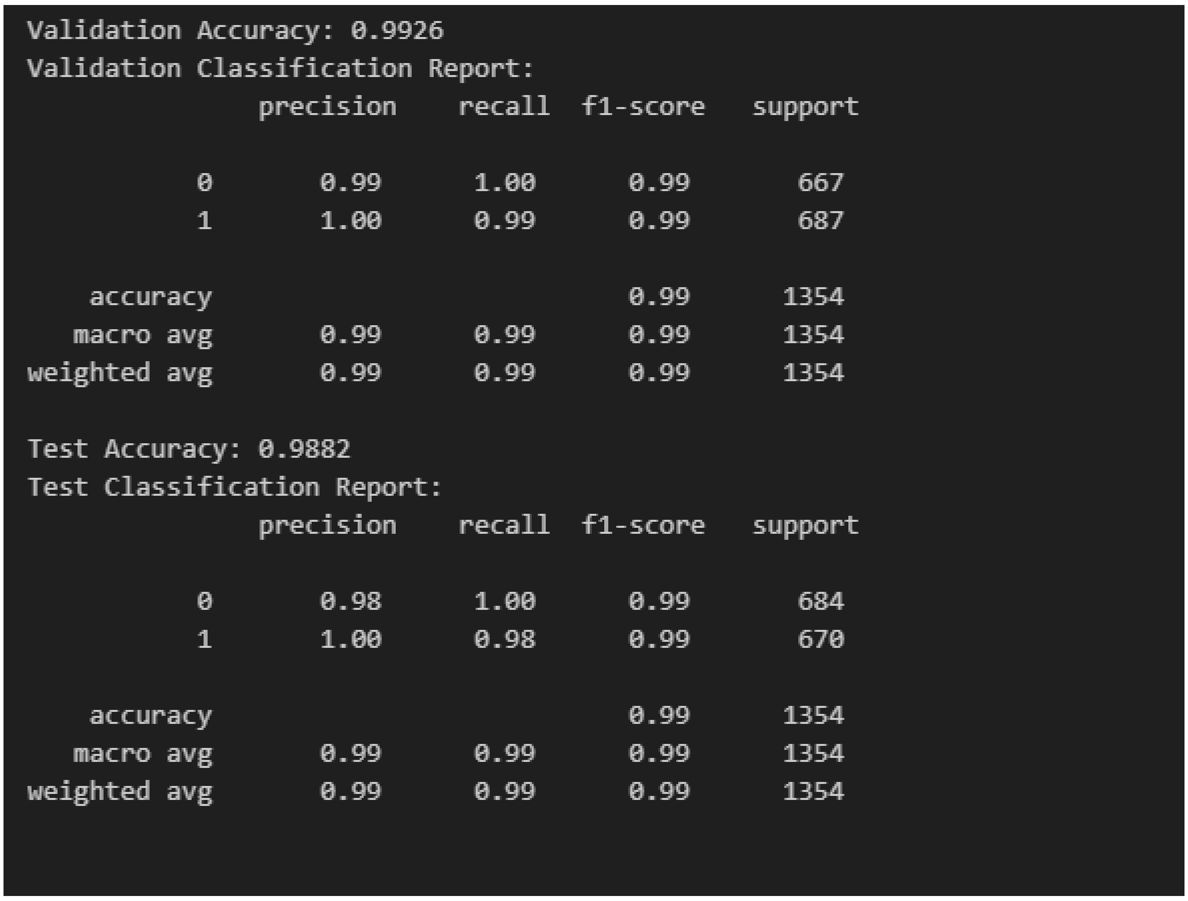
.

3.6.2 Results

In implementation MQ135 used lot of current. And also, it is connecting to the Arduino Nano. Arduino Nano can only provide 3.3 V. Because of that a level converter is used to convert 3.3V to 5.0 V.
MQ135 sensor was unstable when measuring. So, we tested so many levels manually and obtain value. But still the issue persists. MQ135 become unstable when measuring.
LIDAR was returned an error in the startup. We changed the USB configuration of Raspberry Pi module. After that the LIDAR issue fixed.

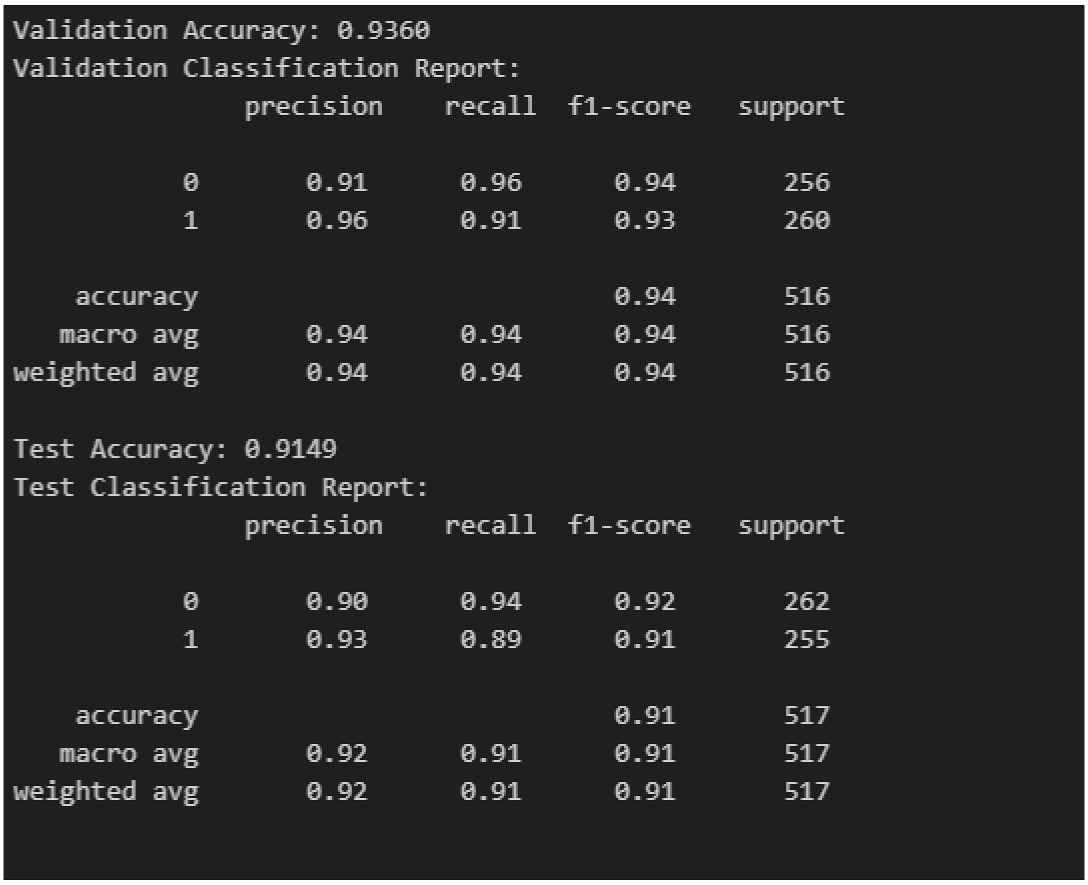
In initial implementation phase the GPS module failed to connect to a satellite. Later we found that it is because we are testing inside a house. GPS module only works when we are outside.

1. Event response model accuracy



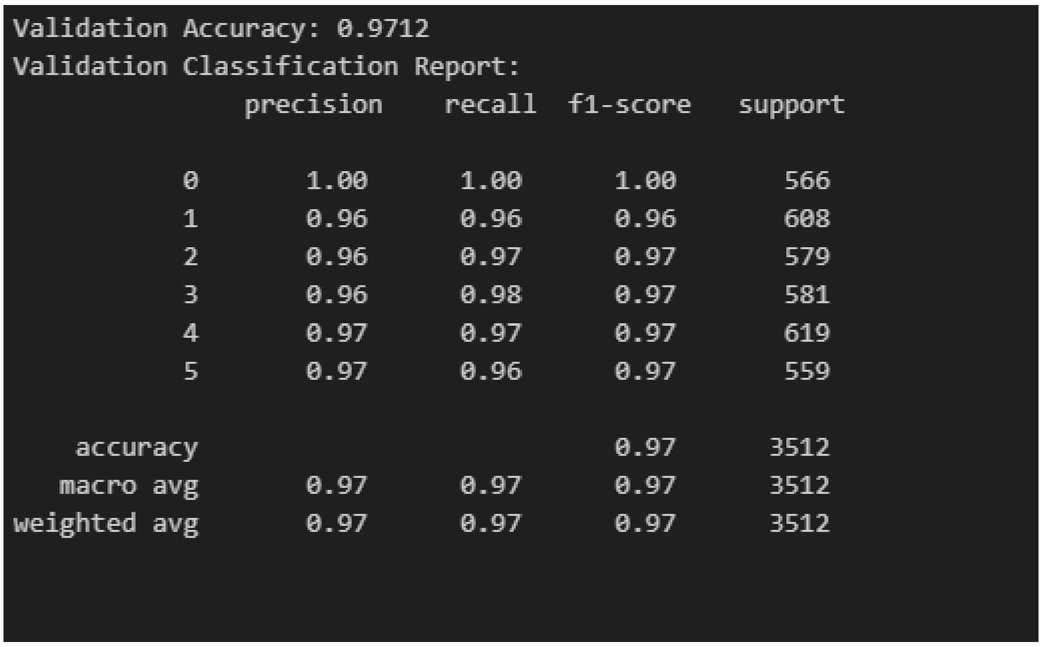
```
Validation Accuracy: 0.9926
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       667
           1       1.00      0.99      0.99       687

    accuracy                           0.99      1354
   macro avg       0.99      0.99      0.99      1354
weighted avg       0.99      0.99      0.99      1354

Test Accuracy: 0.9882
Test Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       684
           1       1.00      0.98      0.99       670

    accuracy                           0.99      1354
   macro avg       0.99      0.99      0.99      1354
weighted avg       0.99      0.99      0.99      1354
```

2. Maintenance prediction accuracy



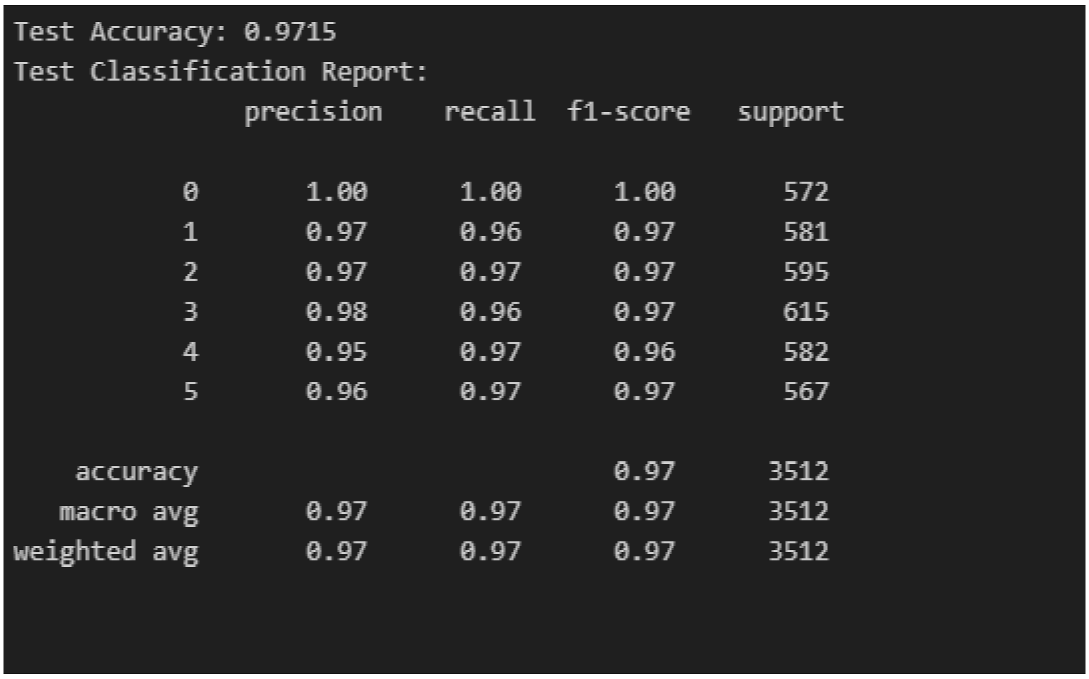
```
Validation Accuracy: 0.9360
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.94       256
           1       0.96      0.91      0.93       260

    accuracy                           0.94       516
   macro avg       0.94      0.94      0.94       516
weighted avg       0.94      0.94      0.94       516

Test Accuracy: 0.9149
Test Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       262
           1       0.93      0.89      0.91       255

    accuracy                           0.91       517
   macro avg       0.92      0.91      0.91       517
weighted avg       0.92      0.91      0.91       517
```

3. Robot assignment accuracy 1



```
Validation Accuracy: 0.9712
Validation Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       566
           1       0.96      0.96      0.96       608
           2       0.96      0.97      0.97       579
           3       0.96      0.98      0.97       581
           4       0.97      0.97      0.97       619
           5       0.97      0.96      0.97       559

    accuracy                           0.97      3512
   macro avg       0.97      0.97      0.97      3512
weighted avg       0.97      0.97      0.97      3512
```

4. Robot assignment accuracy 2



```
Test Accuracy: 0.9715
Test Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       572
           1       0.97      0.96      0.97       581
           2       0.97      0.97      0.97       595
           3       0.98      0.96      0.97       615
           4       0.95      0.97      0.96       582
           5       0.96      0.97      0.97       567

    accuracy                           0.97      3512
   macro avg       0.97      0.97      0.97      3512
weighted avg       0.97      0.97      0.97      3512
```

3.6.3 Discussion

Our final product is a successful night patrolling robot which satisfies customer’s needs. Ability to navigate in various environments with various conditions demonstrates the robot’s toughness. Integration of the sensors to detect the robot surroundings enhances the robot’s ability to roam around the warehouse autonomously without user interaction.

Because we automate the robot’s patrol process, the user interaction with the robot is less. It also reduces the cost and the need for the human personal to monitor the warehouse.

In high risk situations such as when fire or smoke detected it will show in the real time web dashboard.

After setting the main robot system with the slave robots, they will send valuable real-time insights to the firebase database. Which will be then accessible by web dashboard.

Trained maintenance prediction machine learning model gives an alert prior to a maintenance which will reduce the robot’s downtime a lot.

3.7 Conclusion

This robot system satisfies warehouse owners in every aspect. By reducing the cost, by reducing the human personal needed for monitor the warehouse etc. This system protects the warehouse from various hazards and instantly reporting them to the security personal in charge. This research shows the advancement of warehouse security patrol robots. In future all the staff in the warehouses will be replaced by the robots. This system is also reducing the human error caused when monitoring the warehouses. These types of robots increase the operational efficiency and cost efficiency.

.

4. Description of Personal and Facilities

Member	Liyanaarachchi D.L.B. S
Sub Objective Equip robot with the best communication protocol which reduces latency, response time and saves battery life. Develop a suitable algorithm which enables autonomous navigation and intelligent path planning and select the best path available according to the terrain, distance and battery life of the robots. Tasks, 1. Develop machine learning model to predict maintenance. 2. Develop machine learning model to make decisions using battery levels, robot coordinates etc. 3. Develop machine learning model to respond to an incident. 4. Built the hardware part for the robot.	

5. Budget and Budget Justification

Raspberry Pi 8 Gb	Rs.31,500.00
Arduino Nano	Rs.970.00
LIDAR sensor	Rs.19500.00
MQ-135 sensor	Rs.490.00
Pressure sensor	Rs.160.00
Humidity sensor (DHT-11)	Rs.390.00
GPS module	Rs.900.00
Level shifter	Rs.150.00
ADC converter	Rs.310.00

References

[1] Y. Collet, “LZ4: Extremely fast compression algorithm,” GitHub Repository, 2011. [Online]. Available: https://github.com/lz4/lz4

[2] F. Drost, “Zstandard: A stronger compression algorithm,” Facebook Engineering Blog, 2015. [Online]. Available: https://engineering.fb.com/data/zstandard/

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013, doi: 10.1016/j.future.2013.01.010.

[4] D. Mendez, I. Papapanagiotou, and B. Yang, “Internet of Things: Survey on security and privacy,” *IET Netw.*, vol. 6, no. 5, pp. 409–422, Sep. 2017, doi: 10.1049/iet-net.2017.0017.

[5] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.
[Online]. Available: https://doi.org/10.1145/2939672.2939785

[6] F. Mehmood, F. Ahmad and M. A. Jan, “A Machine Learning-Based Air Quality Prediction Model Using Sensors Data,” *Sensors*, vol. 21, no. 9, pp. 1–21, 2021. [Online]. Available: https://doi.org/10.3390/s21093025

[7] D. Patel, S. Patel and H. Bhavsar, “Comparative Analysis of Machine Learning Algorithms for Air Quality Prediction,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 5, no. 5, pp. 48–54, 2020.
[Online]. Available: https://ijsrcse.com/IJSRCSE_vol5_issue5/paper_9.pdf

[8] J. Saini, M. Dutta and G. Marques, “Indoor Air Quality Monitoring Systems Based on Internet of Things: A Systematic Review,” *Int. J. Environ. Res. Public Health*, vol. 17, no. 11, p. 3946, 2020.
[Online]. Available: https://doi.org/10.3390/ijerph17113946

[9] Y. Zhang, L. Ding and L. Yang, “AQI Prediction Using a Hybrid Model Based on CNN and BiLSTM,” *IEEE Access*, vol. 7, pp. 150161–150171, 2019.

[Online]. Available: https://doi.org/10.1109/ACCESS.2019.2947665

[1] Prof. M. K. Tiwari, 2Pooja Sahebrao Suryawanshi, 3Kulasum bi Kalim Sheikh, 4Shankar Bhagwan Teli, 5Jayshari Raju Bairagi | Patrolling Robot | DOI : https://doi.org/10.62226/ijarst20241385

[2] Suniksha B S1 ,Shubhanchal Priya2 , Varshini M3 , Deepthi Raj4 "Warfield Spy Robot with Night Vision Wireless Camera", International Journals of Latest Technology in Engineering, Managements& Applied Sciences (IJLTEMAS) Volume IX, Issues VII, July 2020

[3] Department of Ece, K Stella. (2023). NIGHT PATROLLING ROBOT. European Chemical Bulletin. 12. 3164-3169. 10.31838/ecb/2023.12.s1- B.318.

[4] Sri Dinesh Kumar P, Pragadishwaran T.R,"Implementation Of Spy Robot For A Surveillance System Using Internetprotocol", International Journal for Research Trends and Innovation (IJRTI), Volume 8, Issue 2, 2023.

[5] B. Rithika, M. Yedukondalu, "IoT Enabling Night Patrolling Robot For Womensafety", Journal of Emerging Technologies and Innovative Research (JETIR), Volume 10, Issue 5, 2023

[6] Vivek Upadhyay, Vaibhav Singh, "Roboseciot Based Patrolling Robot", International Journal of Advances in Engineering and Management (IJAEM), Volume 5, Issue 4 April 2023.

[7] Anju Babu, Anna Mariya E S, "Sound Triggered Patrolling And Surveillance Robot Using Deep Learning", International Journal of Engineering Research & Technology (IJERT), ICCIDT – 2023 Conference Proceedings, Special Issue – 2023.

[8] Farabee Khalid, “Night Patrolling Robot”, 2nd International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST), IEEE, 2021

[9] P.Anbumani, K.Feloomi, “Iot Based Smart Night Patrolling Robot”, International Journal of Creative Research Thoughts (IJCRT), Volume 11, Issue 4 April 2023.