Optimizing Warehouse Outdoor Security with Autonomous Patrol Robots

24-25J-053

Final Individual Report

D.L.B.S Liyanaarachchi
IT21210860

Bachelor of Science (Honors) in Information Technology Specialized Computer Systems and Network Engineering

Department of Computer Systems Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

Declaration

I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Name	Student ID	Signature
D.L.B.S Liyanaarachchi	IT21210860	

The supervisor/s should certify the proposal report with the following declaration.

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor: Date

Abstract

The advancement in robotics, artificial intelligence and machine learning has enabled to develop autonomous patrol robots with advanced features. This paper introduces an outdoor warehouse patrol robot designed to monitor outdoor and indoor environments of warehouses without human interference. This robot is equipped with lots of sensors which help the robot to sense the environment and get a deeper idea about its surroundings. Robot is also equipped with advanced detection algorithms, decision making algorithms, event response models etc. This robot not only reduces the human workload that is needed for monitoring the environment but also it ensures 24/7 surveillance with cost effectiveness and improved efficiency.

TABLE OF CONTENTS

Table of Contents

LIST OF APPENDICES

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

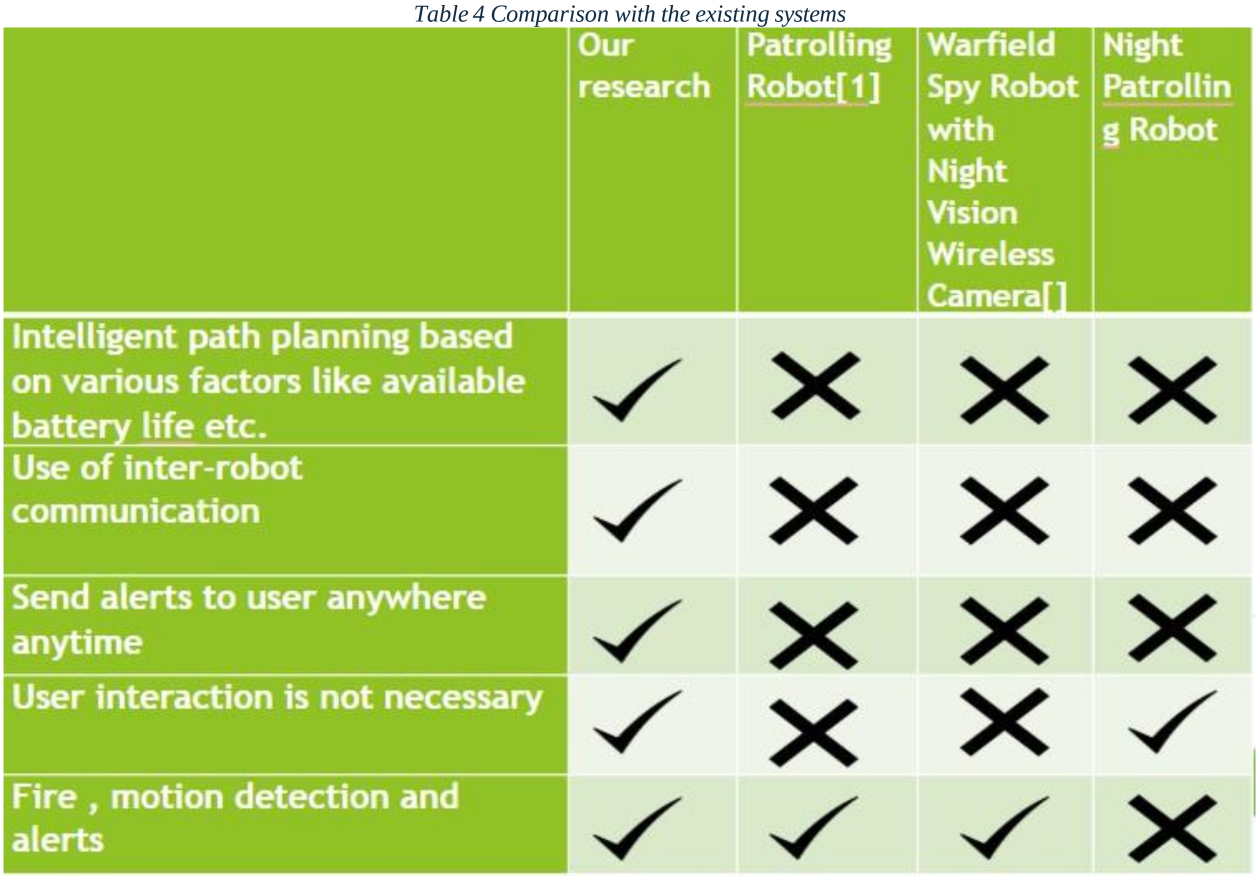
1.1.1 Indoor Positioning System Based on Wi-Fi

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.

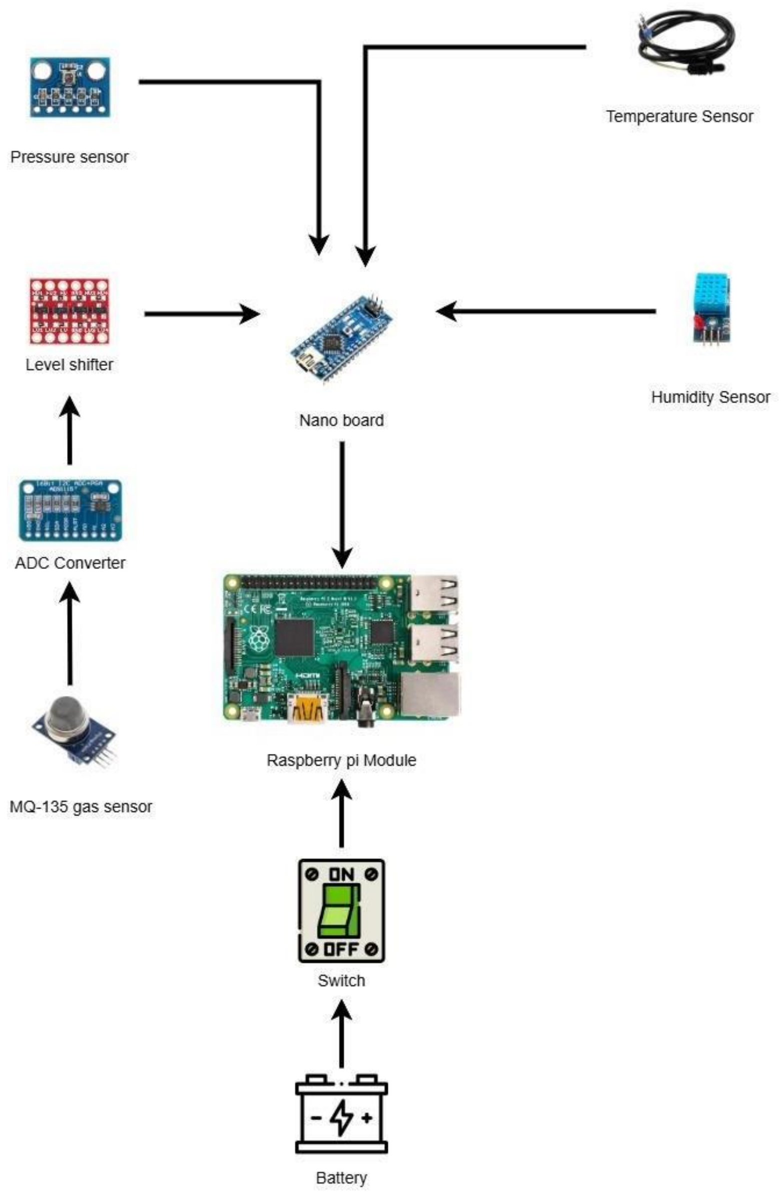
2.2 Specific Objectives

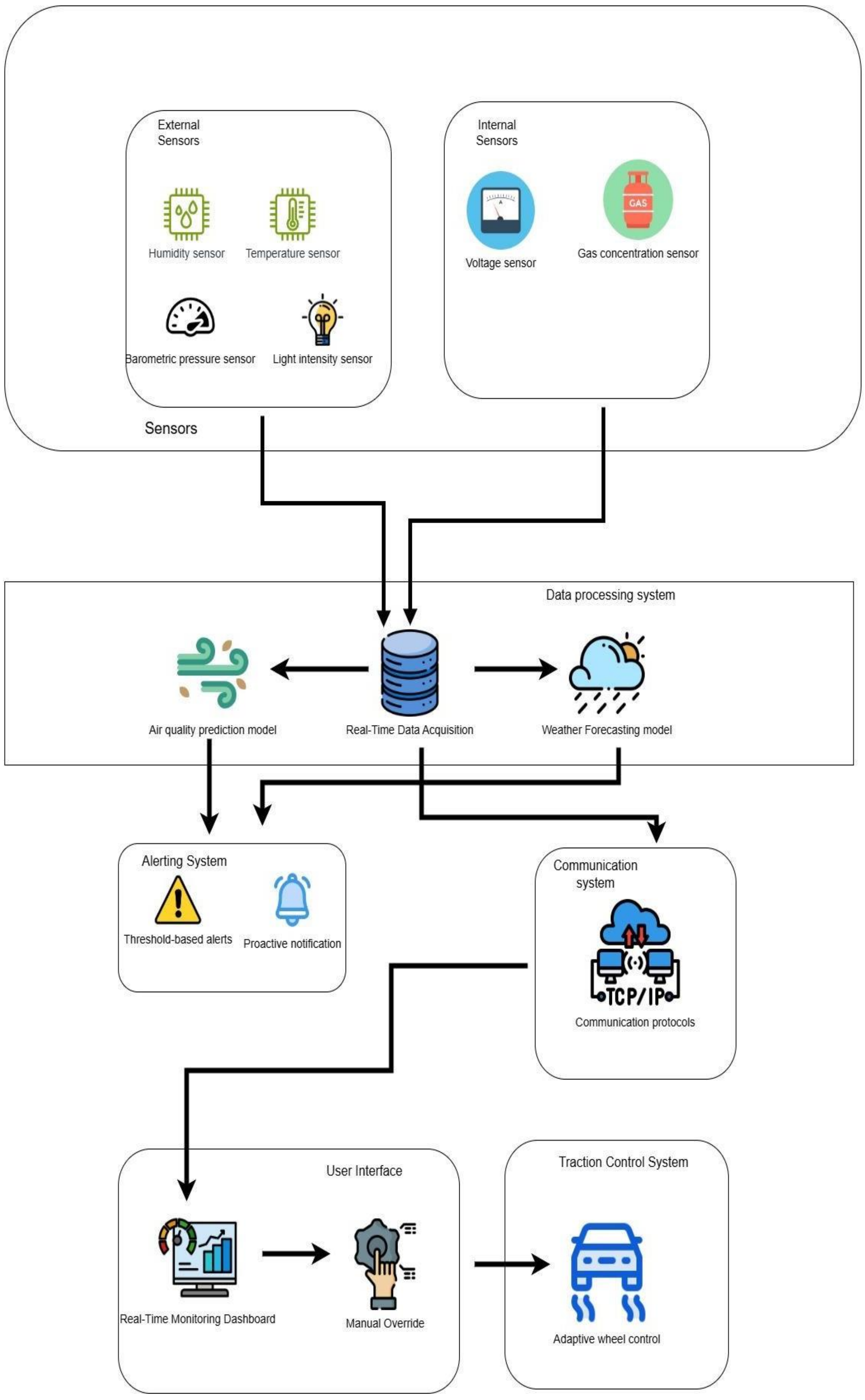
- Equip robot with the best communication protocol which reduces latency, response time and saves battery life.
- Identify environmental temperature, humidity levels etc.
- Obtain real-time status of sensors of the slave robot using the communication protocol implemented.
- Develop a suitable algorithm 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
GAS
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.

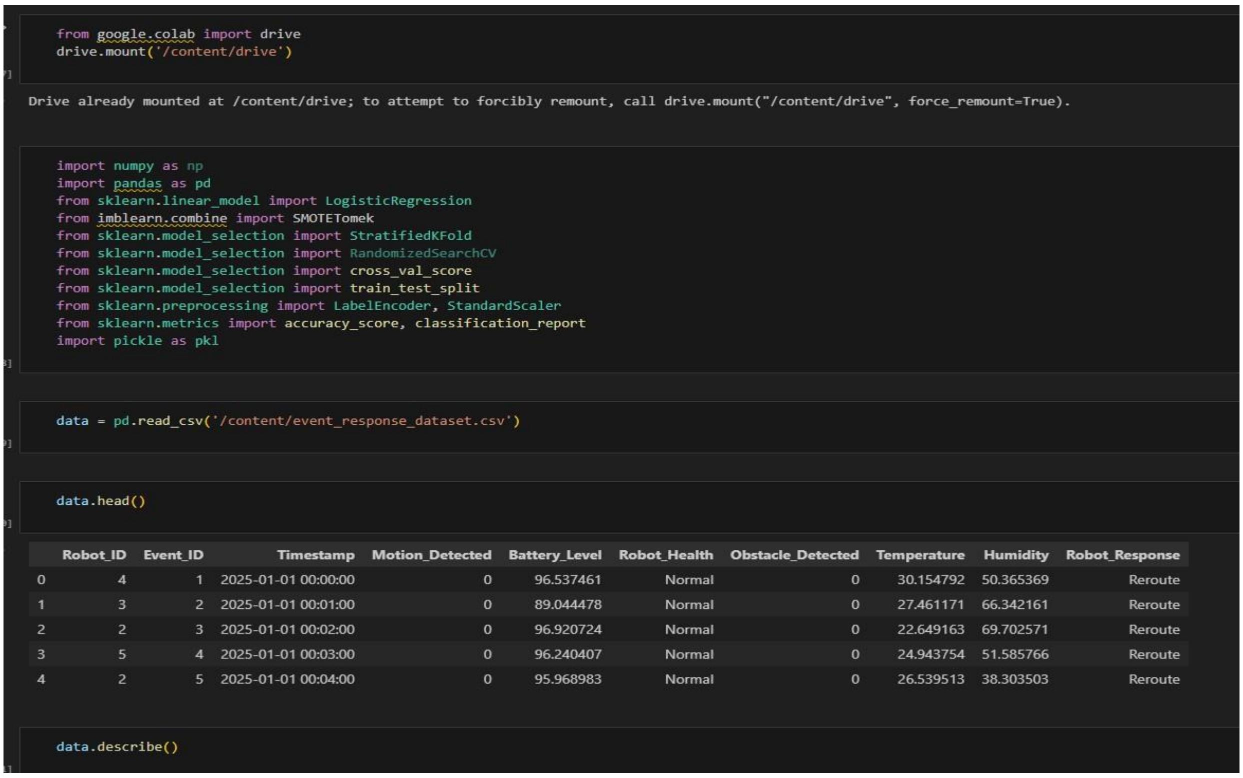
All the sensors are connected to Arduino Nano board. This Arduino Nano board receives the power from raspberry pi board directly via USB-A interface. All the sensors are connected directly to the Arduino Nano board except MQ-135 gas detection sensor. Because of this MQ-135 uses 5V , we used level shifter 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).
```

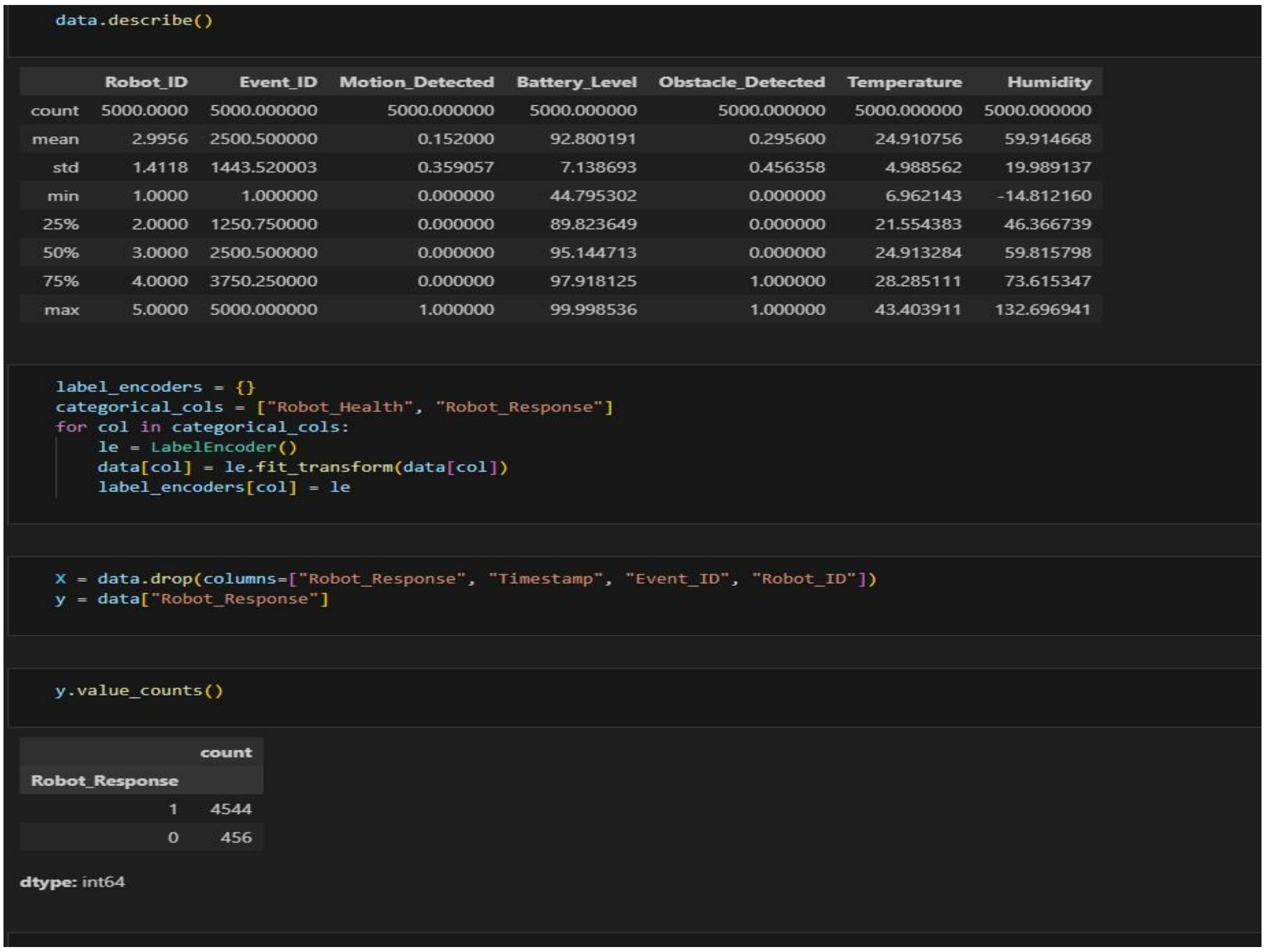
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from imblearn.combine import SMOTETomek
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection 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()
```



```
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

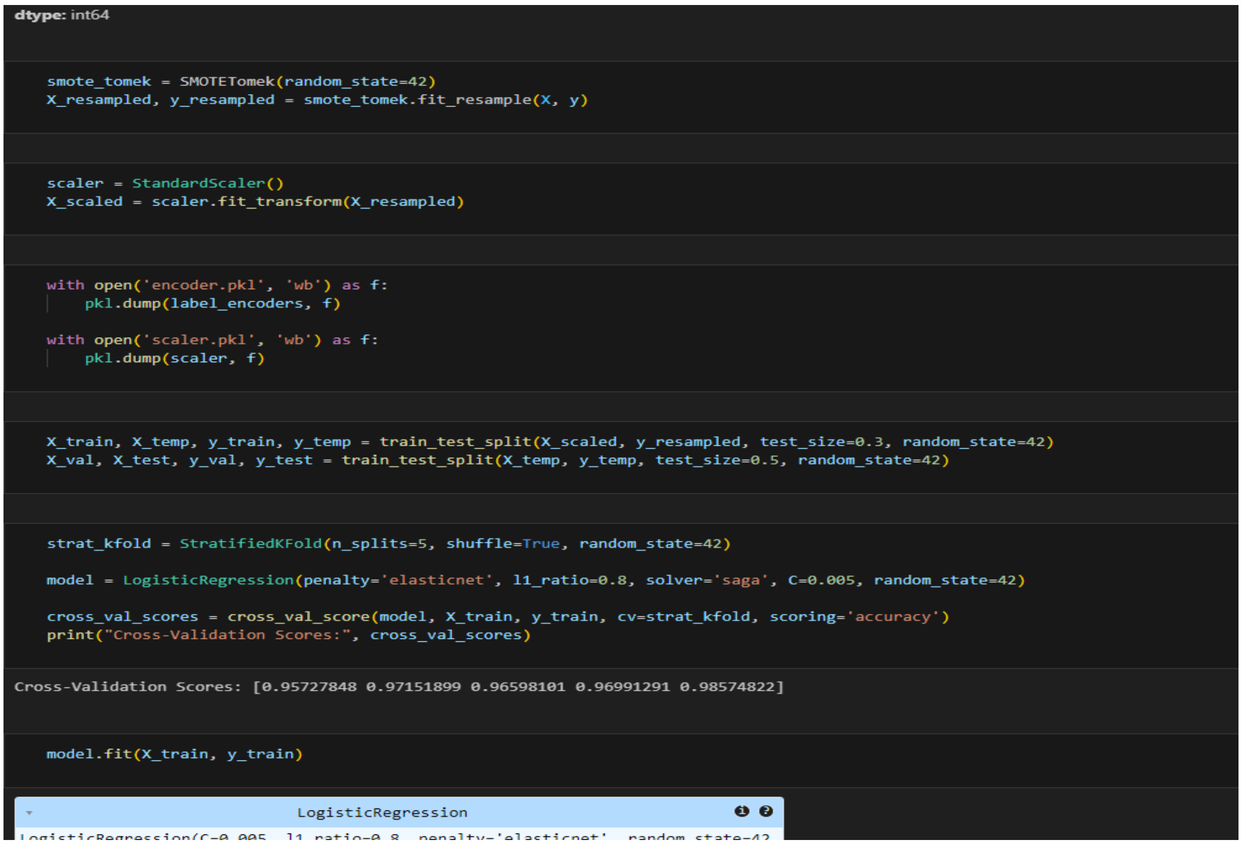
```
label_encoders = {}
categorical_cols = ["Robot_Health", "Robot_Response"]
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```

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

```
y.value_counts()
```

	count
Robot_Response	
1	4544
0	456

dtype: int64



dtype: int64

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

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

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

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

```
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)
```

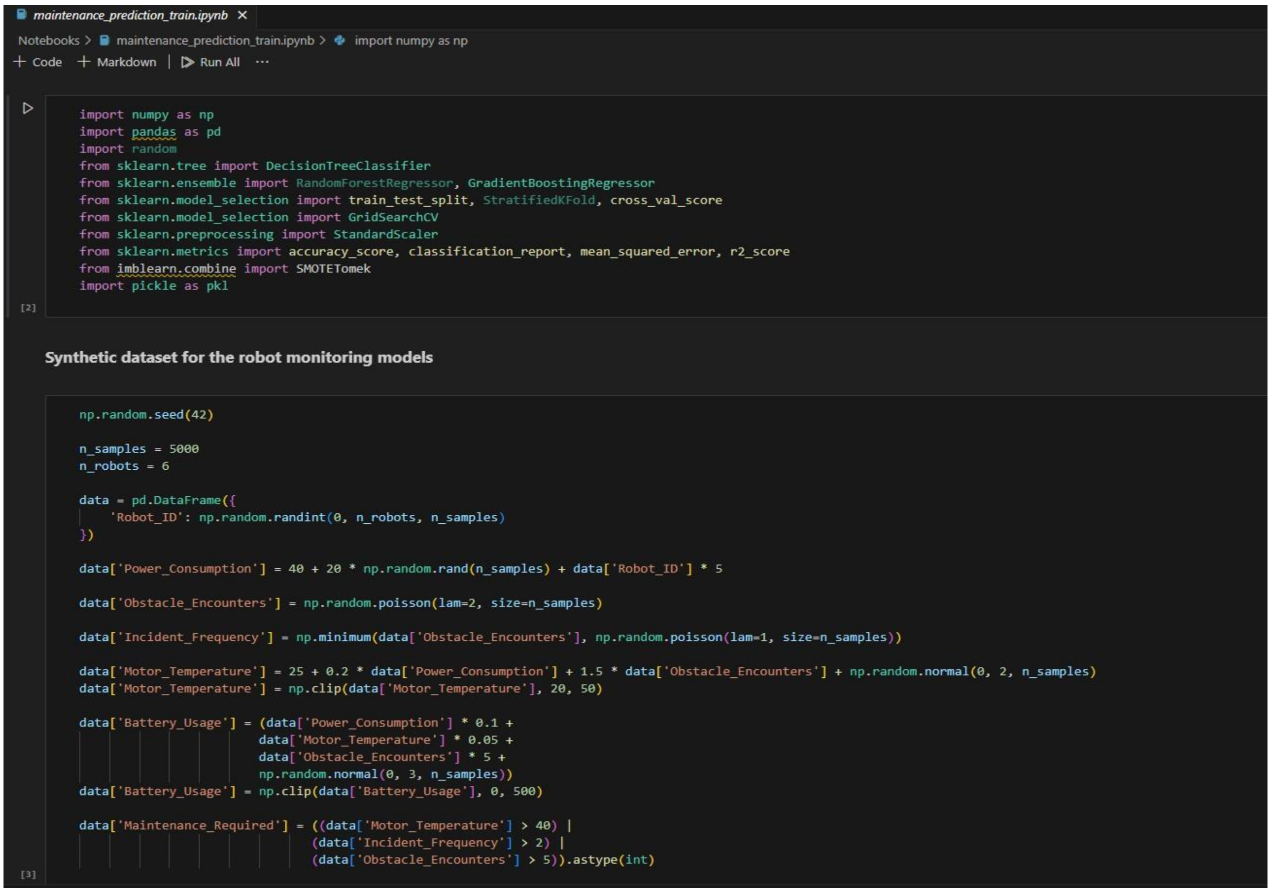
```
strat_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

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]
```

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

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

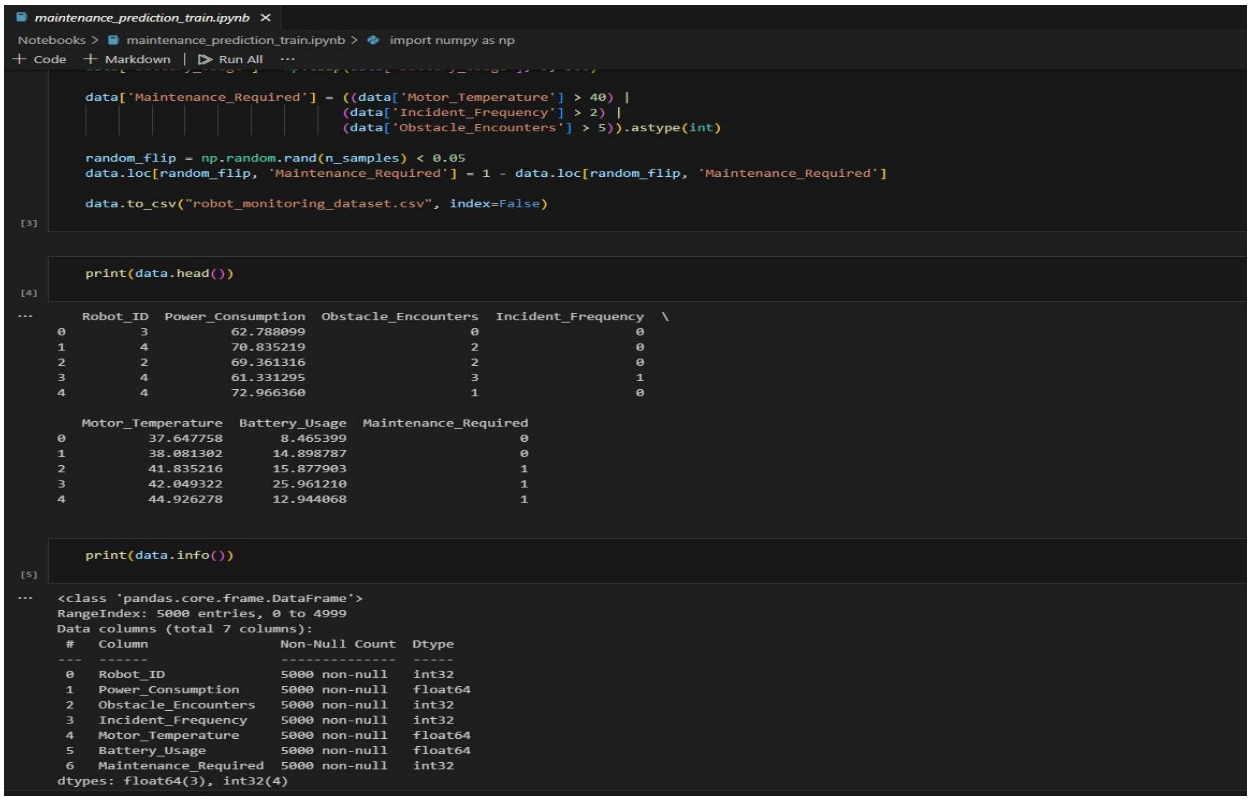
2. Maintenance prediction model development


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

```
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
```
[2]

Synthetic dataset for the robot monitoring models

```
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)
```
[3]


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

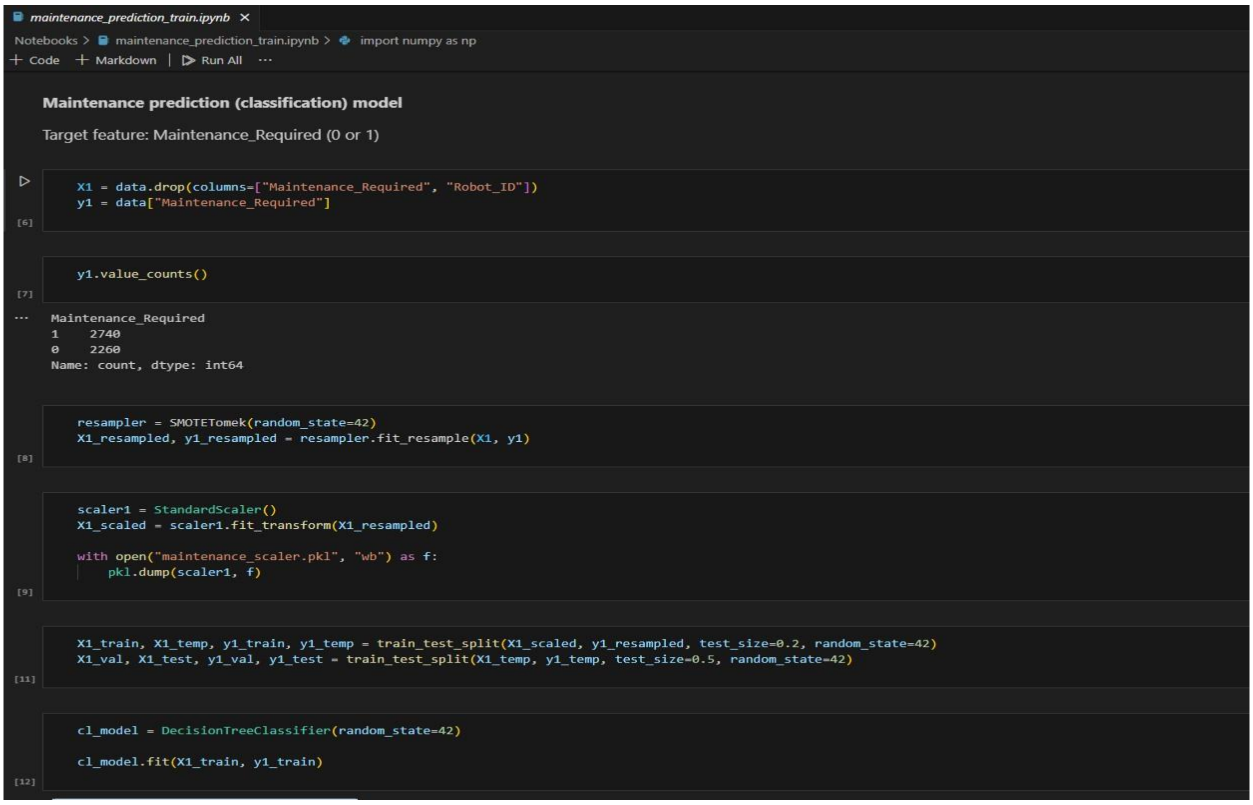
```
data['Maintenance_Required'] = ((data['Motor_Temperature'] > 40) |
                               (data['Incident_Frequency'] > 2) |
                               (data['Obstacle_Encounters'] > 5)).astype(int)

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)
```
[3]

```
print(data.head())
```
[4]

```
   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
```

```
print(data.info())
```
[5]

```
<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)

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

```
y1.value_counts()
```

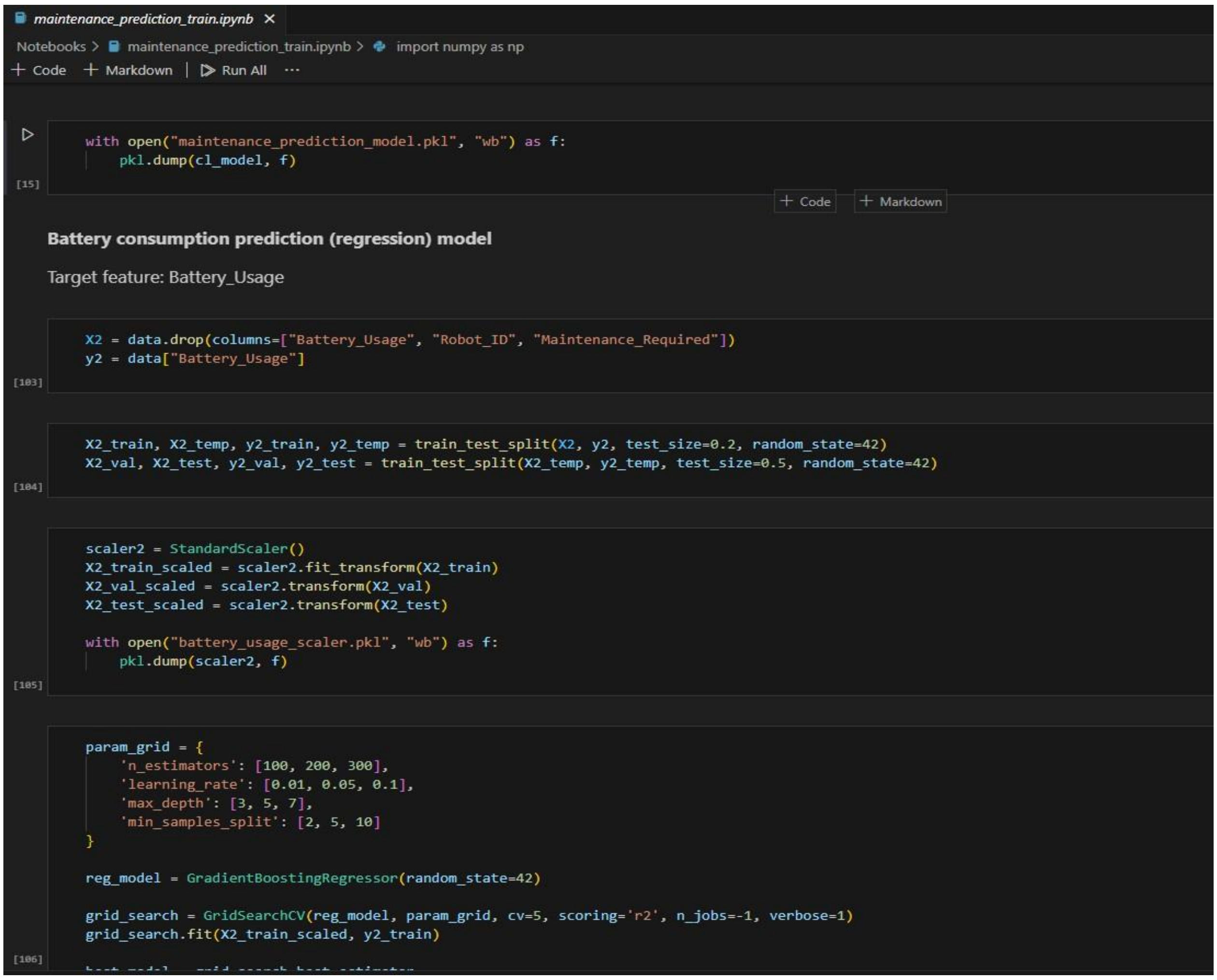
```
Maintenance_Required
1    2740
0    2260
Name: count, dtype: int64
```

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

```
scaler1 = StandardScaler()
X1_scaled = scaler1.fit_transform(X1_resampled)

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

```
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)
```

```
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 ···
+ Code + Markdown

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

Battery consumption prediction (regression) model

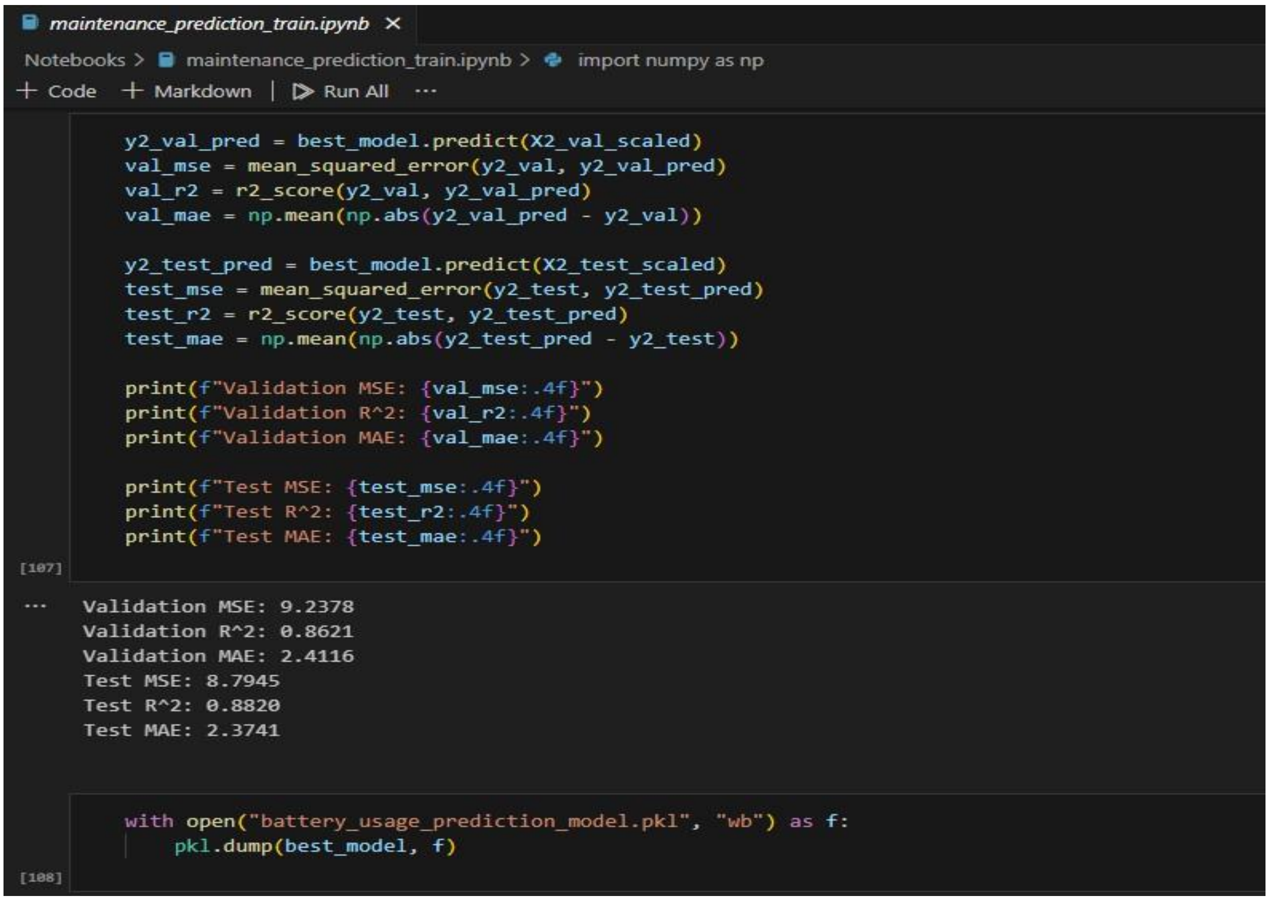
Target feature: Battery_Usage

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

```
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)
```

```
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)
```

```
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)
```

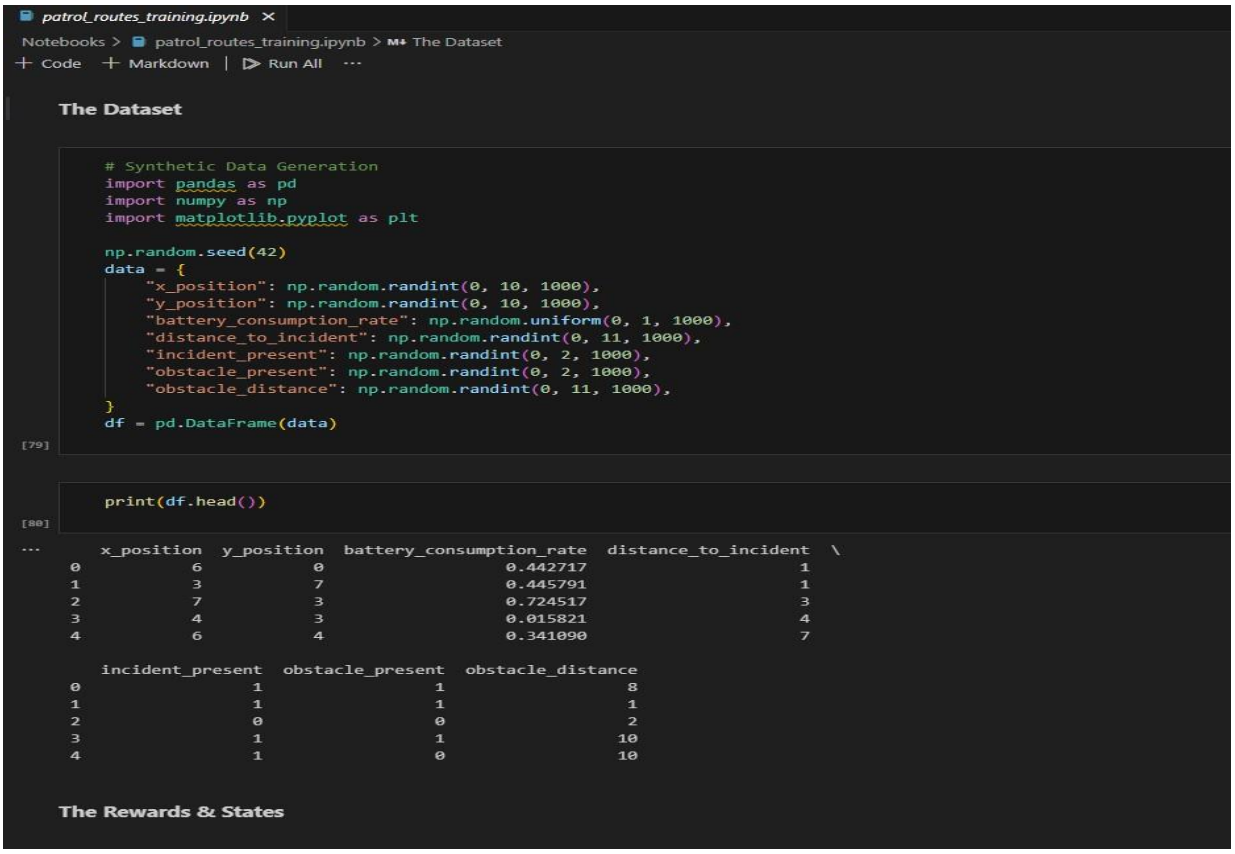

maintenance_prediction_train.ipynb
Notebooks > maintenance_prediction_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
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
```

```
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 ···

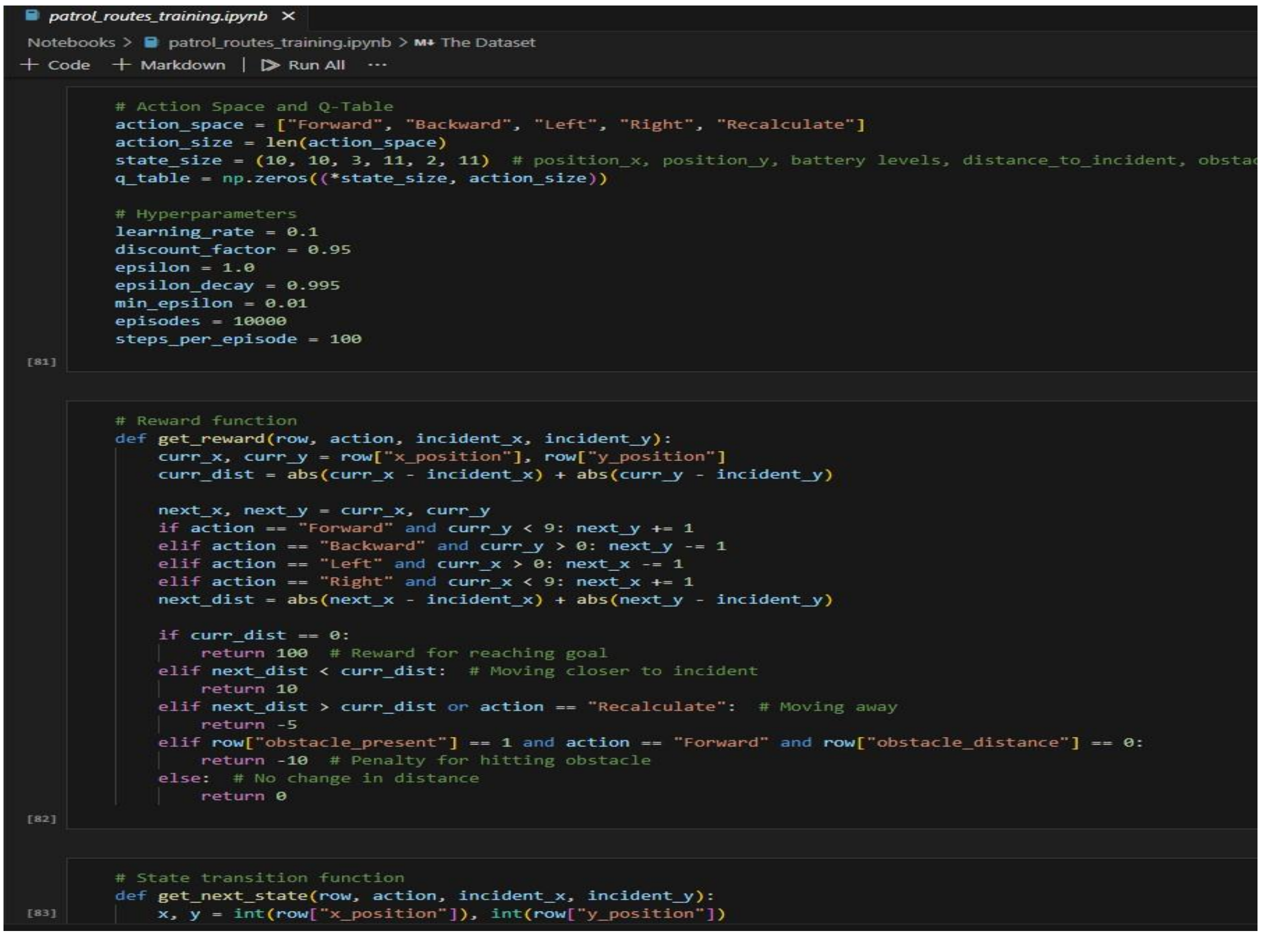
The Dataset

```
# 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]

```
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

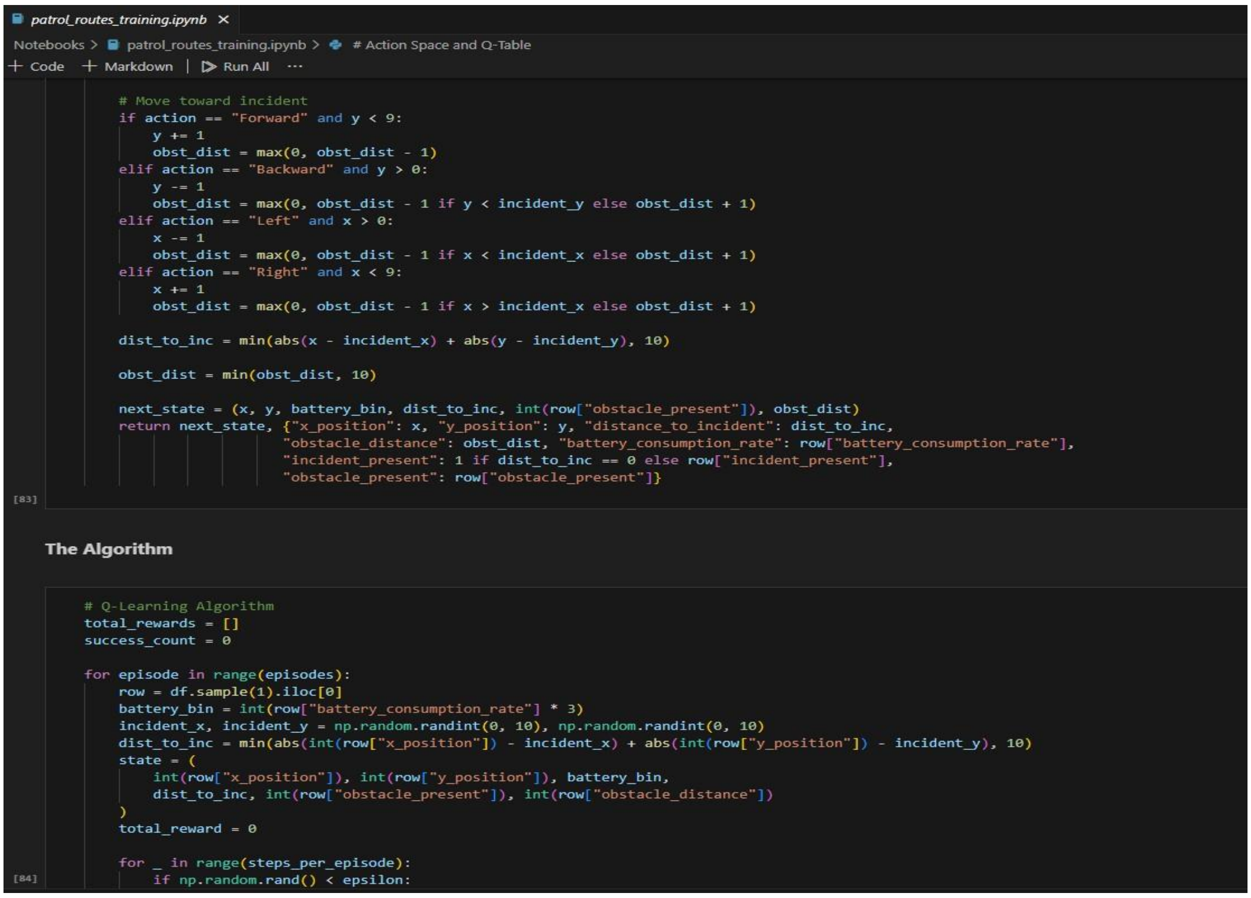


patrol_routes_training.ipynb
Notebooks > patrol_routes_training.ipynb > M↓ The Dataset
+ Code + Markdown | Run All ···

```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]

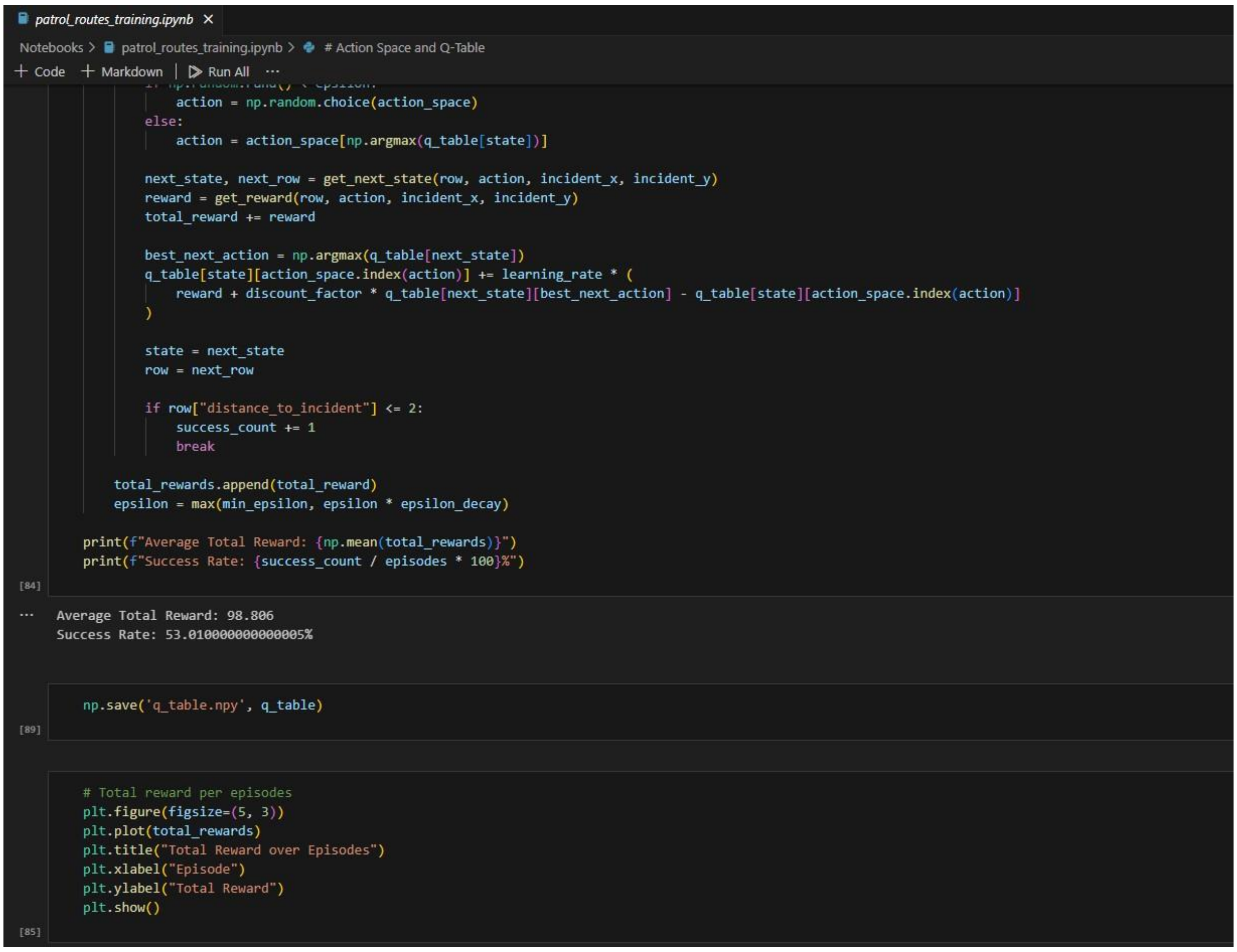


patrol_routes_training.ipynb
Notebooks > patrol_routes_training.ipynb > # Action Space and Q-Table
+ Code + Markdown | Run All ···

```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

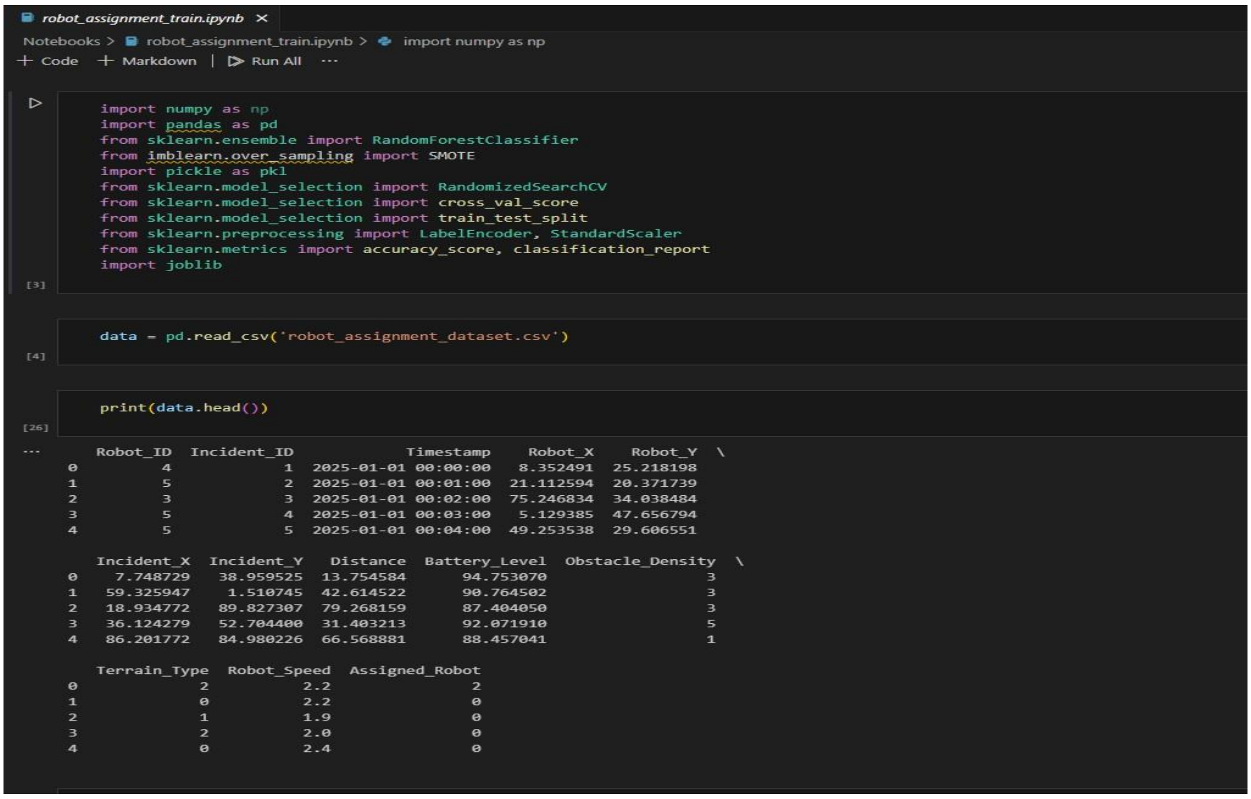
```
                action = np.random.choice(action_space)
            else:
                action = action_space[np.argmax(q_table[state])]

            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}%")
[84]
```

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

```
np.save('q_table.npy', q_table)
[89]
```

```
# 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()
[85]
```

4. Robot assignment training model

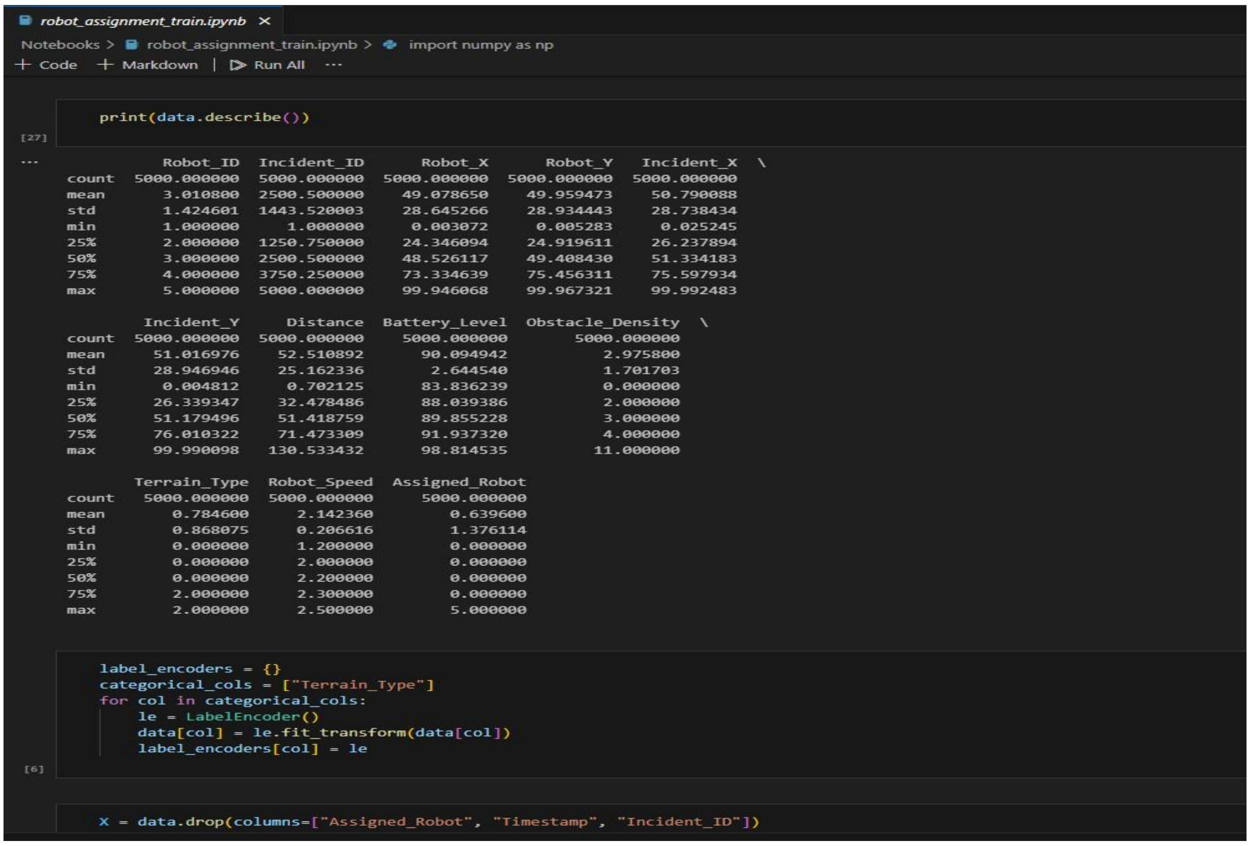

robot_assignment_train.ipynb
Notebooks > robot_assignment_train.ipynb > import numpy as np
Code
Markdown
Run All

```
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
[3]
```

```
data = pd.read_csv('robot_assignment_dataset.csv')
[4]
```

```
print(data.head())
[26]
```

```
   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
```

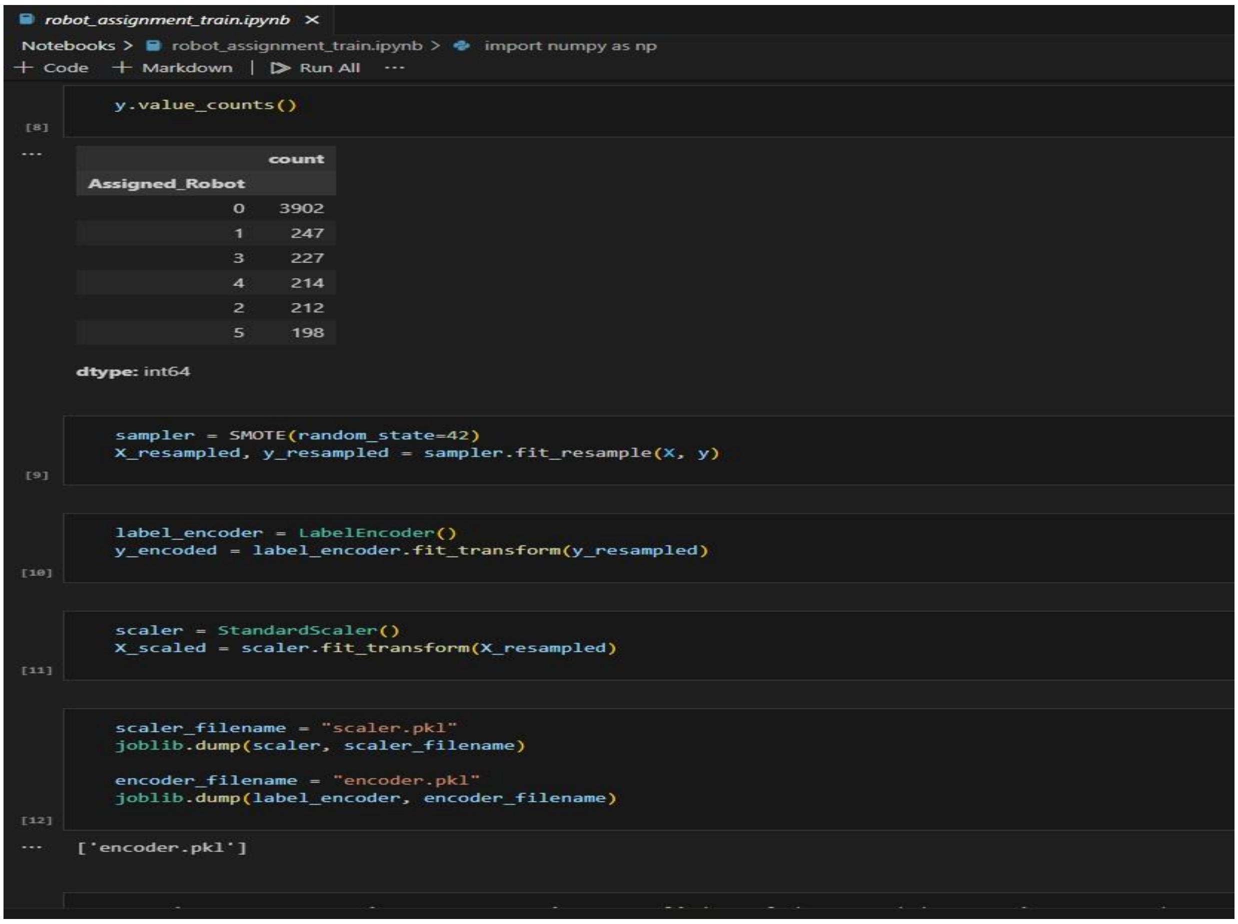

robot_assignment_train.ipynb ×
Notebooks > robot_assignment_train.ipynb > import numpy as np
+ Code + Markdown | Run All ···

```
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]

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

dtype: int64
```

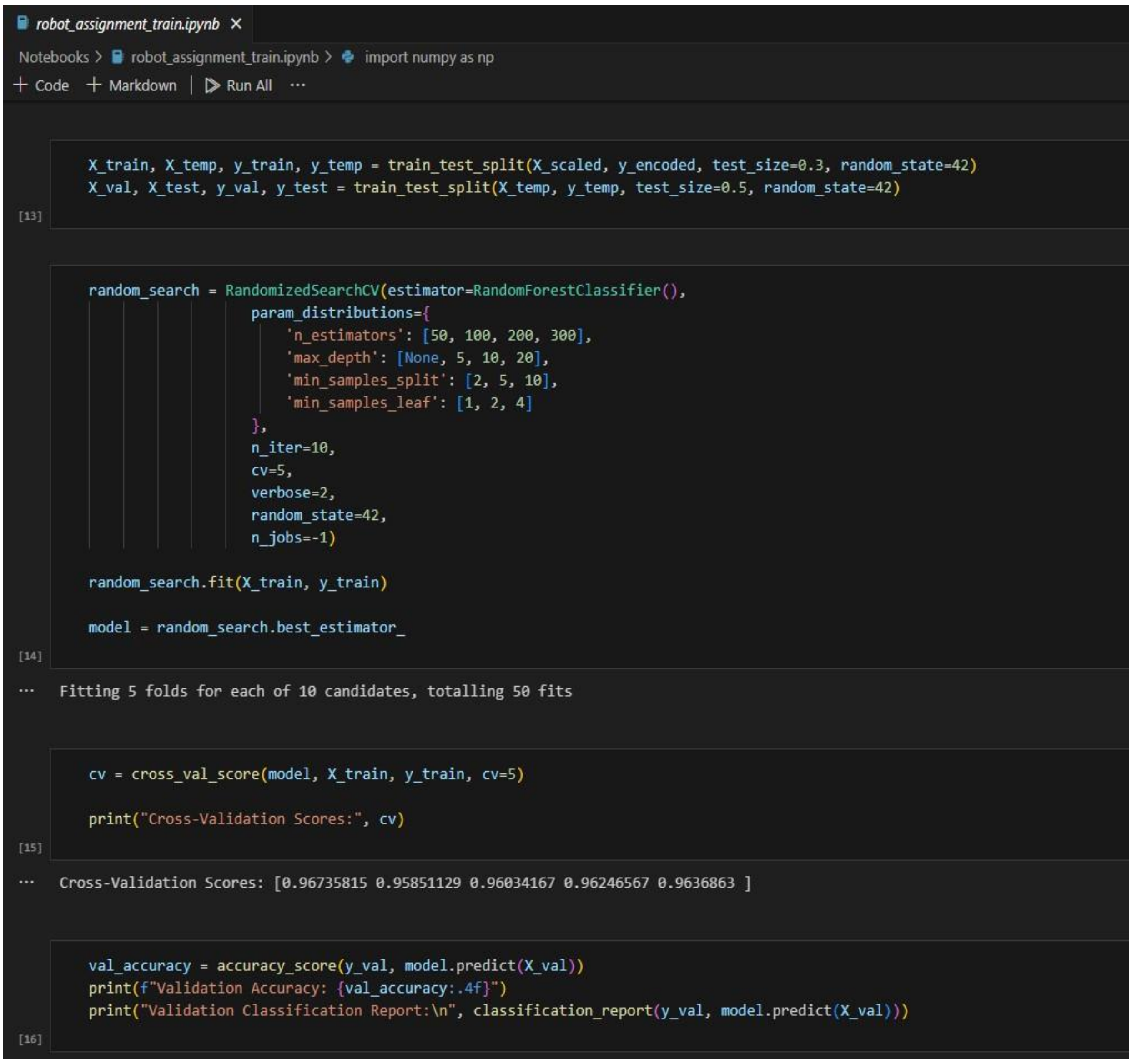
```
sampler = SMOTE(random_state=42)
X_resampled, y_resampled = sampler.fit_resample(X, y)
```
[9]

```
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.

.

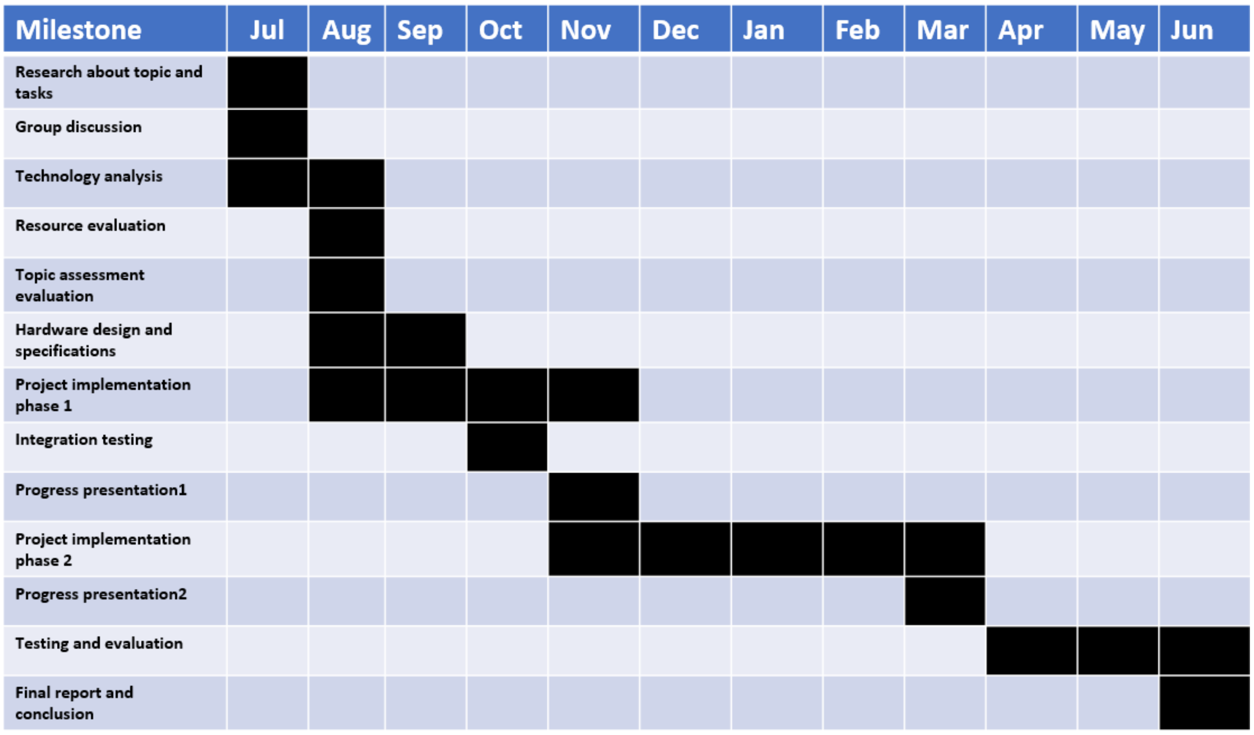
3.4 Project requirements

- 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.

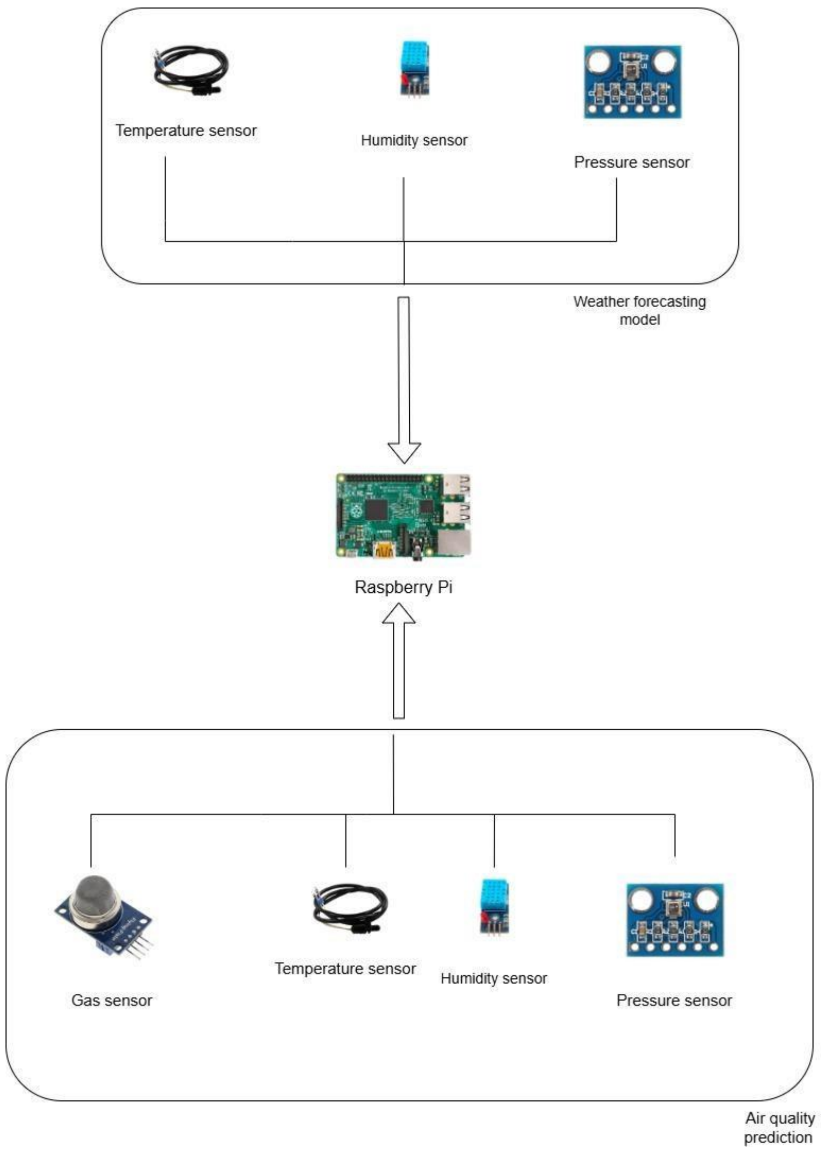
3.5 Timeline



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

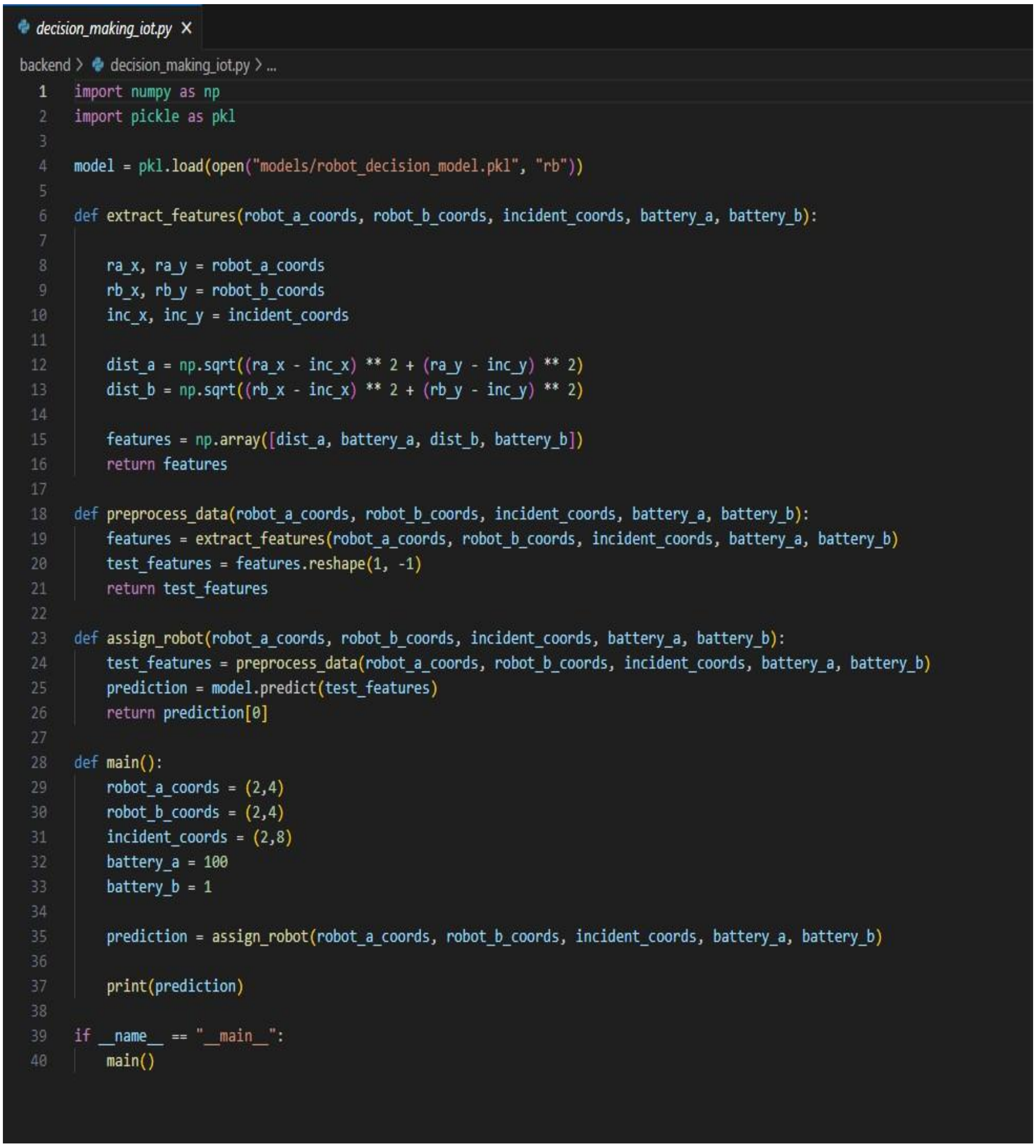
3.6.1 Research Findings



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()
```

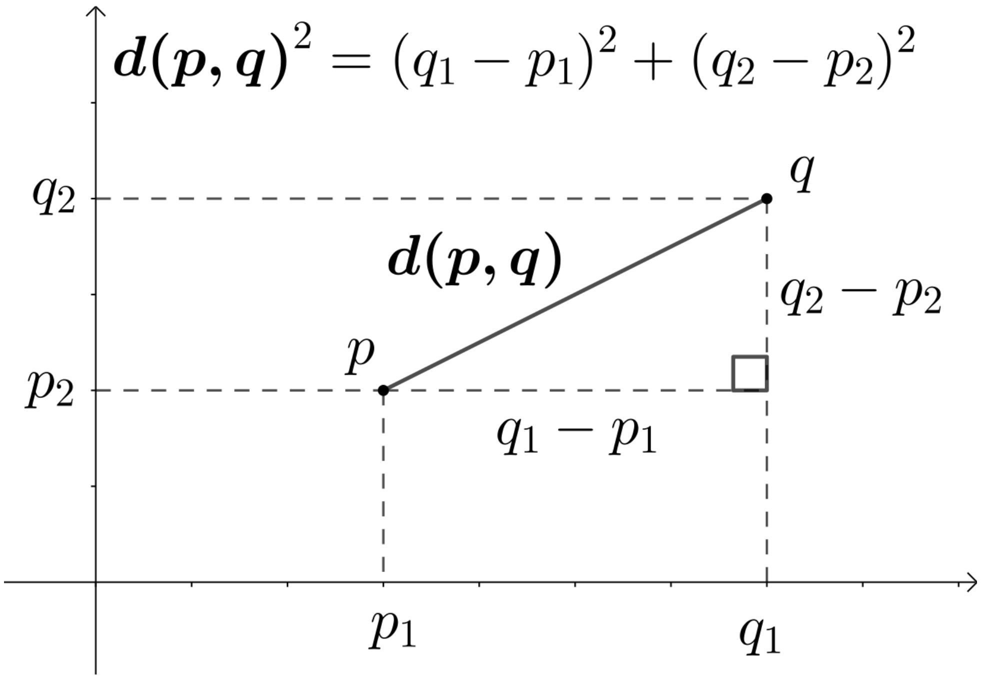
Above model is responsible for determine which robot will be chosen for go to the incident location. This model works by loading a pre trained machine learning model. This model uses pickle library.
Feature array includes,

- 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…

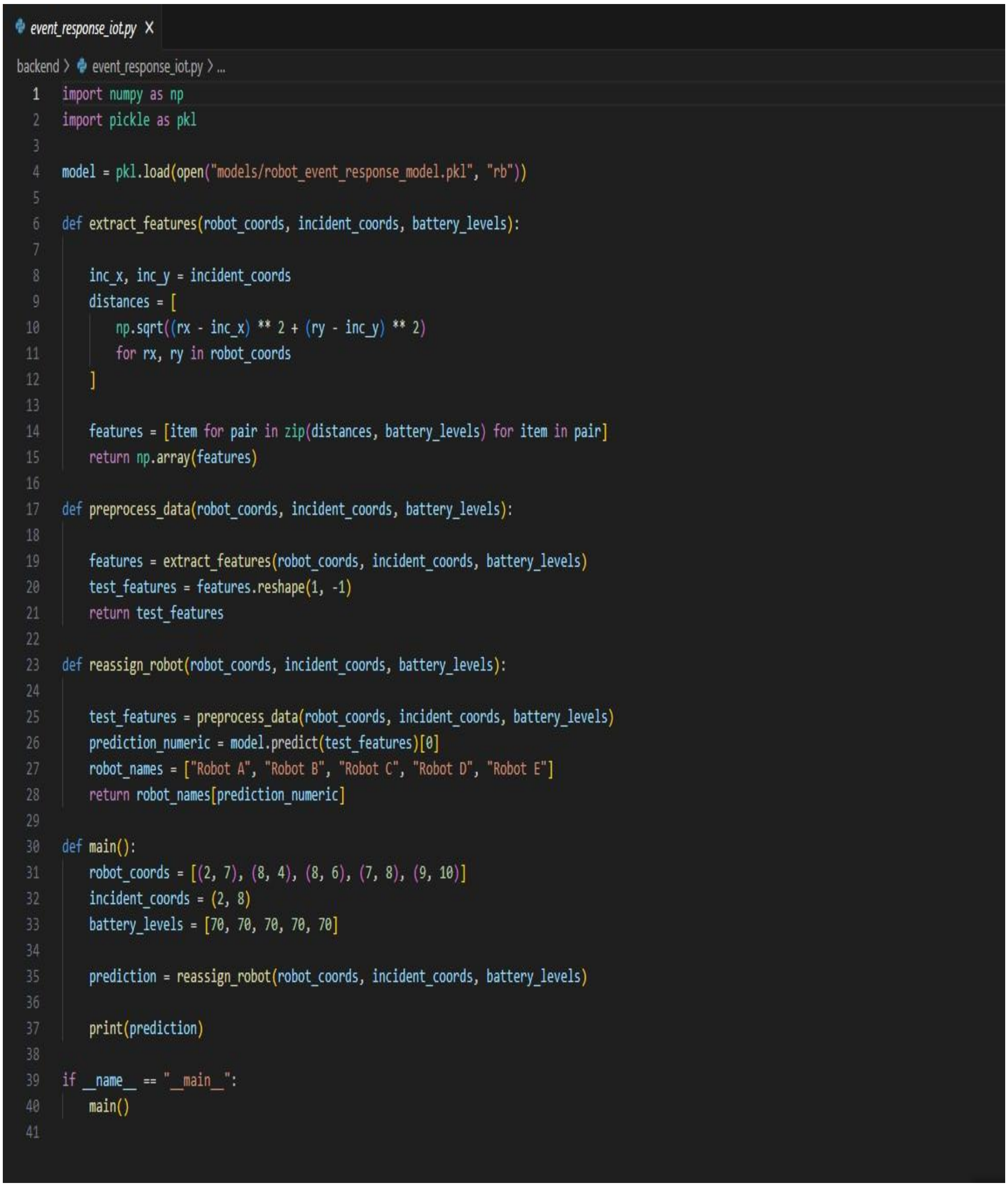
We can use the Pythagorean theorem to compute the Euclidian distance.



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

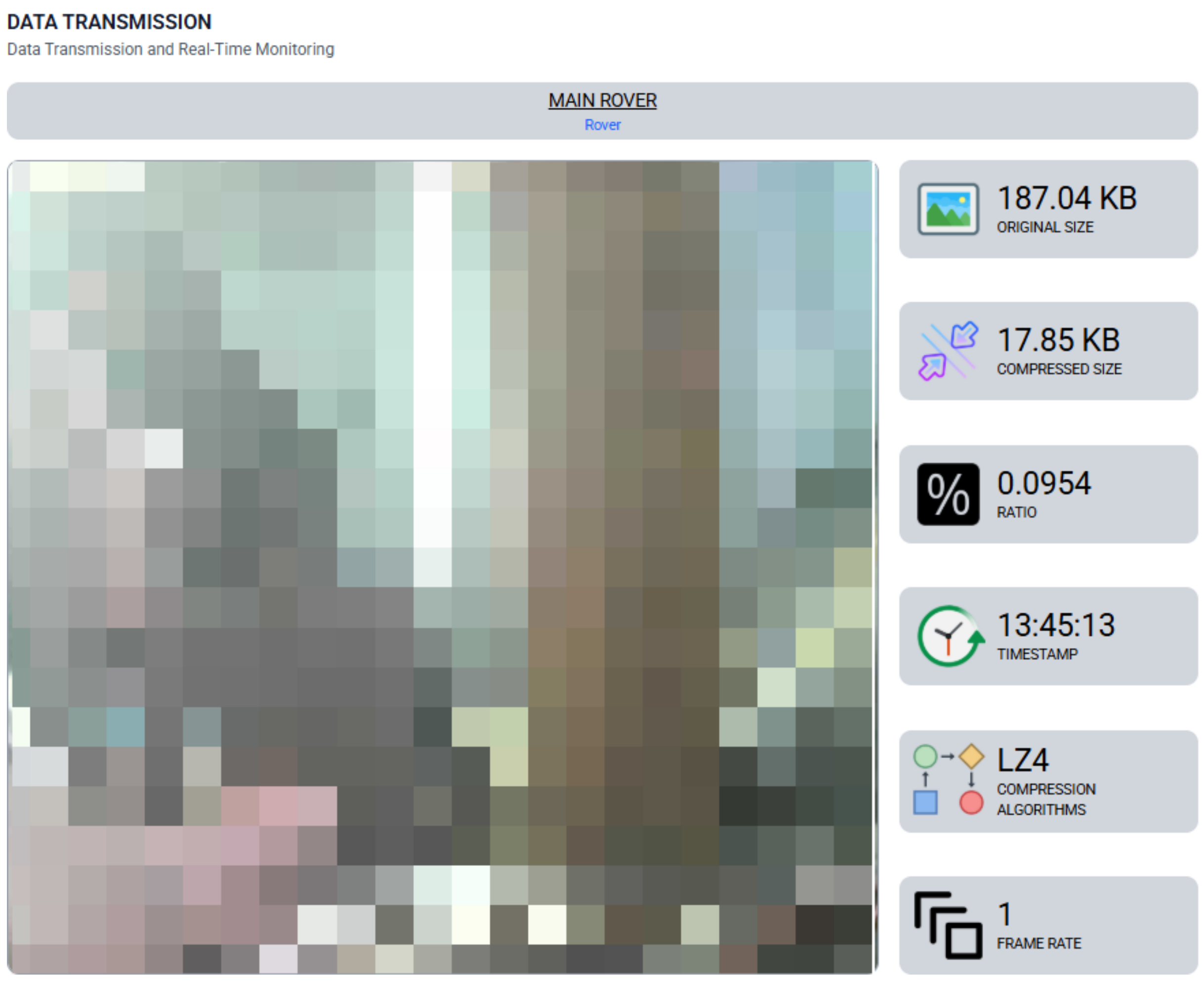
backend > event_response_iot.py > ...

```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

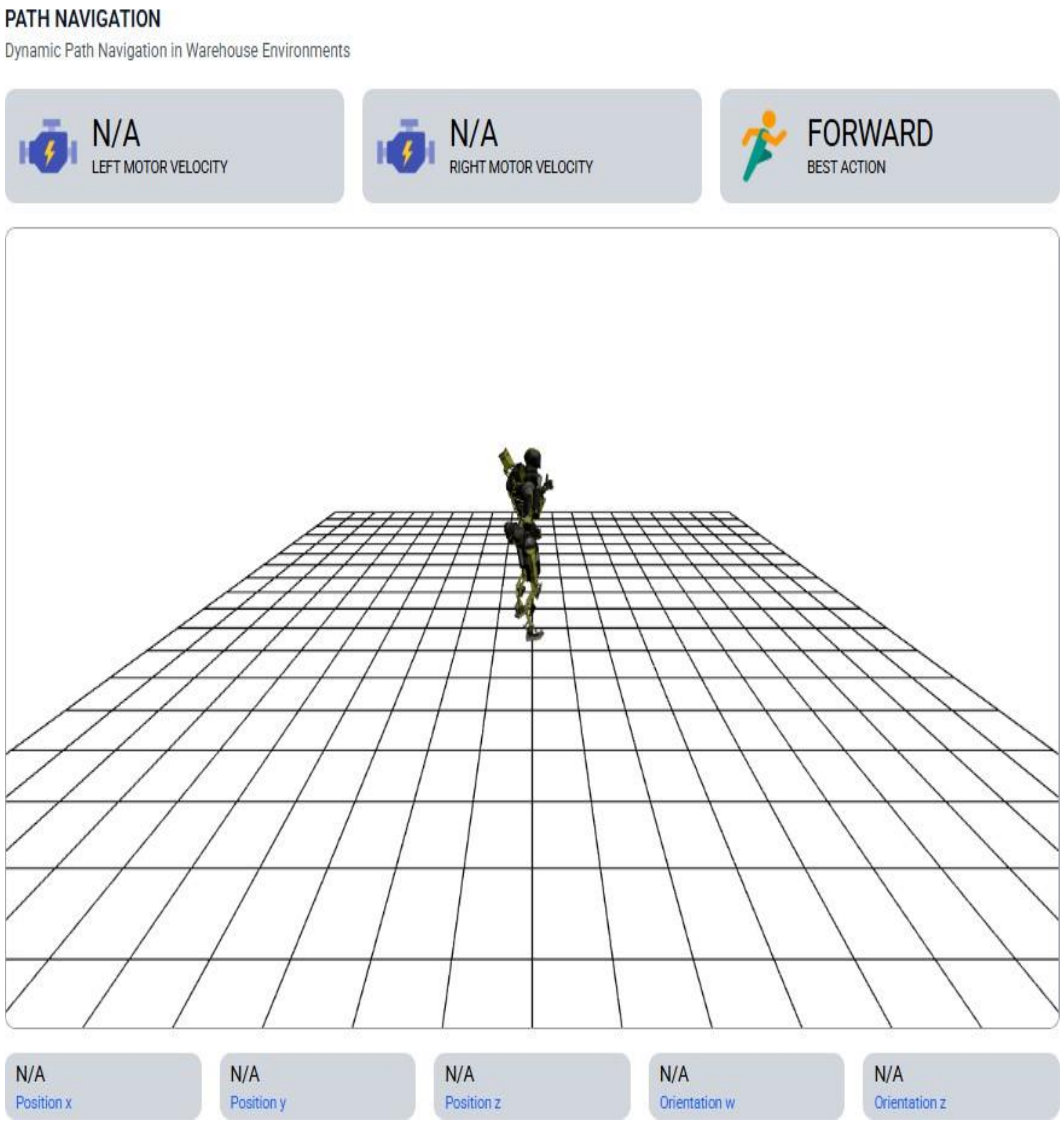
0.0954
RATIO

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

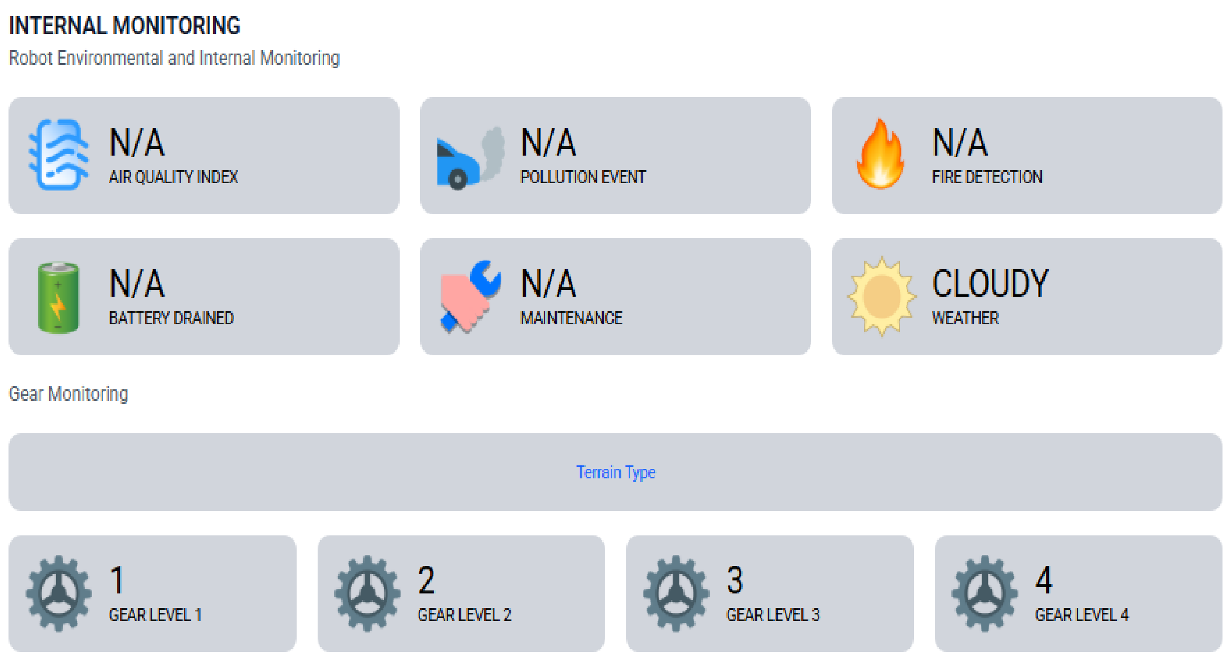
N/A
Orientation w

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

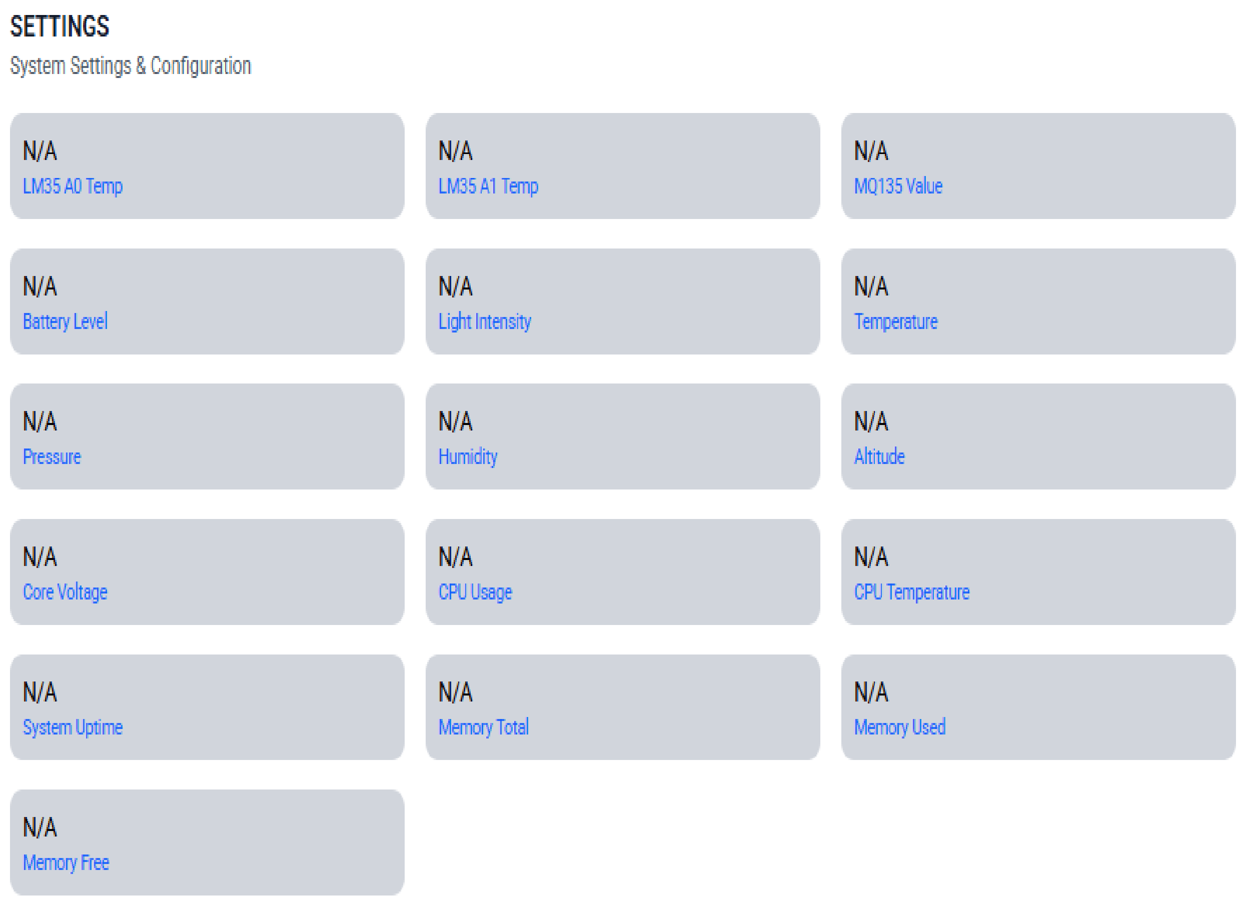
2 GEAR LEVEL 2

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

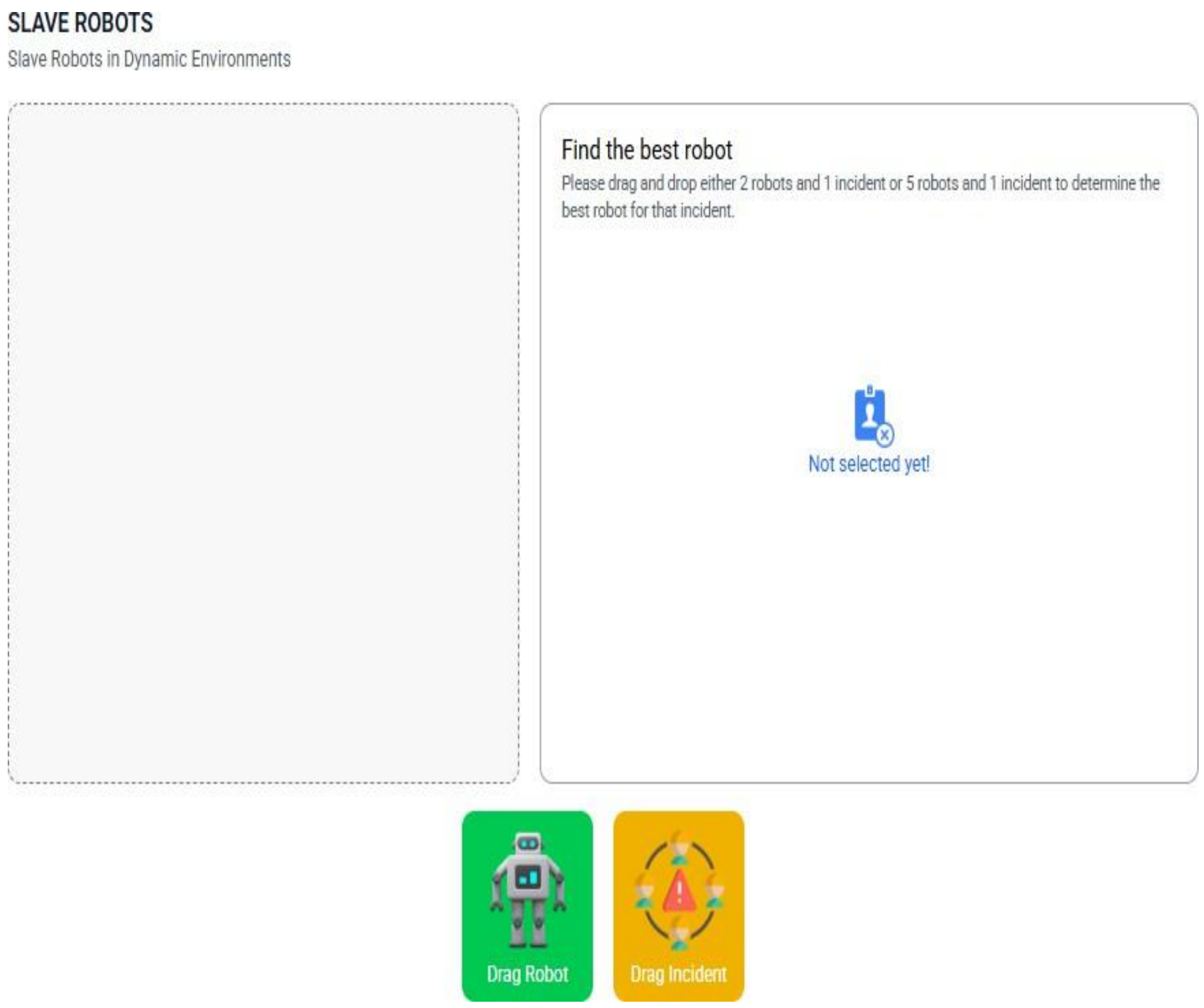

SETTINGS

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.



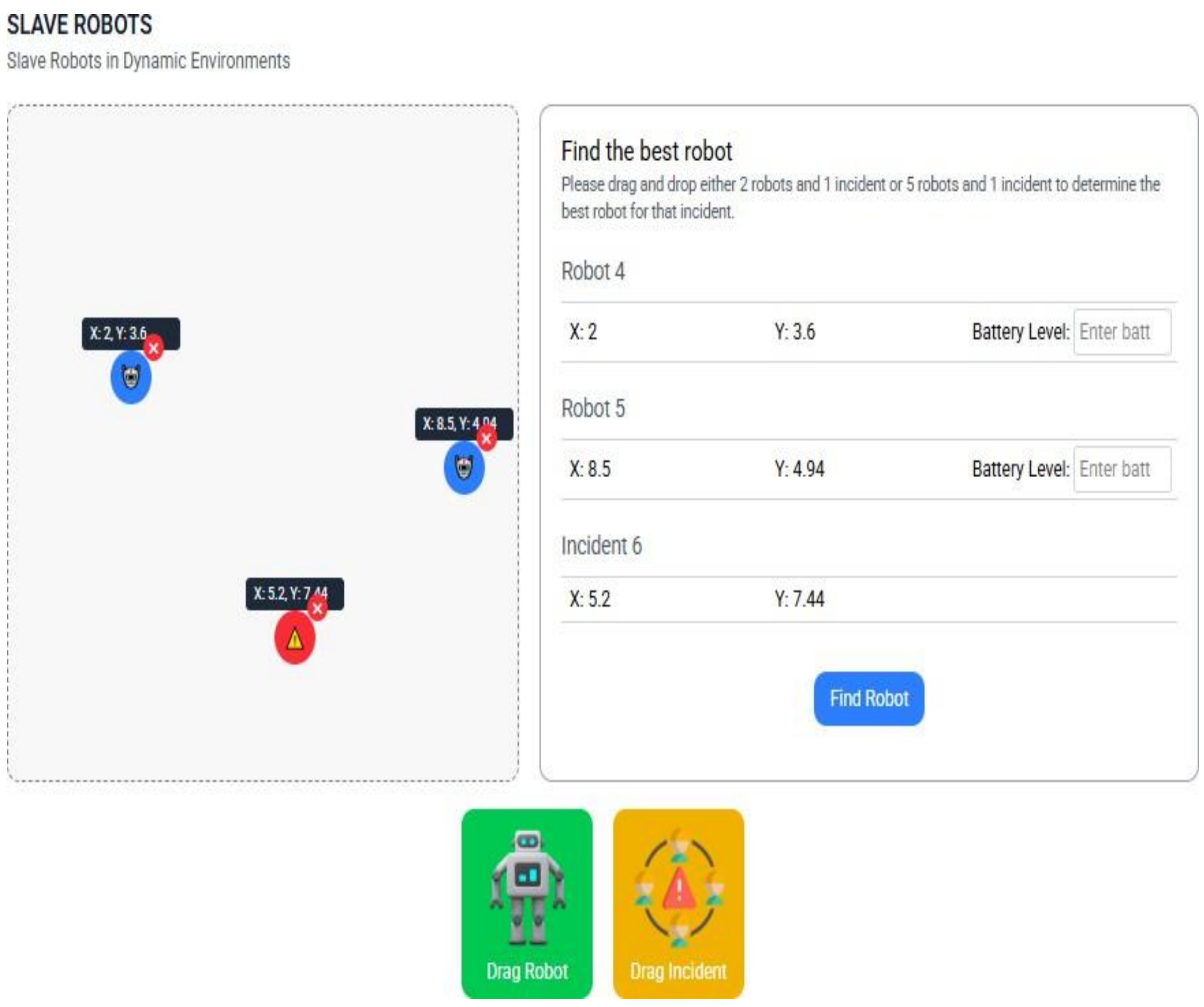
SLAVE ROBOTS

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

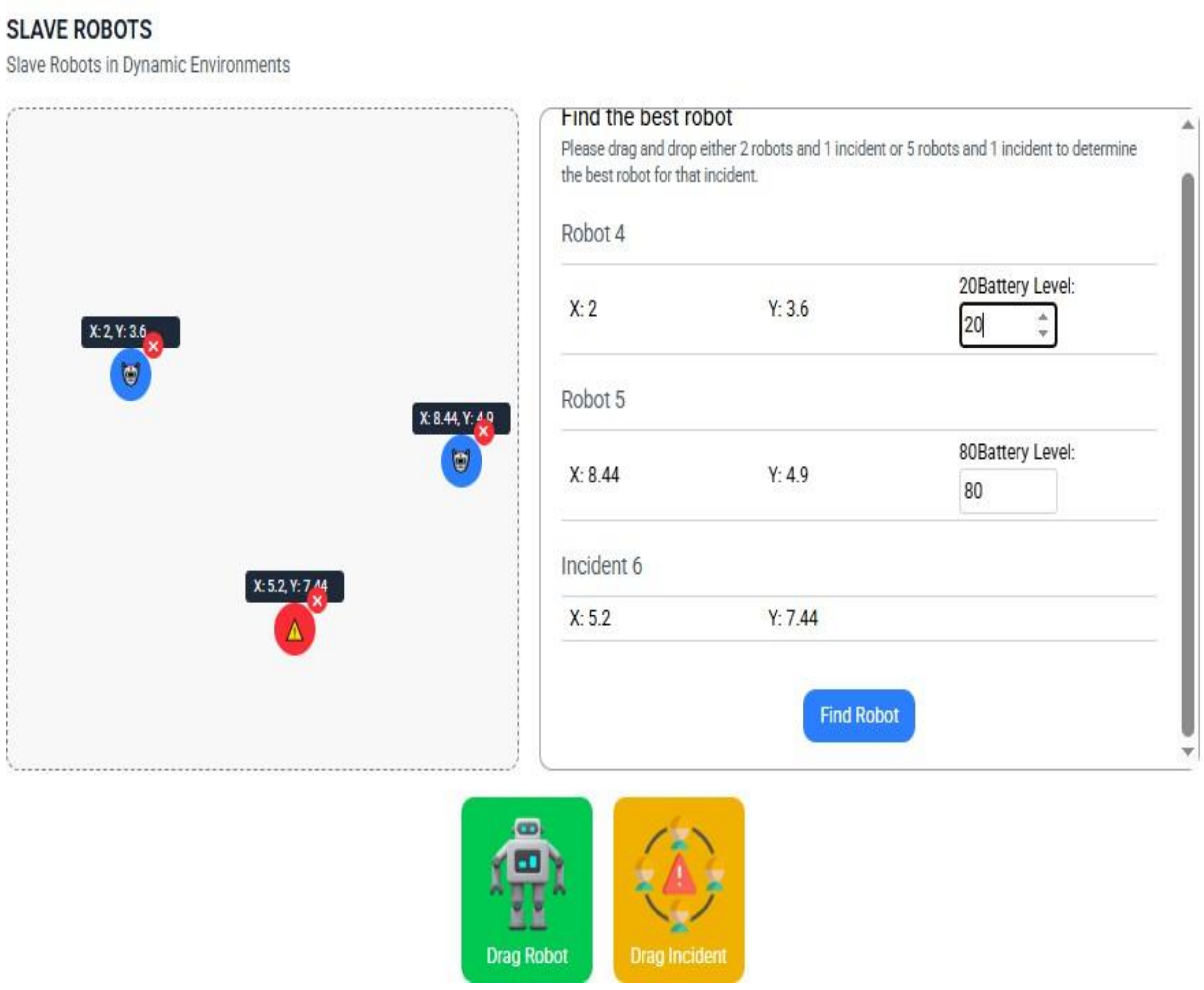
Robot 5

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

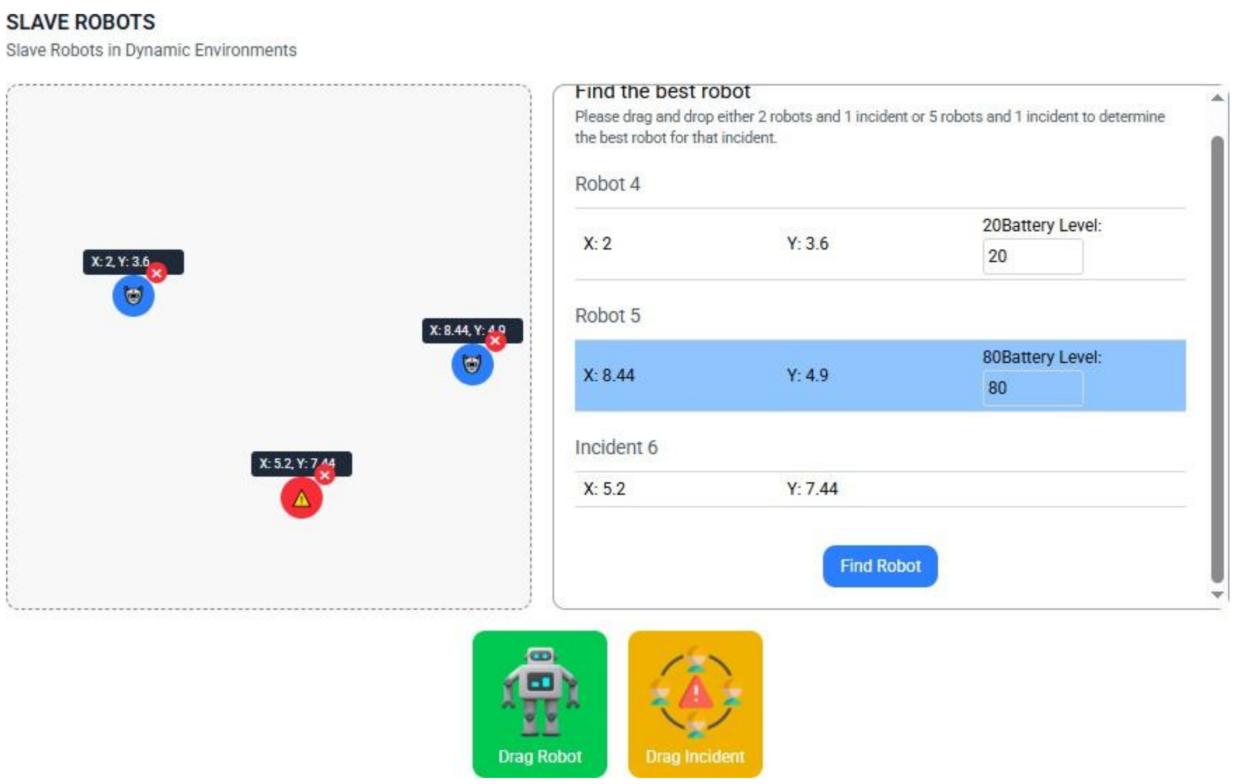
Robot 5

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.

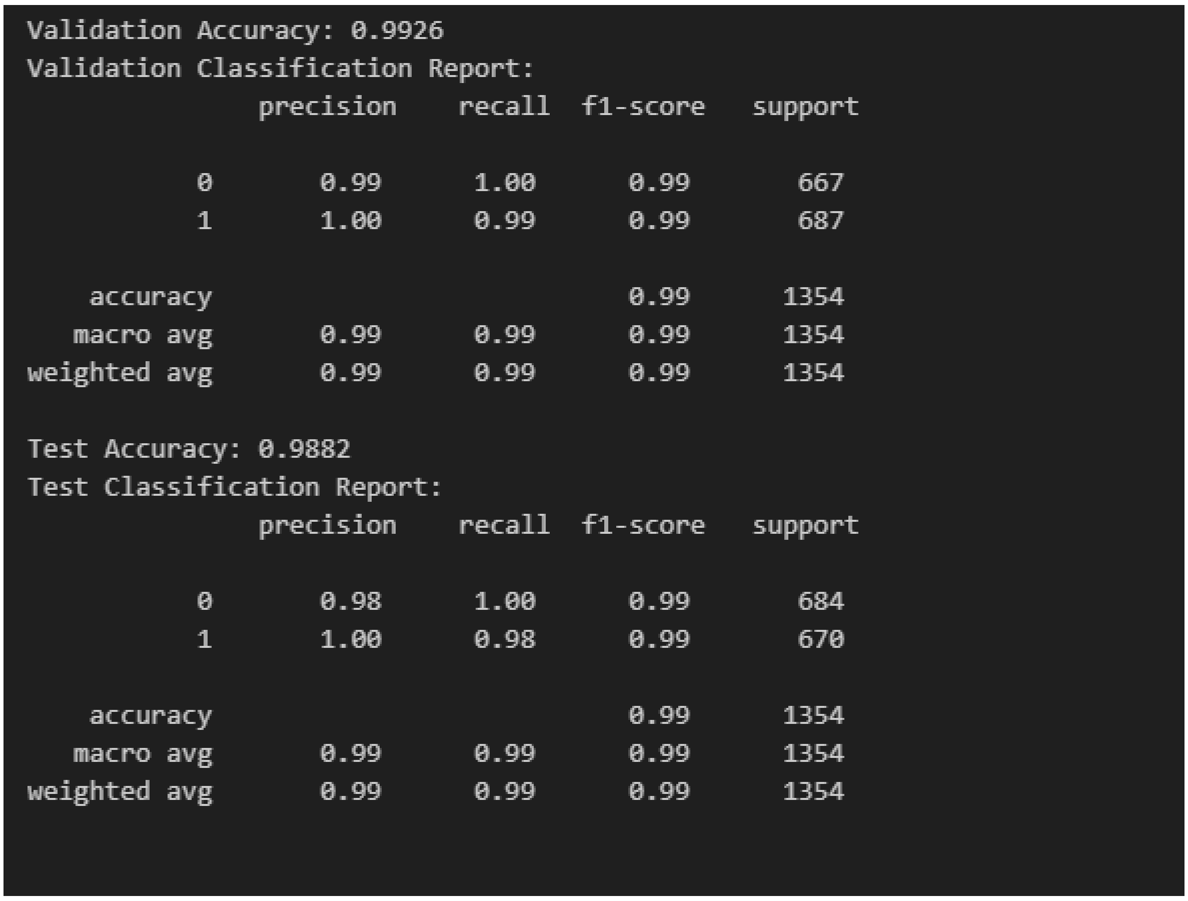
.

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.

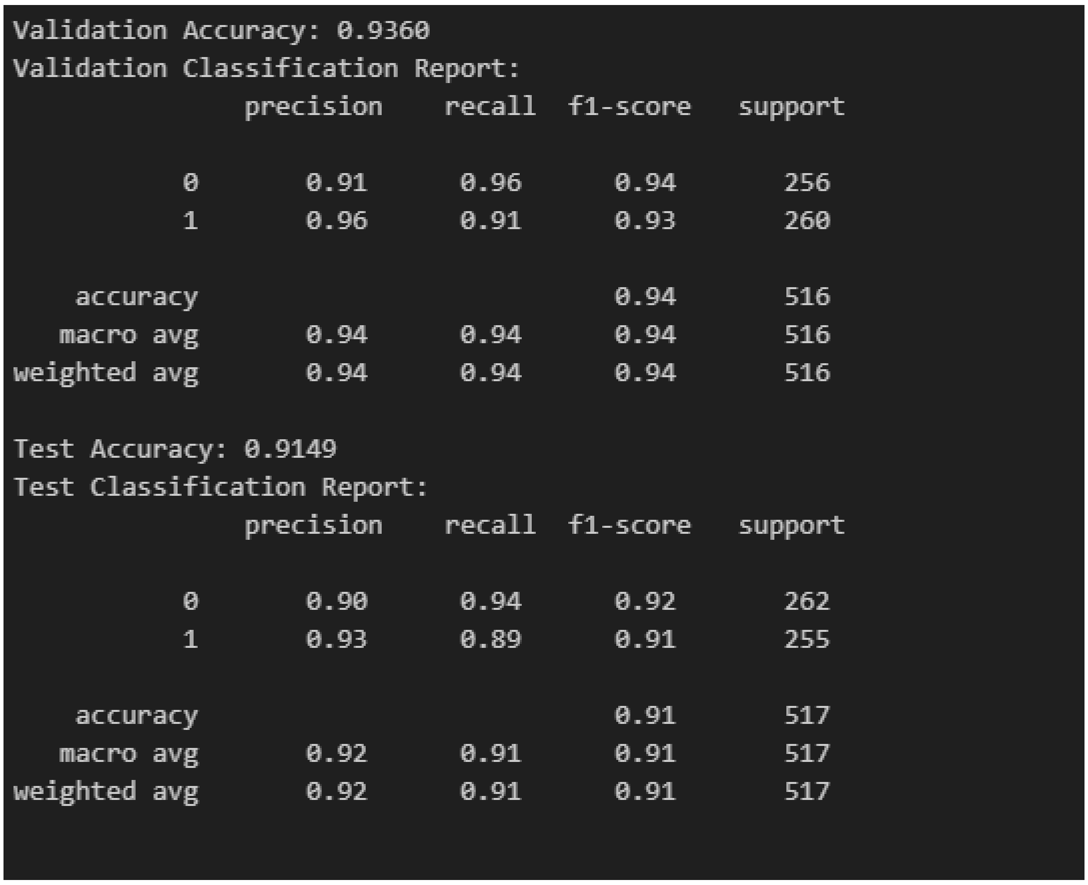
In initial implementation phase the GPS module failed to connect to a satellite. Later we found that it is because we are testing inside a house. GPS module only works when we are outside.

1. Event response model accuracy



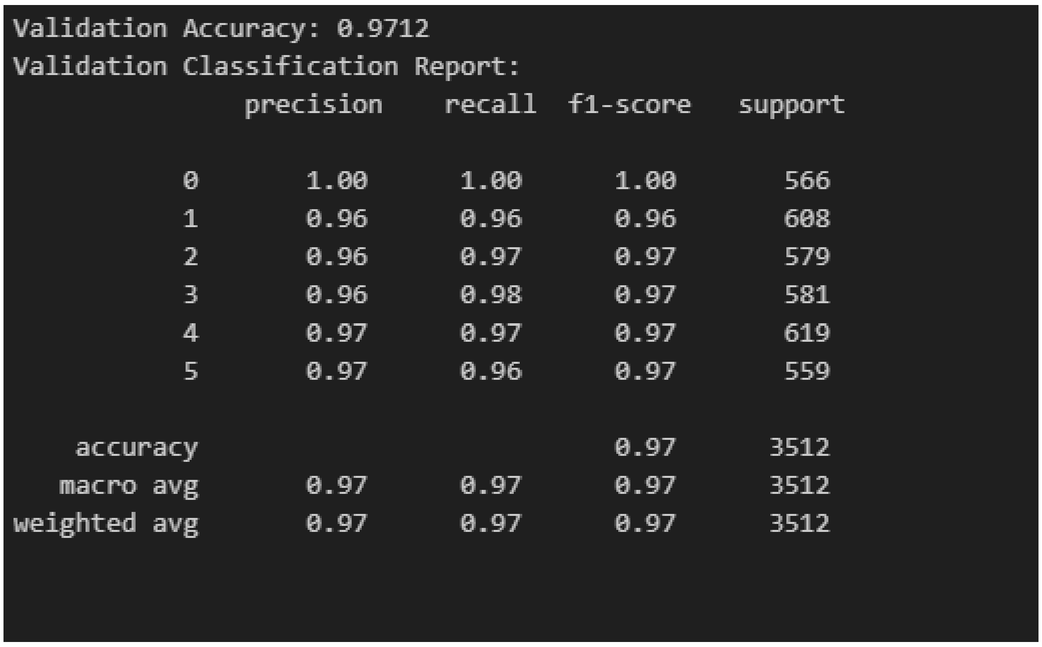
```
Validation Accuracy: 0.9926
Validation Classification Report:
              precision    recall  f1-score   support

           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



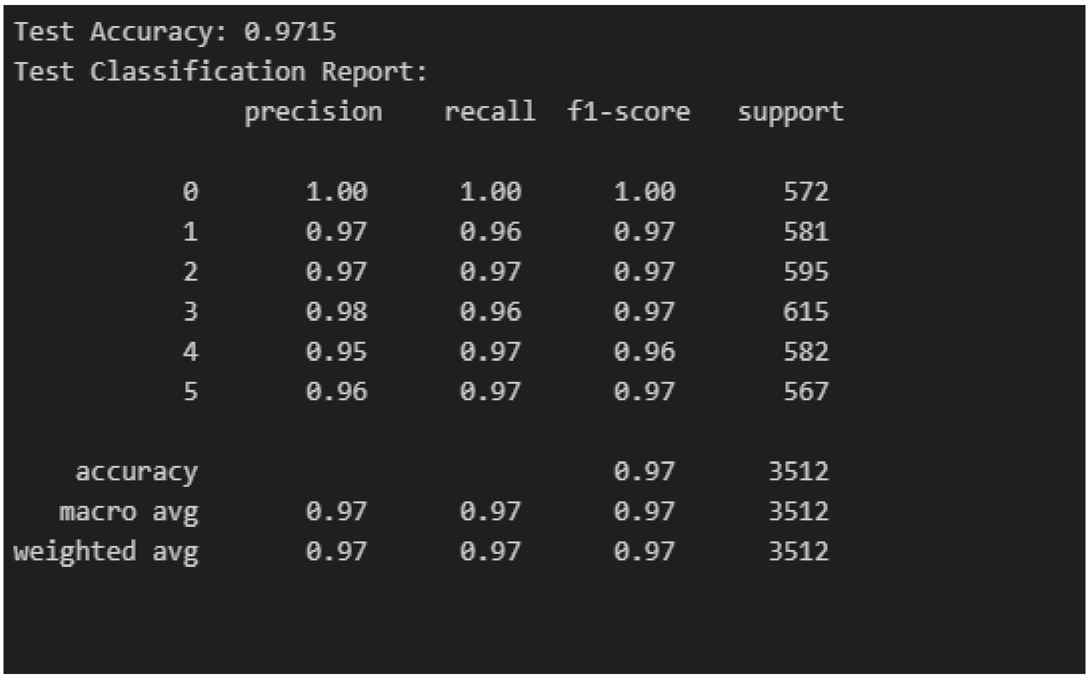
```
Validation Accuracy: 0.9360
Validation Classification Report:
              precision    recall  f1-score   support

           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

Reference List

[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

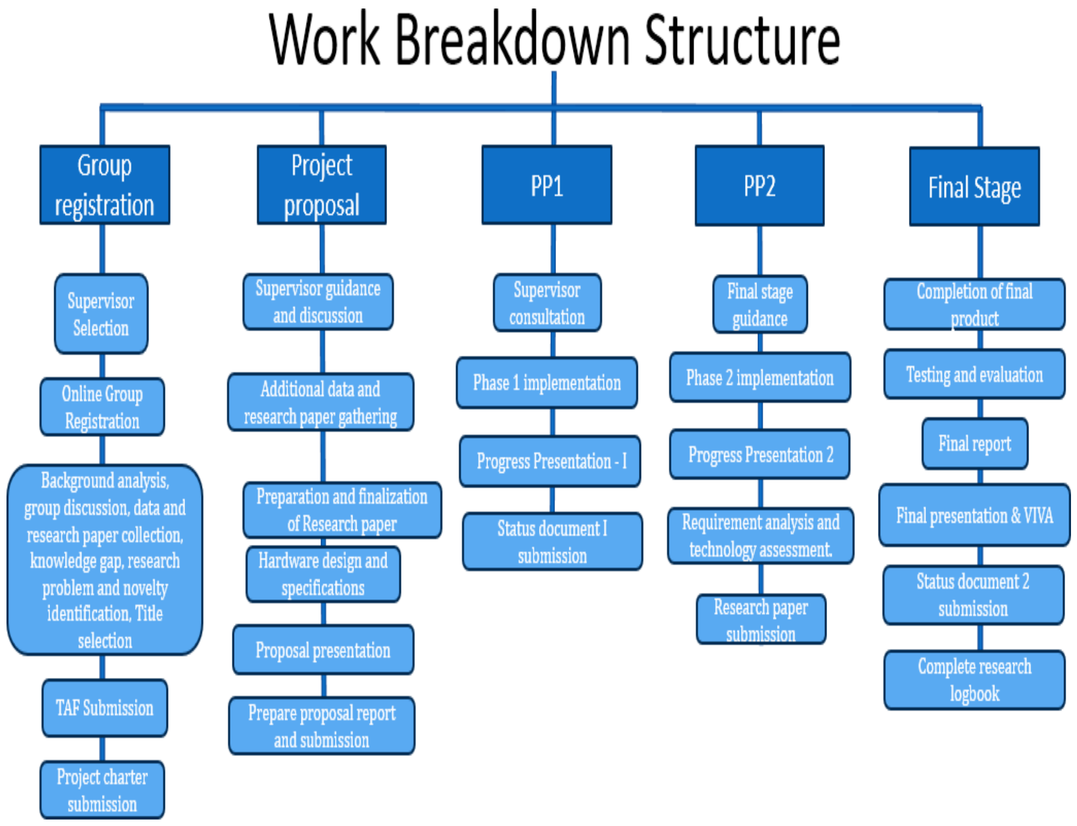
[6] Vivek Upadhyay, Vaibhav Singh, “Roboseciot Based Patrolling Robot”, International Journal of Advances in Engineering and Management (IJAEM), Volume 5, Issue 4 April 2023.

[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.

Appendix A: Work Breakdown Structure


Work Breakdown Structure
Group registration
Supervisor Selection
Online Group Registration
Background analysis, group discussion, data and research paper collection, knowledge gap, research problem and novelty identification, Title selection
TAF Submission
Project charter submission
Project proposal
Supervisor guidance and discussion
Additional data and research paper gathering
Preparation and finalization of Research paper
Hardware design and specifications
Proposal presentation
Prepare proposal report and submission
PP1
Supervisor consultation
Phase 1 implementation
Progress Presentation - I
Status document I submission
PP2
Final stage guidance
Phase 2 implementation
Progress Presentation 2
Requirement analysis and technology assessment.
Research paper submission
Final Stage
Completion of final product
Testing and evaluation
Final report
Final presentation & VIVA
Status document 2 submission
Complete research logbook

Appendix B: Technologies to be used.

MQTT

ARDUINO

RaspberryPi

Firebase
Realtime Database

python