Low Level Design (LLD)

UGV (Unmanned Ground Vehicle) based Surveillance

Revision Number: 2.0
Last date of revision: 02/07/2021

Nidhi Raj Patel

Sambit Rout

Hithul Kannan

• Document Version Control

Date Issued	Version	Description	Author
22nd June 2021	1.1	First Draft	Nidhi Raj Patel
23rd June 2021	1.2	Added Workflow chart	Sambit Rout
23rd June 2021	1.3	Added Exception Scenarios Overall, Constraints	P Sairam
24rth June 021	1.4	Added KPIs	Sambit Rout
24rth June 021	1.5	Added user I/O flowchart	Sambit Rout
24rth June 021	1.6	Model Training	P Sairam
25rth June 021	1.7	Added dataset overview and updated user I/O flowchart.	P Sairam
25rth June 021	1.8	Restructure and reformat LLD	Nidhi Raj Patel
26th June 2021	1.9	Restructuring & Alignment	Hithul Kannan
2nd July 2021	2.0	Added required images, Flowchart, Contents & Restructured Document	Hithul Kannan

Contents

Abstract

An **unmanned ground vehicle** (**UGV**) is a vehicle that operates while in contact with the ground and without an onboard human presence. UGVs can be used for many applications where it may be inconvenient, dangerous, or impossible to have a human operator present. Generally, the vehicle will have a set of sensors to observe the environment, and will either autonomously make decisions about its behaviour or pass the information to a human operator at a different location who will control the vehicle through teleoperation.

1. Introduction

1.1 Why this Low-Level Design Document?

The purpose of this Low-Level Design (LLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The main objective of the project is to detect the mob(illegal) activities, Fire Detection & Accident (Medical Emergency) if it is any accident happen call

for an ambulance and call directly to the helpline number.

UGV surveillance can help:

- Capture the unusual activity, track the location, time, keep the images.
- Describe the performance requirements.
- Include design features and the architecture of the project
- List and describe the non-functional attributes like:
  - Security.
  - Reliability
  - Maintainability
  - Portability
  - Reusability
  - Application compatibility
  - Resource utilization

1.2 Scope

The LLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The LLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system. This software system will be a Web application This system will be designed to detect unusual activity, and fire disasters.

1.3 Constraints

We will only be selecting detecting mob, detecting fire, detecting accident & health emergency.

1.4 Risks

Document specific risks that have been identified or that should be considered.

1.5 Out of Scope

Delineate specific activities, capabilities, and items that are out of scope for the project.

2.Technical specifications

2.1Dataset

Use Cases	Finalized	Source	Link
Mob	yes	Google/Kaggle	https://cutt.ly/rmdTIo9
Fire	Yes	Google/Kaggle	https://cutt.ly/JmdTQyk
Accident & Medical Emergency	Yes	Google/Kaggle/Video Extraction	https://cutt.ly/0mdTw1G

Table 2.0: Dataset Source & Link

2.2 Dataset overview

Consists of 3 different use cases, Mob activity dataset can capture the unusual activity images, and detect the disasters can keep the images where fire happens and in which location. Accident data can help to call in the emergency helpline number to hospital and call an ambulance. Accident video will help to show how much patients are injured and which kind of support they need

2.2.1 Mob

2.2.2 Fire

2.2.3 Accident & Medical Emergency:

2.3 Input schema

Feature name	Datatype	Total Image
Mob	Jpeg, xml, csv & txt	2500
Fire	Jpeg, xml, csv & txt	1512
Accident & Medical Emergency	Jpeg, xml, csv & txt	1391

Table 2.1: Feature names & Total Image count

2.4 Logging

We should be able to log every activity done by the incidents.

- The System identifies at what step logging required
- The System should be able to log each and every system flow.

- Developers can choose logging methods. You can choose database logging/ File logging as well.
- System should not be hung even after using so many loggings. Logging just because we can easily debug issues so logging is mandatory to do.

2.5 Database

System needs to store every request into the database and we need to store it in such a way that it is easy to retrain the model as well.

1. The User chooses the activity dataset.
2. The User gives required information.
3. The system stores each and every data given by the user or received on request to the database. Database you can choose your own choice whether MongoDB/ MySQL.

3. Deployment

1. **AWS**

Fig 3.0: AWS

4.Technology stack

Front End	HTML/CSS/JSt
Backend	Flask
Database	MongoDB
Deployment	AWS
Visualization	Matplotlib, Seaborn, Plotly
Dashboard	Tableau
Version control	GitHub

Table 4.0: Technology Used

5. Proposed Solution

The solution proposed here is an UGV (Unmanned Ground Vehicle) based Surveillance (Unmanned Ground Vehicle) can be implemented to perform above mention use cases. In cases one if UGV detects any mob(illegal) activities at a particular location it will take photos or videos for the evidence and send the police the current location where the mob activities are taking place; the second use case is if UGV detects any natural or human made disasters (fire, smoke, etc..) the UGV detects with its sensors and will send details to concerned authorities and the final use case of UGV is if it finds any health emergency (accident, etc..) it will take emergency action (call ambulance and alert the nearest hospital) for emergency help.

Baseline Model: YOLO V4 TINY, MOBILENET SSD V2, YOLO V5.

6. Model Training/Testing workflow

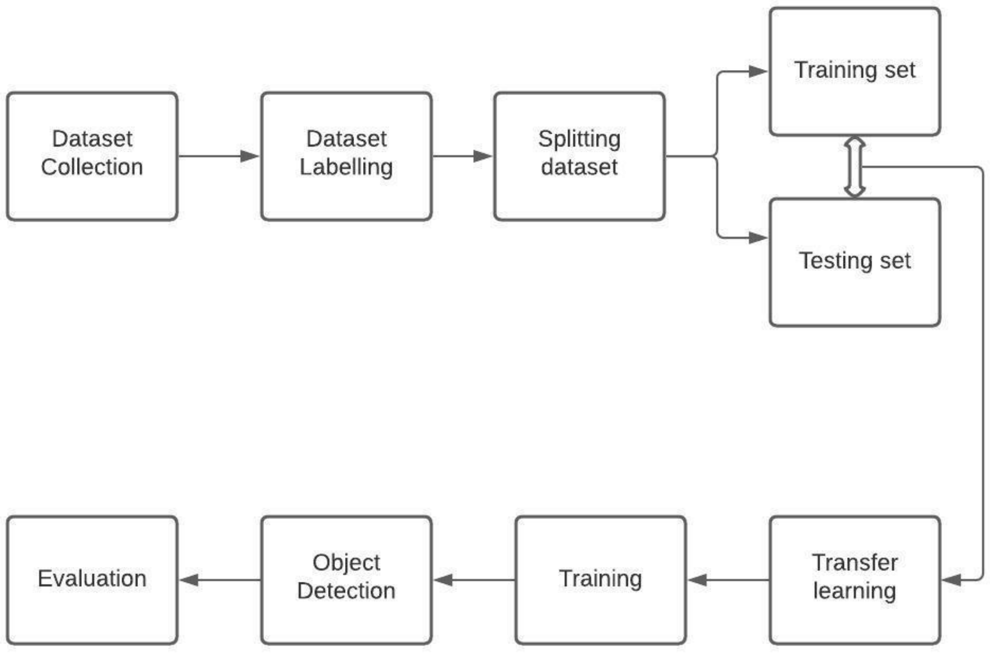

Fig 6.0: Model Training & Testing Workflow

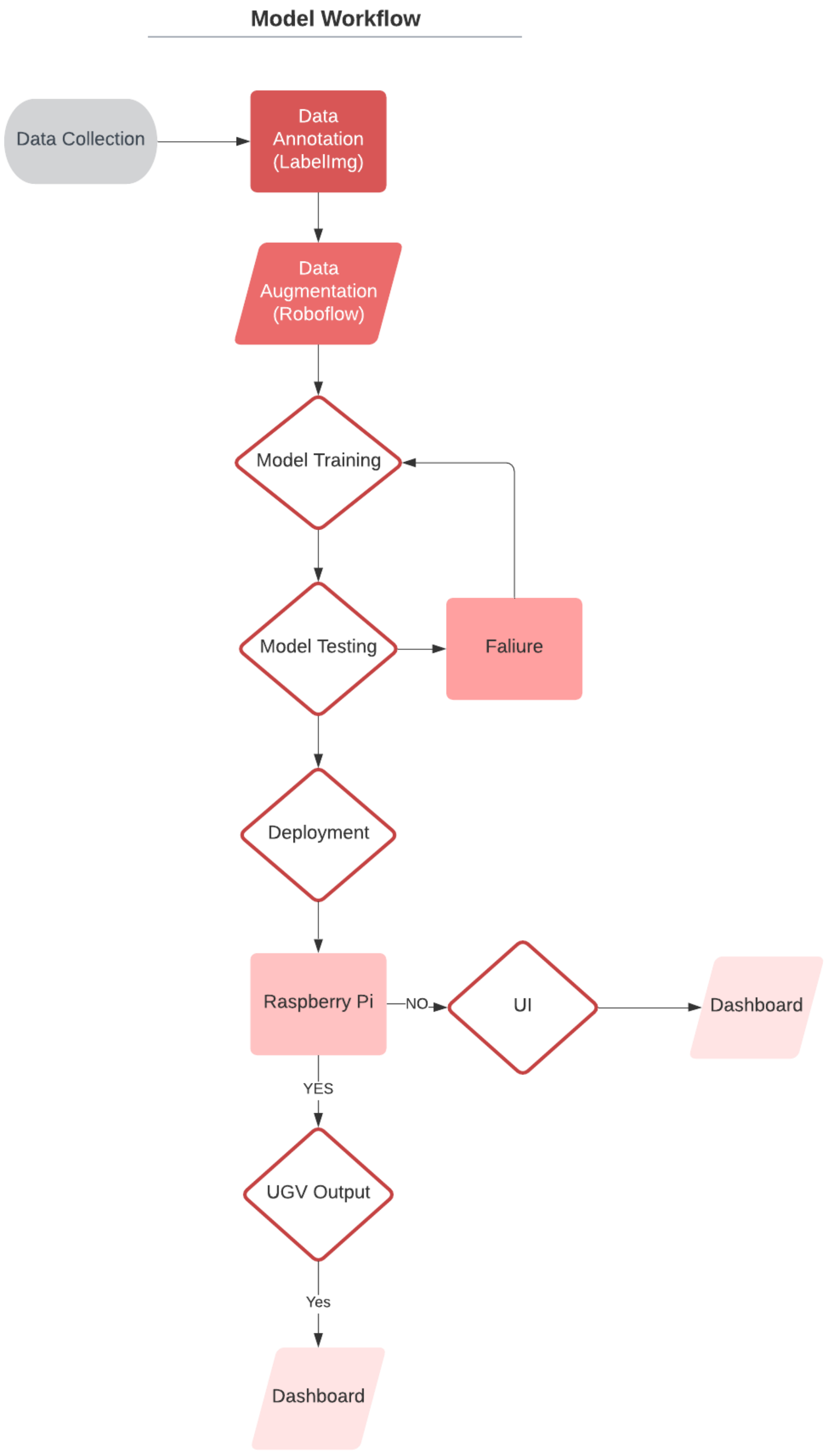

Fig 6.1: Model Overall workflow

7. ROS (Robot Operating System)

7.1 Short About ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms.

7.2 Hardware Requirement for ROS

7.2.1 Raspberry Pi 4 8GB RAM

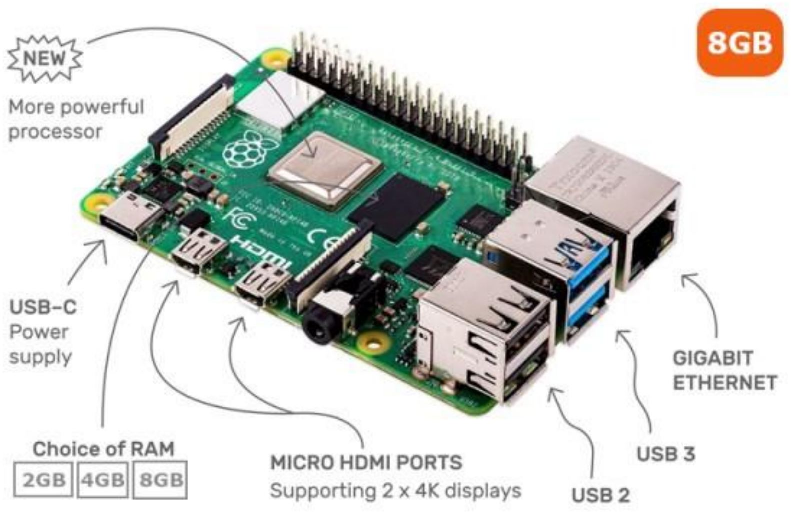

Fig 7.0: Raspberry Pi 4

7.2.2 Lidar

LIDAR is Light Detection and Ranging uses Light in the form of a pulsed laser to detect the obstacle with its ranges. And by having continuous obtaining of ranges throughout the environment with precise angle difference between every ranges it is possible to map the whole environment which is known as Laser scans, which includes Ranges, Angles, Cartesian coordinates and Number of the value obtained. Based on Physical and Scattering process and platform LIDAR has many classifications.

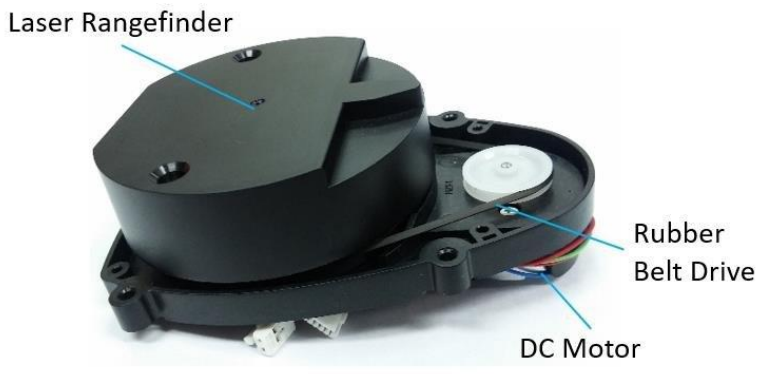

Fig 7.1: Lidar

7.3 Software Requirements for ROS

7.3.1 Robot Operating System (ROS)

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. ROS is a framework on top of the O.S. that allows it to abstract the hardware from the software. This means you can think in terms of software for all the hardware of the robot.

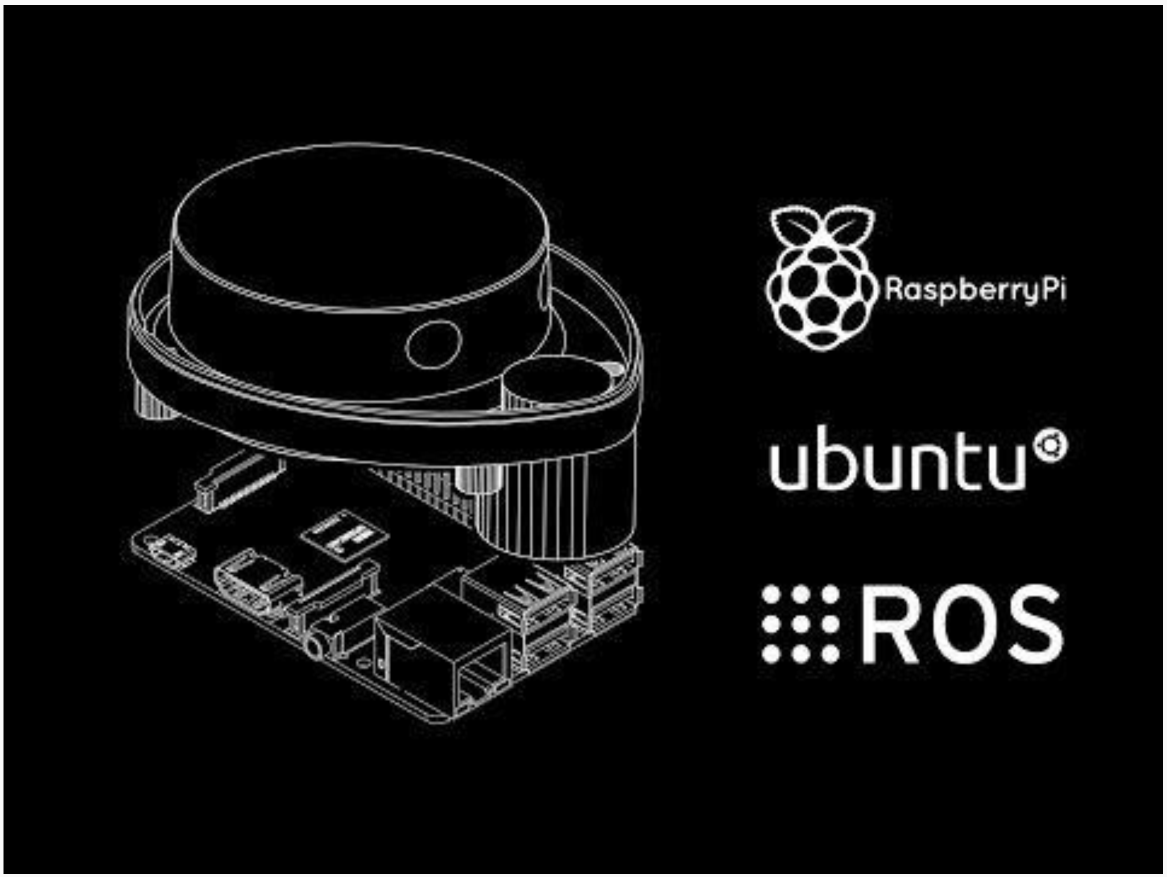

Fig 7.2: ROS Software

7.4 Mapping

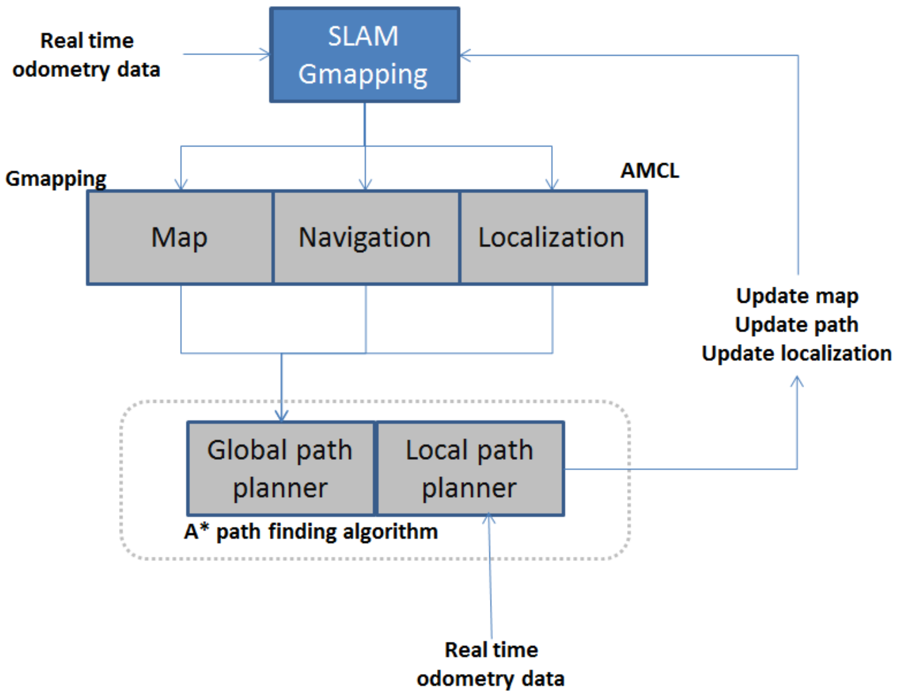

Fig 7.3: Mapping Block Diagram

7.5 SLAM

SLAM is Simultaneous Localisation and Mapping. It is used to map the unknown environment by tracking the current location of the Bot in that same environment. By using SLAM algorithm complete indoor servicing can be done by using Mapping with path planning and following.

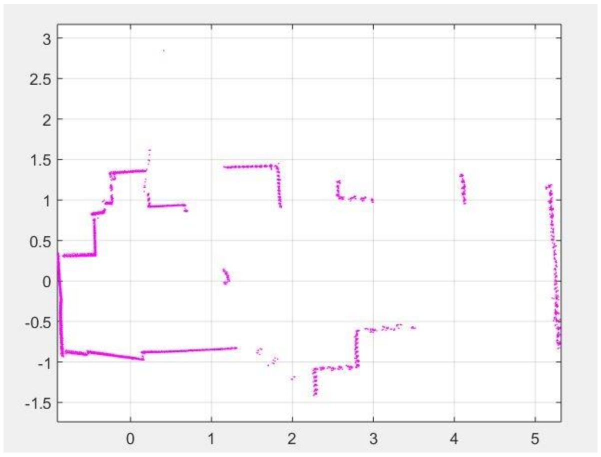

Fig 7.4: Indoor Mapping with LIDAR & SLAM

8. Dashboard

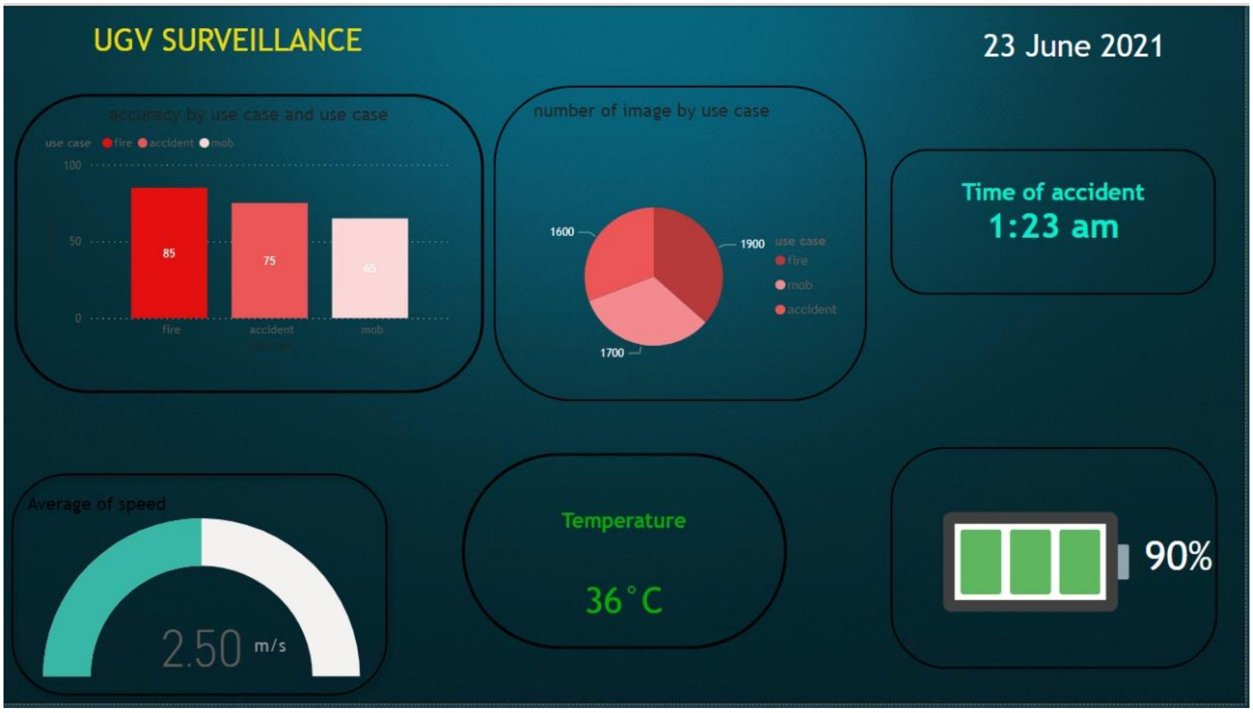

Fig 8.0: Dashboard Interface

9. Error Handling

Should errors be encountered, an explanation will be displayed as to what went wrong?

An error will be defined as anything that falls outside the normal and intended usage.

10.Test Result Based on Use Cases

Use case	Model	Accuracy
Mob	YOLO V4 TINY	65%
Accident	YOLO V4 TINY	75%
Fire	YOLO V4 TINY	85%

Fig 10.0: Model Accuracy Based on Use Case

11.Key performance indicators (KPI)

- Key indicators displaying a summary of the anomaly detection in the society/area.
- Time and workload reduction using the UGV based surveillance.
- To detect mob (illegal) activities and inform police.
- On time alert to the nearest hospital on medical emergency (accident).
- Taking adequate evidence of the mob.
- Send disaster details to concerned authorities.
- Display of battery life and percentage of UGV.
- Distance travelled by UGV.
- Get the exact location of UGV.

12. Conclusion

The Designed UGV (Unmanned Ground Vehicle) will analyse the incident based on the data trained using our algorithm. We can identify the incident and transmit the information, so that concerned authority can take necessary action in order to control the situation.

13. Reference

1. https://en.wikipedia.org/wiki/Unmanned_ground_vehicle
2. Google.com for images of UGV hardware.
3. https://www.ros.org/