

---

# **Graduation Project Report**

for

**N.A.V.I.**

Prepared by :

**Ahmed Mohamed – 2351724**

**Hevra Petekkaya – 2456200**

**Egemen Aksoz – 2315083**

**Middle East Technical University NCC**

**Computer Engineering Department**

**Supervised by Prof. Yeliz Yesilada**

**17/06/2023**

# Contents

|       |   |    |
|-------|---|----|
| 1     | Introduction.....                                 | 3  |
| 1.1   | Purpose .....                                     | 3  |
| 1.2   | Scope.....  | 3  |
| 1.3   | Identification of constraints.....                | 3  |
| 1.4   | Related Work .....                                | 4  |
| 1.5   | Product Overview .....                            | 8  |
| 1.5.1 | Product perspective .....                         | 8  |
| 1.5.2 | Product functions.....                            | 8  |
| 1.5.3 | Identified stakeholders and design concerns ..... | 9  |
| 1.5.4 | User characteristics.....                         | 9  |
| 1.5.5 | Limitations .....                                 | 9  |
| 1.5.6 | Assumptions and dependencies .....                | 9  |
| 2     | Specific requirements .....                       | 9  |
| 2.1   | External interfaces .....                         | 9  |
| 2.2   | Functions.....                                    | 10 |
| 2.3   | Usability Requirements.....                       | 12 |
| 2.4   | Performance requirements.....                     | 12 |
| 2.5   | Logical database requirements .....               | 13 |
| 2.6   | Software system attributes .....                  | 13 |
| 2.7   | Supporting information .....                      | 13 |
| 3     | Software Estimation .....                         | 17 |
| 4     | Architectural Views.....                          | 19 |
| 4.1   | Logical View .....                                | 19 |
| 4.1.1 | Class Diagram .....                               | 19 |
| 4.2   | Process View .....                                | 20 |
| 4.2.1 | Activity Diagram .....                            | 20 |
| 4.2.2 | Sequence Diagrams .....                           | 21 |
| 4.2.3 | Data Flow Diagrams.....                           | 22 |
| 4.3   | Development View.....                             | 24 |
| 4.3.1 | Component Diagram .....                           | 24 |
| 4.4   | Physical View .....                               | 25 |
| 4.4.1 | Deployment Diagram .....                          | 25 |
| 5     | Software Implementation .....                     | 26 |
| 6     | Software Testing.....                             | 27 |
| 6.1   | Unit Testing.....                                 | 27 |

|       |                                  |    |
|-------|----------------------------------|----|
| 6.1.1 | Test Procedure .....             | 27 |
| 6.1.2 | Test cases .....                 | 30 |
| 6.1.3 | Test Results .....               | 31 |
| 6.1.4 | <i>Test Log</i> .....            | 32 |
| 6.2   | Integration Testing .....        | 32 |
| 6.2.1 | Test Procedure .....             | 32 |
| 6.2.2 | Test cases .....                 | 33 |
| 6.2.3 | Test Results .....               | 34 |
| 6.2.4 | <i>Test Log</i> .....            | 34 |
| 6.3   | System Testing .....             | 34 |
| 6.3.1 | Test Procedure .....             | 34 |
| 6.3.2 | Test Scenarios .....             | 34 |
| 6.3.3 | Test Results .....               | 35 |
| 6.3.4 | <i>Test Log</i> .....            | 35 |
| 6.4   | API Performance Testing .....    | 36 |
| 6.4.1 | Test Procedure .....             | 36 |
| 6.4.2 | Test cases .....                 | 38 |
| 6.4.3 | Test Results .....               | 39 |
| 6.4.4 | <i>Test Log</i> .....            | 42 |
| 6.5   | Model Performance Testing .....  | 43 |
| 6.5.1 | Test Procedure .....             | 43 |
| 6.5.2 | <b>Expectations</b> .....        | 43 |
| 6.5.3 | Test Results .....               | 44 |
| 6.5.4 | <i>Test Log</i> .....            | 45 |
| 7     | Project Scheduling .....         | 45 |
| 7.1   | Milestones and Tasks .....       | 45 |
| 8     | Conclusion .....                 | 46 |
| 9     | References .....                 | 47 |
| 10    | Appendices .....                 | 48 |
| 10.1  | Acronyms and abbreviations ..... | 48 |
| 10.2  | Glossary .....                   | 49 |

# 1 Introduction

## 1.1 Purpose

NAVI is a mobile application aimed to assist visually impaired people in reaching their destination by avoiding any potential obstacles such as cars, trees, bicycles, etc. This will be achieved by utilizing a machine learning model that can detect objects accompanied by a system that can measure the distances between the objects and the user. We believe that in order for the user to be able to seamlessly travel around, having knowledge of only the objects in front of the visually impaired is not a piece of adequate information unless they also know how far or how close the objects are to them and in what angle they are with respect to them. Privileges are invisible to those having them and they are usually taken for granted. Being able to easily walk around without having the fear that we might encounter an accident is a good example of an ability we have that we take for granted. However, according to the World Health Organization, there are at least 2.2 billion people having near or distance vision impairment (World Health Organization, 2022). Consequently, there are at least 2.2 billion people lacking a very basic yet much-needed ability that many of us simply take for granted. So why is it now that we are attempting to approach and solve this problem? Are there not solutions out there that are already aimed at solving this issue? Of course, there are; however, they were all built with technologies that were available at that time. With the advancement of time, technology is also improving like never before, especially with the arrival of machine learning. Hence, we believe that it is the right time to look for other alternative solutions that potentially can bring more mobility by eliminating the need for external apparatus, affordability by providing a free mobile application, and overall better user experience.

## 1.2 Scope

The goal of our project is to propose an alternative approach by building a mobile app that can assist visually impaired people with navigation both indoors and outdoors. The assistance will be provided by letting the user know about the identity of the objects and the distances of the objects from the user. Allowing the visually impaired to have a navigation experience as similar as possible to a person with no visual impairment will be considered to be a benefit of the project. In addition to that, we hope that this project will act as a motivation and as another source for people who are also looking forward to working on research projects with similar ideas. As for the objectives of this project, the below bullet points can be checked:

- A user-friendly frontend for the mobile app is needed to be built so that the user can interact with the system.
- A machine/deep learning model that is trained for object detection needs to be finetuned to detect the specific set of objects that we believe are most common to be encountered indoors and outdoors. In order to be fine-tuned for the detection of the target objects, a new dataset needs to be prepared.
- Either the machine/deep learning model that is trained for detection needs to be extended to learn distance and angle estimation as well, or another system or technique detached from the machine/deep learning model needs to be utilized for distance estimation and angle estimation.

## 1.3 Identification of constraints

- The project operates on no budget, meaning that premium software and cloud hosting solutions are not accessible and have to be substituted with local hosting and custom scripts that are less efficient.

- Bystanders might get picked up by the camera, to ensure that their privacy is not violated, the image captured is never stored or sent as a raw image, the image is discarded once predictions have been made.
- The application operates in real time, meaning we need to ensure that response time of the server and the turnaround time for a request are minimized and are within acceptable ranges.

#### 1.4 Related Work

In research projects such as ours, literature reviews play a very crucial role. That is because the aims of research projects are to propose an alternative, which is expectedly better in some ways, to what already exists or propose a completely new approach to solve a problem. When it comes to our research project, it is more about both providing an alternative to already existing solutions, which utilize completely different technologies, and possibly improving the already existing solutions using similar technology to ours. Let's start with already existing solutions utilizing similar technology to ours and explain how ours will be different from theirs. As a first example, VIZIYON (SreerajM, 2020): Assistive handheld device for the visually challenged, utilizes a handheld device that is powered by an Arduino microcontroller, a Raspberry Pi, and a Raspberry Pi camera to capture real-time video. As can be seen in Figure 1, for object detection, the Fast-RCNN convolutional network model is being used and placed on the Raspberry Pi. It is worth noting that Arduino is a microcontroller, whereas Raspberry Pi is a microprocessor having its own operating system. An Arduino on its own can't perform complex calculations, hence the need for a Raspberry Pi. As for distance calculation, an Ultrasonic sensor is used. For our proposal, we aim for minimum dependency on external resources as possible from the user's perspective. For that reason, instead of a Raspberry Pi camera, we will be using the already existing camera on the user's mobile phone. Moreover, instead of the user being required to have an additional microcontroller and microprocessor of their own, in our proposal, we only require the user to be connected to the internet and the rest of the processing will be taken care of in the cloud. The reason behind our willingness to perform the processing on the cloud is to free all the users from the burden of carrying the hardware needed for processing with themselves. As a final difference, our approach will be using a model from the YOLO family instead of Fast-RCNN. There are several reasons behind this, an important reason being that we plan to use a model which extends the YOLO model to learn distance estimation as well, which will also eliminate the need of having a sensor. Further details about our model choice will be discussed in the product overview part.

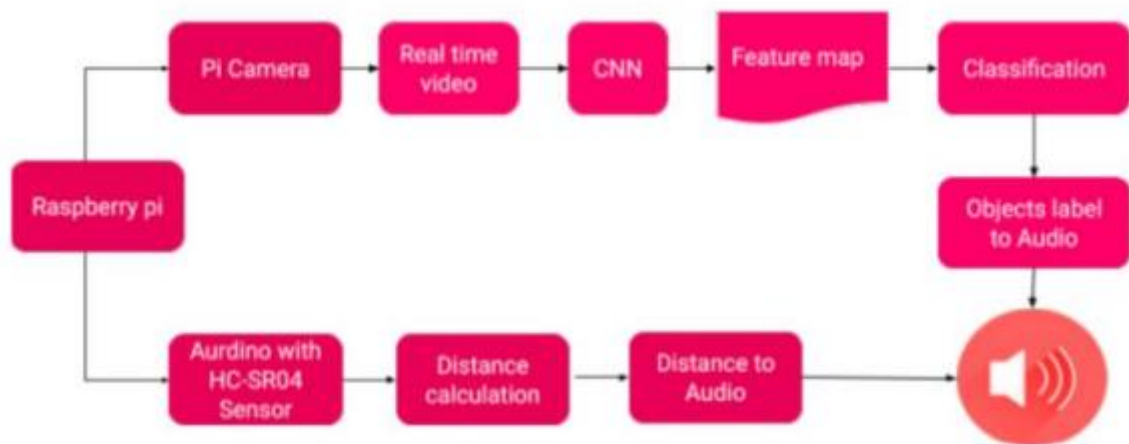


Figure 1: VIZIYON Implementation (SreerajM, 2020)

As another literature review, let's examine the Deep Learning Based Audio Assistive System for Visually Impaired People (S. Kiruthika Devi, 2021). In this paper, YOLOv3 CNN architecture is used for multilabel object detection. As for giving positional guidance, as it can be seen in Figure 2, it divides 4 the user's viewpoint into 9 sectors combining top, center, and bottom as the vertical position combined with left, center, and right as the horizontal position. YOLOv3 doesn't only perform object classification but also performs detection implying that it also has positional knowledge: x and y coordinates. By using this positional knowledge as an input to a hardcoded algorithm, which can be found in Figure 3, for location finding, positional guidance is provided.

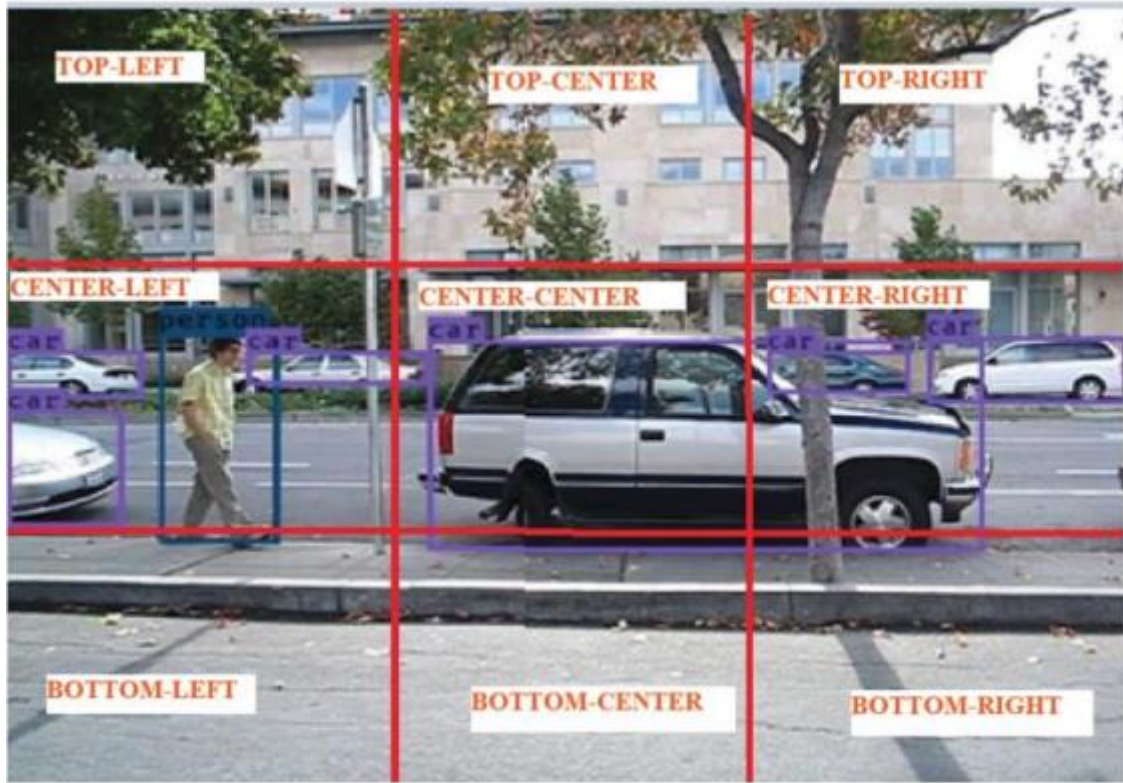


Figure 2: Screen Division for direction guidance (S. Kiruthika Devi, 2021)

#### Pseudocode Location Finding ()

```

center_x = round((2 * x + w) / 2)
center_y = round((2 * y + h) / 2)
if center_x <= W / 3:
    W_pos = "left"
elif center_x <= (W / 3 * 2):
    W_pos = "center"
else:
    W_pos = "right"
if center_y <= H / 3:
    H_pos = "top"
elif center_y <= (H / 3 * 2):
    H_pos = "center"
else:
    H_pos = "bottom"

```

Figure 3: Pseudocode Location Finding (S. Kiruthika Devi, 2021)

So, how is our proposal different from this study? In our proposal, we will be giving the user distance estimation and the angle the object is at with respect to the user rather than giving

them an approximation of where the object might be from a specific set of sectors. We believe that our implementation has both pros and cons compared to theirs. As a pro, in our implementation, the user will have the knowledge of how far the object is and what angle it is located at with respect to them so that they can take precautions accordingly, e.g., no need to worry about hitting a tree that is 20 meters away; in other words, the objects close by are more of an obstacle that needs to be considered when compared to far away objects.

Now, let's examine an AI Suitcase that is more advanced and with more functionalities, (Page, 2020). This suitcase includes sensors and cameras. In addition to the suitcase, in order to program a destination, plan a route, give vibrations in the suitcase's handle so that the user can be directed in the right direction, a mobile phone application is being used. Furthermore, this suitcase can also perform facial recognition so that it can let the user know if there is a friend nearby.

Unfortunately, regarding the machine learning algorithms used or the way the sensors are utilized, no information is given since this product might be commercialized. As previously mentioned, this suitcase has more functionalities than our proposed model; however, when it comes to the ease of usability, our proposed model performs better because having the suitcase with you wherever you go is not very practical.



*Figure 4: AI Suitcase (Page, 2020)*

Apart from the approaches that use similar technology to ours, such as using machine/deep learning models, we can also investigate other approaches hoping to gain some insights from them as well. As previously mentioned, the solutions developed to solve a problem are built upon the available technology of that period of time. Knowing that even though the concept of machine/deep learning traces back to 1943, the practical work that was done with it didn't come around until the 2000s because machine/deep learning requires a vast amount of data and that amount of data wasn't around even until after 1983, the official birthday of the Internet. The reason behind this is that even though there were connections between devices, there weren't any sources, such as social media applications, that data couldn't have been generated from. In short, now that we have all the ingredients, machine/deep learning technology has risen and is



being used; however, let's travel back in time and see what technology or tool was around to help the visually impaired.

Most commonly used were the white canes, which can be seen in Figure 5. These tools can only help the visually impaired to a maximum of 1.5 meters. The canes will touch an object and then the user will change their direction not to topple over but the user doesn't have a clear idea of what the object was. In other words, the white canes don't help the visually impaired picture the environment that they are present in, unlike our proposed model where the visually impaired will both know the identity of the object and their distance from it. As another example, there are also guide dogs that can assist the blind or visually impaired. However, they aren't affordable for lots of people, and even after getting one, they still have their veterinary expenses. Putting the expenses aside, guide dogs also require care to be given: grooming, feeding, etc.



Figure 5: A person using white cane for assistance (Everything You Need to Know About White Canes, 2021)

Now that we have an overview, which can also be seen in Table 1, of what was there for the visually impaired before machine/deep learning and after machine/deep learning, we can finally feel confident to analyze our own proposal.

| Compare/Contrast | Model Used | Performs Object Detection | Performs Distance Calculation | Performs Route Planning | Gives Audio Warnings | Ease of Use |
|------------------|------------|---------------------------|-------------------------------|-------------------------|----------------------|-------------|
| N.A.V.I.         | YOLOv8     | Yes                       | Yes                           | No                      | Yes                  | High        |
| Smart Suitcase   | Unknown    | Yes                       | No                            | Yes                     | No                   | Low         |
| VIZIYON          | R-CNN      | Yes                       | Yes                           | No                      | Yes                  | High        |
| Rudimentary Cane | N/A        | No                        | No                            | No                      | No                   | Medium      |
| DLAASVI*         | YOLOv3     | Yes                       | No                            | No                      | Yes                  | N/A**       |

Table 1: Comparison between NAVI and other assistive navigation tools

\* Deep Learning Based Audio Assistive System for Visually Impaired People.

\*\* Available only as a model. Hasn't been implemented on a usable device.



## 1.5 Product Overview

### 1.5.1 Product perspective

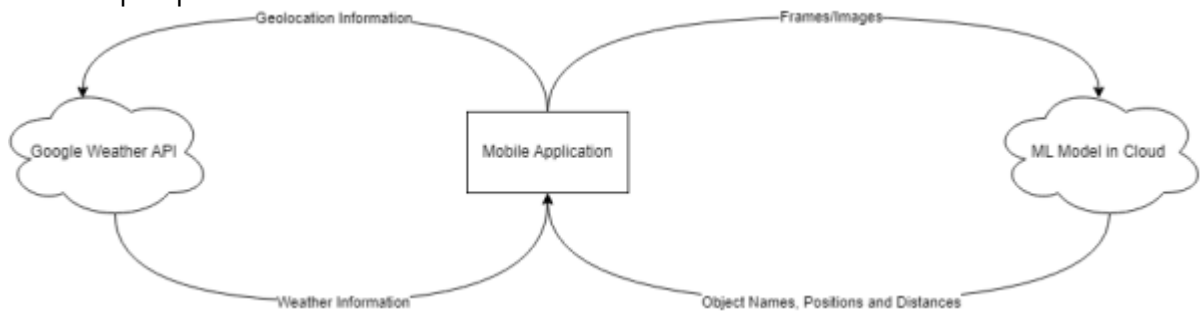


Figure 6: Project N.A.V.I. Context Model

Figure 6 above shows the contextualized structure of our system and how it interacts with external systems. The client code, frontend, will be running in a mobile application. Furthermore, to get the weather information, the mobile application provides an API token to Google Weather API, in case a valid Zip Code is unavailable, the user's geolocation information will be requested and used for weather forecasting. Finally, to get the detected objects, their distances, and angles with respect to the user, the mobile application sends requests with accompanied frames/images to an external cloud server, where the frames will be processed and the object names, distances, and angles will be returned from it.

#### 1.5.1.1 System Interfaces

**Machine Learning API:** The API will serve as the bridge between the front-end and the ML model, which will be hosted on an external cloud. It will handle sending processed camera feed, receiving information about obstacles, and relaying it back to the front-end.

**Google API:** The Google API will be used to access current weather data to inform the user once the application is activated.

#### 1.5.1.2 User interfaces

- **Weather Condition Alert:** The application's weather condition alert shall include information about current weather conditions in the user's vicinity, such as temperature and general condition. The alert will be sent to the user's phone's screen reader to be relayed audibly.
- **Main Page:** The application's main page shall include a prompt for the user to begin object detection, after which the page shall send audio alerts of the objects detected.
- **Introductory Page:** The application shall feature a help page that informs the user on the usage of the application and provides a short introduction the first time the application is opened. It shall provide a brief explanation of the app's available functions and how to interact with them.

### 1.5.2 Product functions

- The software shall inform the user of the current weather condition once navigation mode is activated.
- The software shall capture image snapshots from the user's phone camera and transmit said images to the back-end, where the ML model will analyze individual images for road hazards and obstacles.
- If it detects any, the back-end shall relay information regarding the obstacle, its estimated distance, and estimated relative direction from the user to the front-end, which will then alert the user using automated Text-To-Speech.

- The software shall request access to the user's camera and geolocation information.

#### 1.5.3 Identified stakeholders and design concerns

- Weather API Provider: Their weather API will be used to fetch weather information.
- Cloud Service Provider: The back-end and ML model will be hosted on their servers.
- Visually Impaired Application Users: Their camera feed will be accessed by the application and fed into the back-end.
- Bystanders: might show up in snapshots, images of them will not be stored.

#### 1.5.4 User characteristics

- Users will generally be blind or severely visually impaired
- Users are not expected to be technically adept
- Users are expected to walk slower than the average person, we assume that people with visual impairments walk much slower than people with no severe visual problems
- Users will not suffer from major hearing disabilities; this is essential for the use of the application as all alerts are auditory

#### 1.5.5 Limitations

- Due to the size of the ML model, the backend cannot be stored on the mobile phone
- The phone's camera is essential to the mobile app's functionality, permission to access the camera's feed must be requested from the user
- As the ML model will be hosted remotely, the mobile app must have constant access to the internet as long as it is in operation

#### 1.5.6 Assumptions and dependencies

- It is assumed that users will have either an IOS or an Android phone, this will allow us to focus on main-line operating systems and avoid having to deal with compatibility issues.
- It is assumed that users will have sustainable internet connections as long as the application is in use. Since the ML model will be stored remotely, the application cannot be used without an active internet connection.
- It is assumed that users will walk relatively slowly while using the app, blind people walk relatively slower than the average person and this gives us more room for processing time
- • It is assumed that the app will not operate in exotic locations (i.e., forests, caves, deserts, etc....), this will allow us to improve the performance of the ML model as it won't need to process extra classes of objects that are extremely rare to come by in an urban environment.

## 2 Specific requirements

### 2.1 External interfaces

| System Interface | Functionality                       | Input                   | Output  |
|------------------|-------------------------------------|-------------------------|---|
| Weather API      | Fetch Weather Information           | Geolocation Information | City Weather Conditions                                       |
| Cloud Service    | Host ML Model and Process API Calls | Images/Frames           | Detected Objects with their Estimated Distances and Positions |

Table 2: External Interfaces

## 2.2 Functions

- Object Detection
  - The system shall warn the user about nearby obstacles while navigation mode is active.
    - The system will alert users with the name of the object.
    - The system will notify the users audibly of the distance and position to the object.
- Weather Conditions Information
  - The system shall inform the user about weather conditions once navigation mode is activated.

|  |  |
|--|--|
| Use Case   | Object Detection   |
| Actors   | User, ML Model   |
| Cross References   |  |
| Actor Intentions   | System Responsibility  |
| 1. User Starts Navigation Mode   |  |
|  | 2. The system sends frames to the ML model to detect objects and calculate distances and positions |
| 3. The ML Model processes the frames and extracts detected objects                   |  |
| 4. The ML Model calculates the distance and position of each of the detected objects |  |
| 5. The ML Model sends the processed information to the system                        |  |
|  | 6. The system receives information about detected objects from the ML model                        |
|  | 7. The system relays information to user via audio alerts  |

Table 3: Object Detection Use Case

|                                |  |
|--------------------------------|--|
| Use Case                       | Weather Conditions Information   |
| Actors                         | User, Weather API  |
| Cross References               |  |
| Actor Intentions               | System Responsibility  |
| 1. User Starts Navigation Mode |  |
|                                | 2. The system collects geolocation information from the user's phone and sends an API request to the Weather API |

|  |  |
|--|--|
| 3. The Weather API responds with information based on the given geoinformation |  |
|  | 4. The system receives weather information                                     |
|  | 5. The system displays weather information pop-up and informs the user audibly |

*Table 4:Weather Conditions Information Use Case*

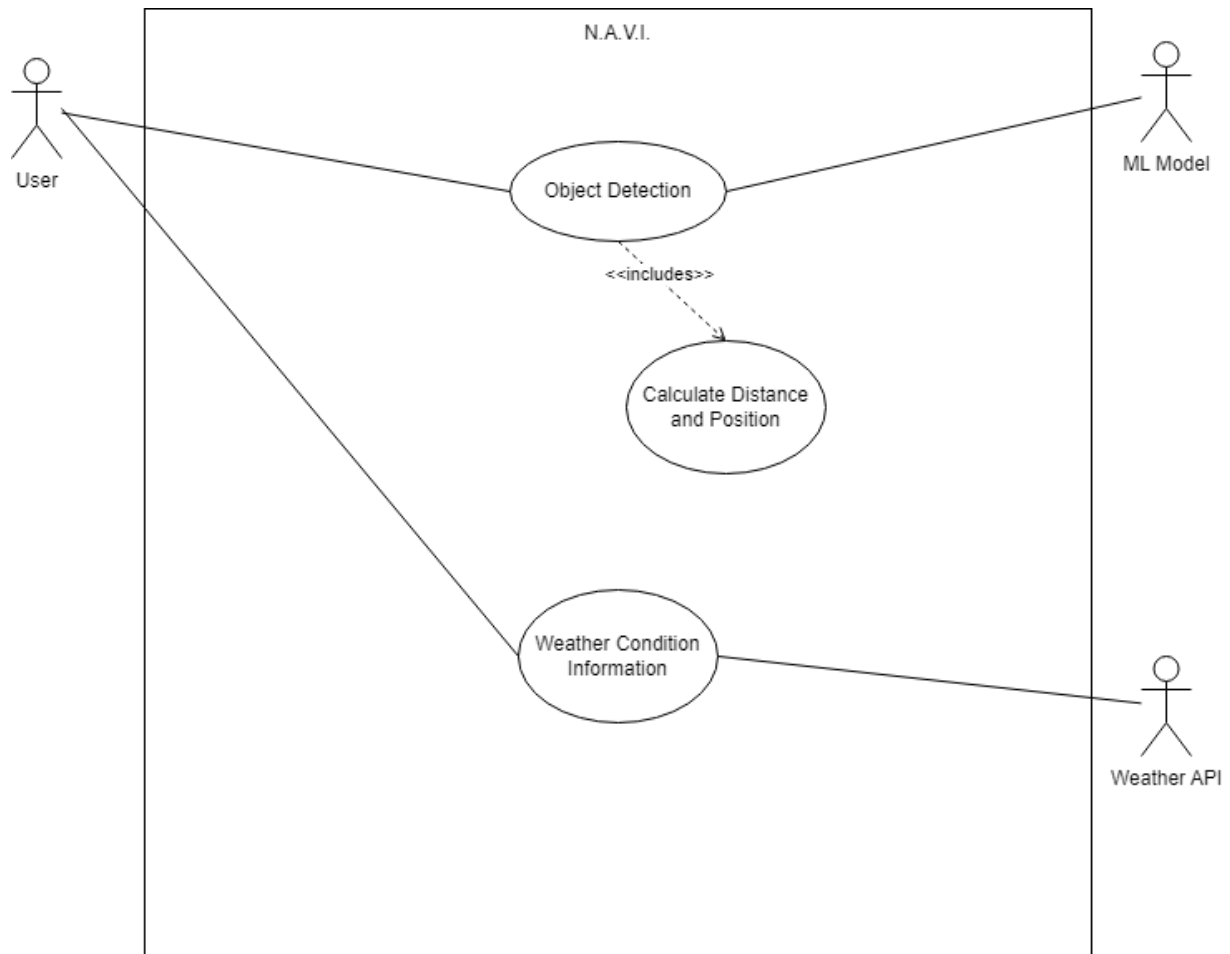


Figure 7: Use Case Diagram

### 2.3 Usability Requirements

- The components of the app interface (such as the main and the video permission page) will be simple to comprehend.
- A help page that explains how to use the application will be available.
- Simple and understandable warning messages will be provided during object identification, no complex language will be used and messages will be kept short.

### 2.4 Performance requirements

- The images are expected to be minimum 0.4 MP resolution because YOLOv8 can process 608\*608 pixel size (Team, 2021).
- A navigation server shall be able to manage connections from several client devices. Giving an estimation of the number of clients is not possible for now because that is dependent on the cloud service that is to be used and the cloud services have not been examined yet.
- The performance shall depend upon the hardware components of the mobile devices.
- Object capture, sent data size, connection speed, and processing speed all be crucial to achieve acceptable communication delay between the ML model and the app.

## 2.5 Logical database requirements

The system will not store any information about users or any other distinct entities, so a logical database is not needed for the purposes of the application.

## 2.6 Software system attributes

- Reliability
  - If any system component fails to function properly, an error message should be displayed to users.
  - The application's dependability is primarily determined by the software tools (Flutter, Text-to-Speech) and hardware tools (camera, sensor, lens, etc.) used in system development. The camera, sensor, and other devices are expected to function flawlessly.
- Availability
  - The application shall be available in only bright places which have enough light.
  - The system shall be accessible for only Android/iOS devices.
  - The system shall support English as a usage language.
  - Users of the product are expected to be blind or visually incapable, so the GUI of the mobile application shall be simple and distinct as well as screen reader friendly.
- Security
  - The system shall be accessible for only developers; users are not able to edit or modify any part of the mobile application or ML model.
  - Application shall not access user's cameras unless the user provides permissions
- Maintainability
  - Thanks to the cloud, apart from the API we will use, different APIs can be added to develop some functionalities, so this application will be always open to update by developers.
- Portability
  - The application shall be written with Flutter for design part, In the ML part include a python code to implement functionality. Flutter is a portable UI toolkit that allows you to create native-like apps for mobile, web, and desktop from a single codebase (Flutter: the framework for cross-platform applications, 2021). Python also is designed to be portable. Its programs are supported on any modern computer OS (Python vs other programming languages, 2022).

## 2.7 Supporting information

In this section, the machine-learning model that we are planning to use will be discussed. In brief, there are two main functionalities of our application. Firstly, the application will let the visually impaired know what objects they are facing e.g. car, bench, bicycle. Secondly, the application will let the visually impaired know how far and at what angle these objects are with respect to the user. The object detection will be done with the help of the machine learning model. After doing extensive research, please refer to the Literature Review subsection, we decided that choosing an ML architecture from the YOLO family would be a wise decision. The usual object detection models have two stages. In the first stage, it detects regions that possibly can have objects. In the second stage, it performs the classification. In other words, in the second stage, it figures out the class that the object belongs to. Examples of architectures that are implemented such would be Fast-RCNN (Girshick), Faster-RCNN (Shaoqing Ren, 2016), etc. These kind of models are called Region Proposal Networks since they propose the regions that

have the potential of containing an object and then perform classification. On the other hand, there are also architectures that perform region proposing, which is called bounding box prediction, and classification simultaneously. The YOLO family (Sharma, 2022) is an example of this type of architecture. This model embraces an end-to-end architecture that gives it the speed it needs; it can process 45 frames per second. Hence, it is able to achieve a state-of-the-art result for real-time object detection. Now, let's dive deeper into the YOLO family. How does it perform localization? It basically uses the Intersection over Union metric to measure the localization accuracy. When training images are given as an input to the model, in addition to the pixel values of the image, the specific region, or it can also be called the bounding box, is also given as x and y coordinates along with the value of width and height. Hence, the model is aware of the ground truth regarding the bounding box region. As it can be seen in Figure 8, a bounding box is being drawn around an image that belongs to the class drowsiness to make it ready to be supplied to the model. The intersection region is the place where the model's prediction of the object's location overlaps with the ground truth of where the object is as can be seen in Figure 9. The higher the intersection the better because that implies that the prediction of the model is very close to where the object really is. Whereas, the union is the overall region that encompasses both the predicted area and the ground truth. The division between the overlapping area and the union area is the value of IOU. The value of this metric can maximally be 1 and that is the case where the predicted bounding box is exactly over the area where the object of interest is. In other words, that is exactly what we want our model to achieve regarding the localization.

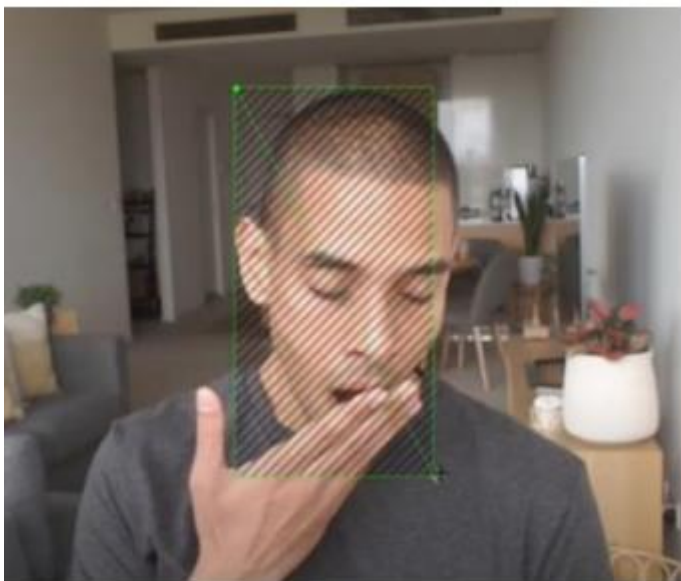


Figure 8: Bounding Box Creation for the Image While Labelling



Figure 9: Bounding Boxes Intersection

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Now, we know that YOLO will be responsible for object detection, but what about distance estimation? For that we have two options. We can either try to find a model that can perform both the object detection and distance estimation together, combine the two seemingly different tasks into one, or let the machine/deep learning model to only take care of object detection and use physics as an additional step for distance estimation. We will come into explaining the distance estimation with physics after investigating the model that can handle both object detection and distance estimation.

Dist-YOLO (Marek Vajgl, 2022) is an ML architecture that extends the functionality of YOLO to include distance estimation as well. In YOLOv3, the prediction vector, which is the output of the ML model, gives  $p = (b, c, o)$ .

$b \Rightarrow$  coordinates produced by the bounding box ( $x, y, w, h$ )

$c \Rightarrow$  confidence of belonging to a certain class  $c = (c_1, c_2, c_3, \dots, c_n)$

$o \Rightarrow$  expresses objectness; i.e, confidence that  $p$  captures an object

In addition to the already present parameters in the prediction vector, Dist-YOLO architecture adds the distance parameter to it as well. Consequently, the loss function also needs to be updated to include the loss due to miscalculation of the distance so that the model can optimize itself based on the distance parameter as well. The Dist-YOLOv3 architecture which is an extension of YOLOv3 (Joseph Redmon) architecture can be seen in Figure 10.

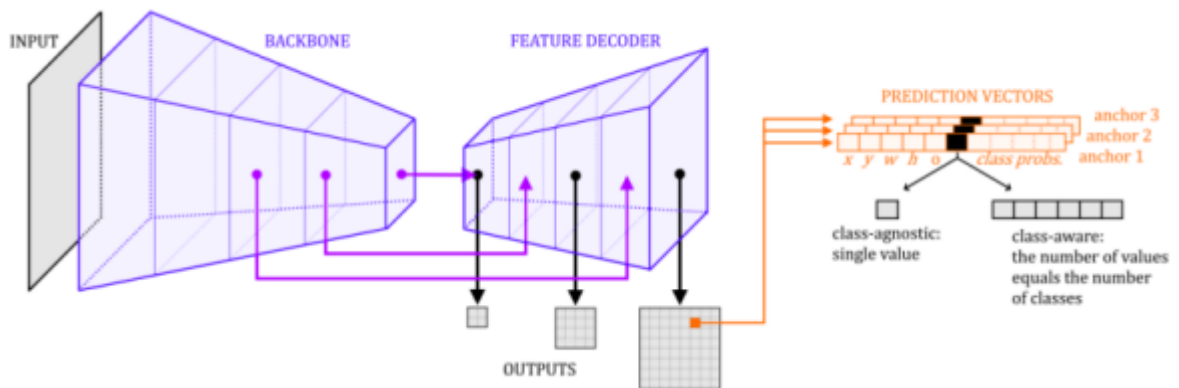


Figure 10: Dist-YOLOv3 Architecture (Marek Vajgl, 2022)

Here comes the interesting part, what was it that inspired the designers of Dist-YOLOv3? Two main principles were driving them. Firstly, it was Multi-Task learning (Taylor Mordan) and secondly it was Curriculum Learning (Yoshua Bengio). Multi-Task learning implies that learning two or more seemingly different concepts or tasks together or more accurately in the same architecture with one another can help the model perform better than if it was designed for only one of the tasks. In our case, the designers of Dist-YOLOv3 believed that in addition to localizing objects, training the model to also learn distance estimation could in fact improve the overall performance of the model. It might appear hard to accept this fact because we see the task of distance estimation as another completely different responsibility hence its addition to the model would imply more complexity and more computational power. However, that wasn't the case. After testing out the performance of the model before and after the addition of the distance calculation task, it was seen that the model performed better with the addition of distance calculation task. That might be because learning distance calculation might help the model figure some similarities between the localization and distance calculation task and that similarity

couldn't have been found if it wasn't the relationship between the "seemingly" different tasks. Think of it this way: while learning how to play guitar, the player's ear gets sensitive to different frequencies and tones and they eventually become familiar with concepts such as intensity, frequency, period, wave, etc. Consequently, once they learn these concepts in their physics class, even though they might be used differently, the learner will not find it very challenging to internalize and familiarize themselves with these concepts. They might even find correlations between the two fields, namely music and physics, in order to better understand the concepts that they share: frequency, wavelength, intensity.

What about Curriculum Learning? That is about making the model learn in a planned and organized manner rather than randomly and this approach was brought up by observing how students learn better when they follow a curriculum that was designed carefully and thoughtfully. In school, first students start learning basic mathematical operations, then they get acquainted with equations, and slowly they dive into harder topics because the easy ones behave like building blocks so that harder topics can be built on top of them. Same concept applies to Dist-YOLOv3 model. In order to teach the model about object detection, we utilize the data-augmentation technique, which is resizing the object, rotating, etc. Basically, data augmentation helps to increase the amount of data we have and helps the model get familiar with different orientations and sizes of an object, hence it helps improve the performance of the model for object detection. However, it has an adverse effect if used with distance estimation because the size, which is one of the parameters affected by data-augmentation, of the object affects how far it is from us. So, what do we do? It affects one of the tasks positively and the other negatively? It would have been great if we could use it with one and not the other right? Here, comes the role of curriculum learning; first teach your model how to detect objects and meanwhile freeze the weights that are responsibly from distance prediction. By freezing, it means the part of the model that learns distance estimation is in sleep mode; the weights don't get affected by gradient descent. Consequently, we can utilize data-augmentation without worrying that it will have an adverse effect on the model's performance in distance estimation because the model isn't learning that task while data-augmentation is being used. Later, once the model learns object detection, activate the weights responsible from distance estimation and stop the data-augmentation. Now, since the weights responsible from distance estimation have woken up and can actually learn, we stopped data-augmentation so they won't be adversely affected. Voila, we got the benefit of dataaugmentation and avoided its adverse effect thanks to curriculum learning.

All that was due to us willing to integrate the distance estimation task with object detection into one ML model because of how it exploits and utilizes the beautiful and new-born concepts in machine learning. However, if we couldn't use Dist-YOLOv3 model due to either it having some advanced concepts or us realizing that it has properties that don't align smoothly with 19 our project's functionalities, then we will have to detach the two tasks, namely object detection and distance estimation, from one another and use the YOLO model for object detection and some other method for distance estimation. A method that we currently have in mind is the triangle similarity method, (Rosebrock, 2021). Firstly, we are required to find the focal length  $F$  of the camera. To find  $F$ , we first need to take the picture of an object by knowing how far it is from the camera. Now, we will be knowing the value of  $D$  and  $W$ . We also need to know the  $P$  value that is the Perceived width, which is the pixels appearing in the camera. Finally, by using the formula,  $F = (P \times D) / W$ , we can calculate the value of the focal length of the camera. By rearranging the formula, we can calculate the distance,  $D = F * (W/P)$ .

As it was previously mentioned, the model outputs  $p = (b, c, o)$  and the “b”, bounding box, parameter includes the perceived width of the object. This is where we get the needed P value from. The actual 17 width of all the object classes can be stored in the program and retrieved whenever needed. Moreover, we already calculated the F value. At last, we are ready to calculate the distance.

It is worth mentioning that we had an interview with a visually impaired person. The interview was intended to help us see our project from an actual user's perspective and help us figure out how we can improve our project's functionalities. After having the interview, we realized that only giving distance information of the detected object wasn't really helpful to the visually impaired user because the object might be in various angles with respect to the user. For example, an object that is 2.5 meters away from the user could be 45 degrees to the left of the user or 45 degrees to the right of the user and that makes a significant change in perception. Consequently, we decided to add angle calculation as another functionality of our mobile application. However, since we decided to add the angle calculation functionality recently, we didn't have enough time to investigate it in detail. For that reason, the SRS report might not be very explanatory with respect to this functionality. We believe that this shouldn't arise a problem because we are following the Agile Development principle that understands and accepts that user requirements can change and affect the development stage.

### 3 Software Estimation

Inputs:

| Inputs               | Complexity | Multiplier |
|----------------------|------------|------------|
| Detected Object Data | High       | 8          |
| Weather Data         | Medium     | 6          |

Outputs:

| Inputs              | Complexity | Multiplier |
|---------------------|------------|------------|
| Directions          | High       | 8          |
| Object Detection    | High       | 8          |
| Weather Information | Low        | 3          |

Inquiries:

| Inputs                      | Complexity | Multiplier |
|-----------------------------|------------|------------|
| Send Images/Frames to Cloud | High       | 8          |
| Send Geo-info to Cloud      | Medium     | 5          |

Logical Internal Files:

| Inputs                     | Complexity | Multiplier |
|----------------------------|------------|------------|
| Distance and Position Data | Medium     | 5          |

External Internal Files:

| Inputs   | Complexity | Multiplier |
|----------|------------|------------|
| ML Model | High       | 8          |

Unadjusted Total of Function Points: 18

Inputs: 14

Outputs: 19

Inquiries: 13

Logical Internal Files: 5

External Internal Files: 8

Total UTFP: 59

Influence Multiplier:

|    | General System Characteristics   | Degree of Influence |
|----|----------------------------------|---------------------|
| 1  | Distributed Functions            | 3                   |
| 2  | Performance                      | 3                   |
| 3  | Heavily Used Configurations      | 2                   |
| 4  | Transaction Rate                 | 4                   |
| 5  | On-line Data Entry               | 3                   |
| 6  | End-user Efficiency              | 4                   |
| 7  | On-line Updates                  | 2                   |
| 8  | Complex Processing               | 4                   |
| 9  | Reusability                      | 2                   |
| 10 | Installation Ease                | 4                   |
| 11 | Operations Ease                  | 5                   |
| 12 | Data Communications              | 5                   |
| 13 | Multiple Sites                   | 0                   |
| 14 | Facilitate Change                | 1                   |
|    | Total Degrees of Influence (TDI) | 42                  |

$$VAF = TDI * 0.01 + 0.65 = 42 * 0.01 + 0.65 = 1.07$$

$$ATFP = UTFP * VAF = 59 * 1.07 = 63.13$$

$$LOC = ATFP * \text{Unit Size}$$

Assume Unit Size of 128, as flutter is based on C.

$$LOC = 63.13 * 128 = 8080.64$$

Effort Estimation with COCOMO Model:

while our team is small, we face strict deadlines and our development environment is unstable due to the inclusion of other courses. So, the semi-detached model is the most suitable compromise.

$$a = 3.0, b = 1.05, c = 0.38$$

$$KDSI = LOC / 1000 = 8.08064$$

Effort in Man-Months =  $a * KDSI^b = 3 * 8.08064^{1.05} = 26.9$  Man-Months

Development Time =  $2.5 * MM^c = 2.5 * 26.9^{0.38} = 8.73$  months

Jones's First-Order Effort Estimation:

19 Our team is largely inexperienced when it comes to organizational skills, adding to that the fact our team has lost a member and our planning was generally ad hoc, we will use the worst in class exponent, we also assume a shrink-wrap software as our application is self-contained and readily available on mobile devices.

Estimated Effort =  $(ATFP^{(3 * \text{Class Exponent})}) / 27 = (63.13^{(3 * 0.45)}) / 27 = 9.97 \approx 10$  man-months

## 4 Architectural Views

The architecture of software systems will be described based on the Logical View, Process View, Development View, and Deployment View, as suggested by Kruchten (1995)<sup>1</sup>

### 4.1 Logical View

#### 4.1.1 Class Diagram N/A

Class diagrams are used when developing an object-oriented design model. However, since we have a research-oriented project that doesn't follow an object-oriented design, a class diagram is not available.

However, we have data structures present in our project. For example, there is a dictionary that holds class names and their actual width, in centimeters, in real life: {'car': 448, 'bicycle': 175, 'couch': 152, etc.}. This dictionary will be utilized while calculating the distances and angles of the detected objects from the user.

Moreover, image compression might also be needed while sending frames to the external server in order to reduce the data traffic. Image compression will be needed to be performed in the frontend, hence we will have matrices, 3 dimensional arrays in case of RGB images, as another data structure to be utilized. Figure 11 is an example of how an image is represented when prepared for processing for the ML model. Each array element represents a pixel of an image in the respective color channel.



Figure 11: Three-Dimensional Array Representation of an Image

<sup>1</sup> P. B. Kruchten, "The 4+1 View Model of architecture," in IEEE Software, vol. 12, no. 6, pp. 42-50, Nov. 1995, doi: 10.1109/52.469759.

## 4.2 Process View

The process view shows how the system is composed of interacting processes.

### 4.2.1 Activity Diagram

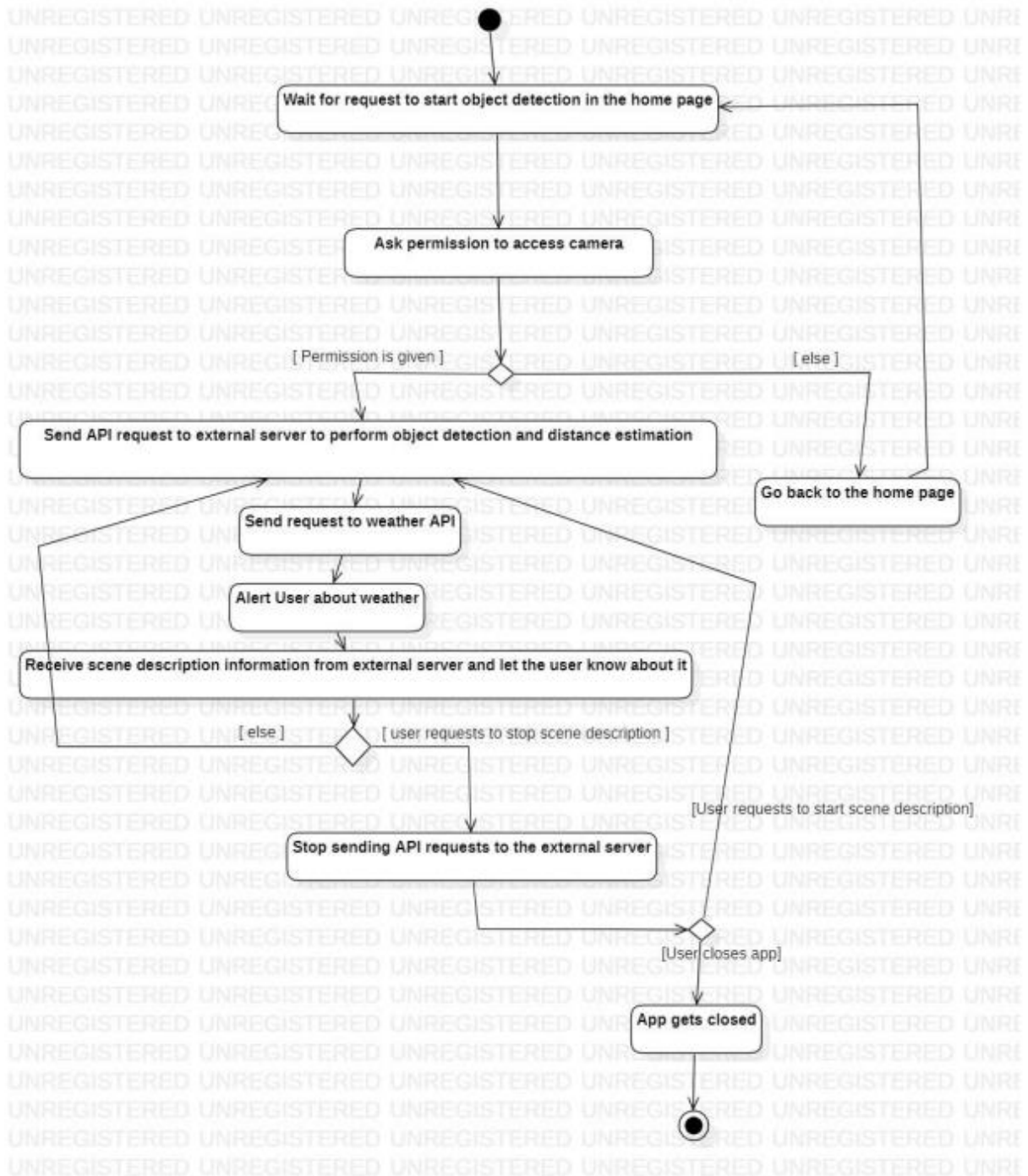


Figure 12: Activity Diagram

The core functionalities of the application will start executing once the user requests it. Permission to access the camera will follow the request because the functionalities need frame data from the camera in order to execute. If permission is not given, the user will be forwarded to the home page where they can request for the core functionalities to start again and hence permission will be asked once again.

If the user gives permission to access the camera, an API request will be sent to the external server to perform object detection, distance estimation, and angle estimation. The API requests will be sent periodically in a loop until the user requests to stop the process. The number of API requests to be made per unit of time has not been decided on yet. The decision will be taken after performing some experimentation, which can be done after a prototype gets ready, and research.

It is worth mentioning that we had an interview with a visually impaired person. The interview was intended to help us see our project from an actual user's perspective and help us figure out how we can improve our project's functionalities. After having the interview, we realized that only giving distance information of the detected object was not helpful to the visually impaired user because the object might be in various angles with respect to the user. For example, an object that is 2.5 meters away from the user could be 45 degrees to the left of the user or 45 degrees to the right of the user and that makes a significant change in perception. Consequently, we decided to add angle calculation as another functionality of our mobile application.

#### 4.2.2 Sequence Diagrams

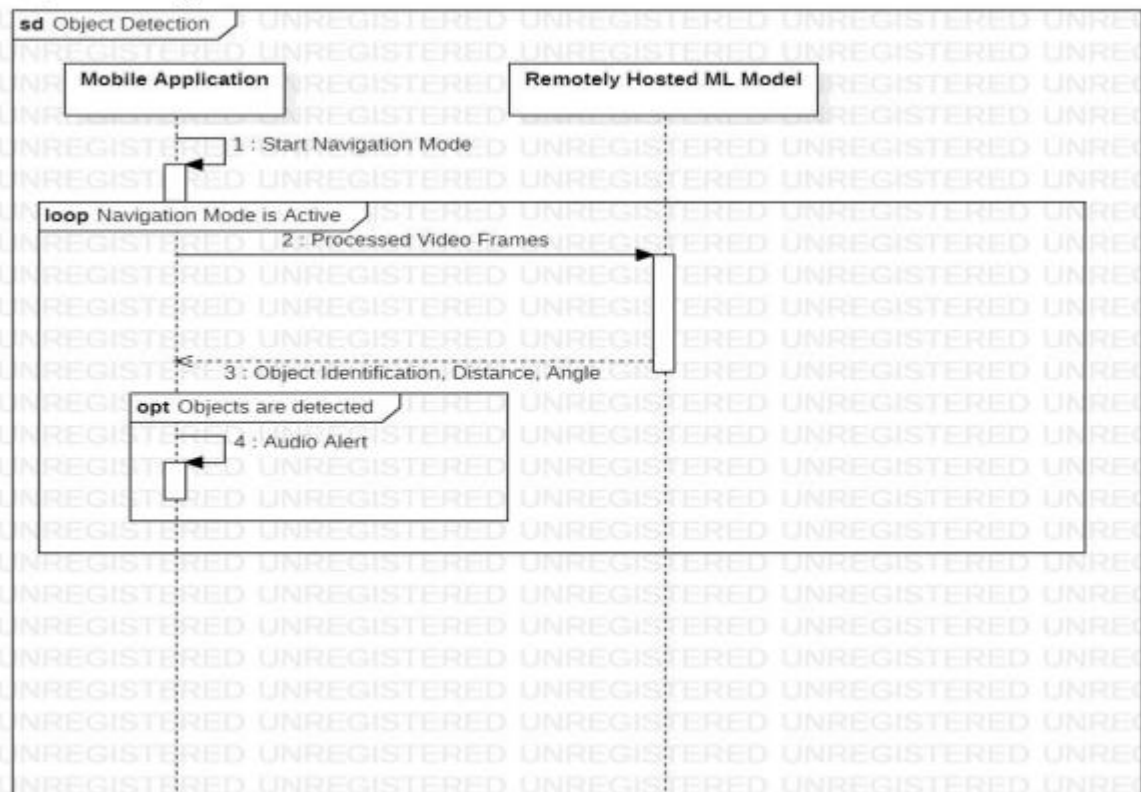


Figure 13: Object Detection Use Case

The sequence diagram above displays how the software carries out object detection as well as its communication with the remote ML model. Operation starts once the user activates navigation, after which the front-end handles capturing and processing video frames and sending them to the ML model. The model then extracts objects from the frame, calculates their distances and relative 22 positions on a 3x3 grid, then sends the identified objects and their information to the front-end. If objects are detected, the front-end alerts the user about the objects and their information via audio.



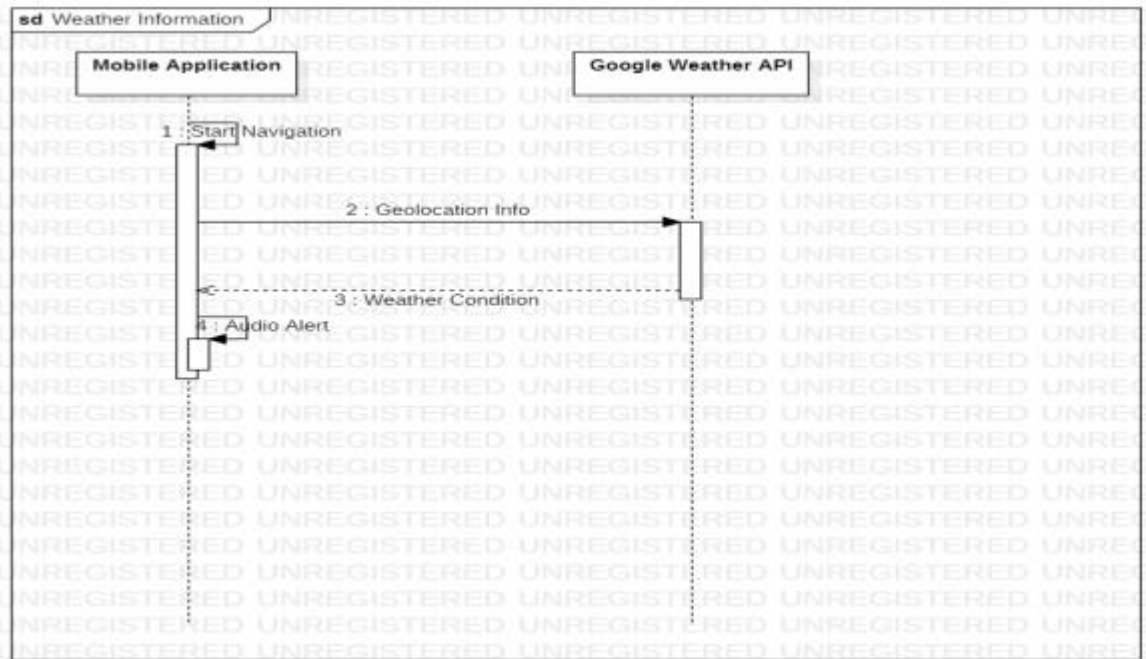


Figure 14: Weather Information Use Case

The sequence diagram above details how the software handles the weather information use case. Once the user starts navigation, the application makes an API call to Google’s weather API using geolocation info collected from the user’s phone, the front-end then processes the reply and gives the user an audio alert about the current weather conditions.

#### 4.2.3 Data Flow Diagrams

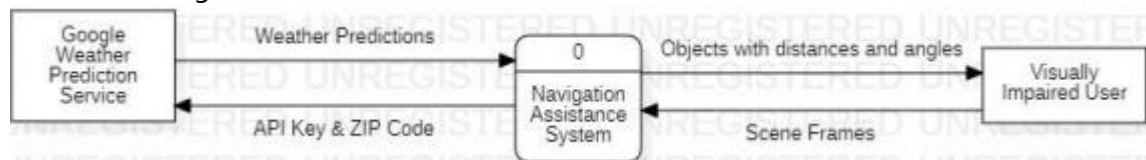


Figure 15: Dataflow context-level

The data flow diagram for our system when considered at the context-level can be seen in Figure 15. At this level, it was aimed to show the system boundaries and external entities. The first external entity is the Google Weather Prediction Service meaning that we will be using an API provided by Google to get weather information and provide it to the other external entity, which is the Visually Impaired User. To use the API, the internal system will need to provide an API key and ZIP code.

As for the other external entity, namely the visually impaired user, the internal system will provide scene descriptions to it, which consist of the objects and their distances and angles from the user. The internal system needs frames that will be provided by the Visually Impaired User so that in return it can provide scene descriptions.

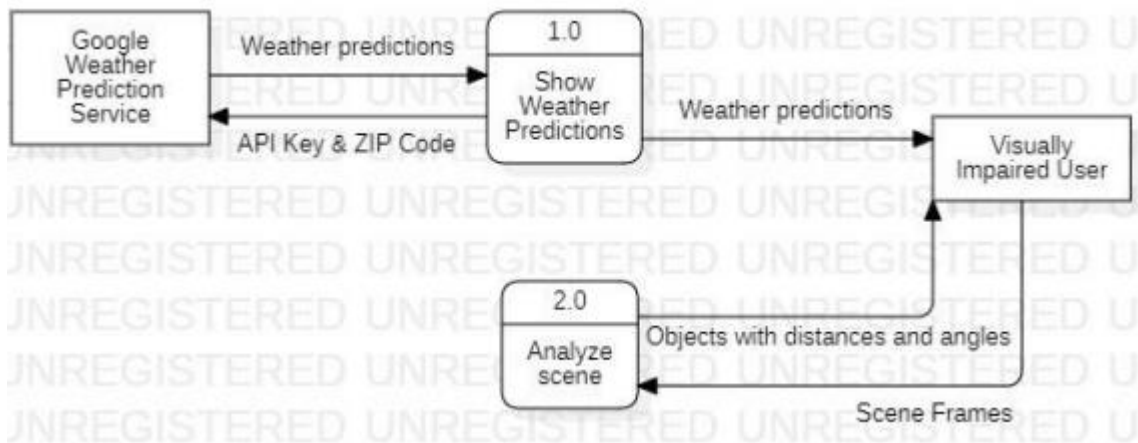


Figure 16: Dataflow level – 0

The data flow diagram for our system when considered in level-0 can be seen in Figure 16 where the system's major processes are displayed. The 1.0 process uses the Google Weather Prediction Service to provide the Visually Impaired User with weather prediction information. The 1.0 process is located inside the frontend of the mobile application. In addition, the 2.0 process will perform scene analysis, which will be done on the external server. The process needs scene frames coming from the visually impaired user and will process the data to return objects with distances and angles for the sake of scene description to the visually impaired user. Further information about distance and angle calculations will be given in Dataflow level-1. Note that even though it is called an external server, it is still within the system boundaries but located elsewhere. The system boundary includes the application's backend, which is hosted in the external server, and the application's frontend, which is hosted in the mobile.

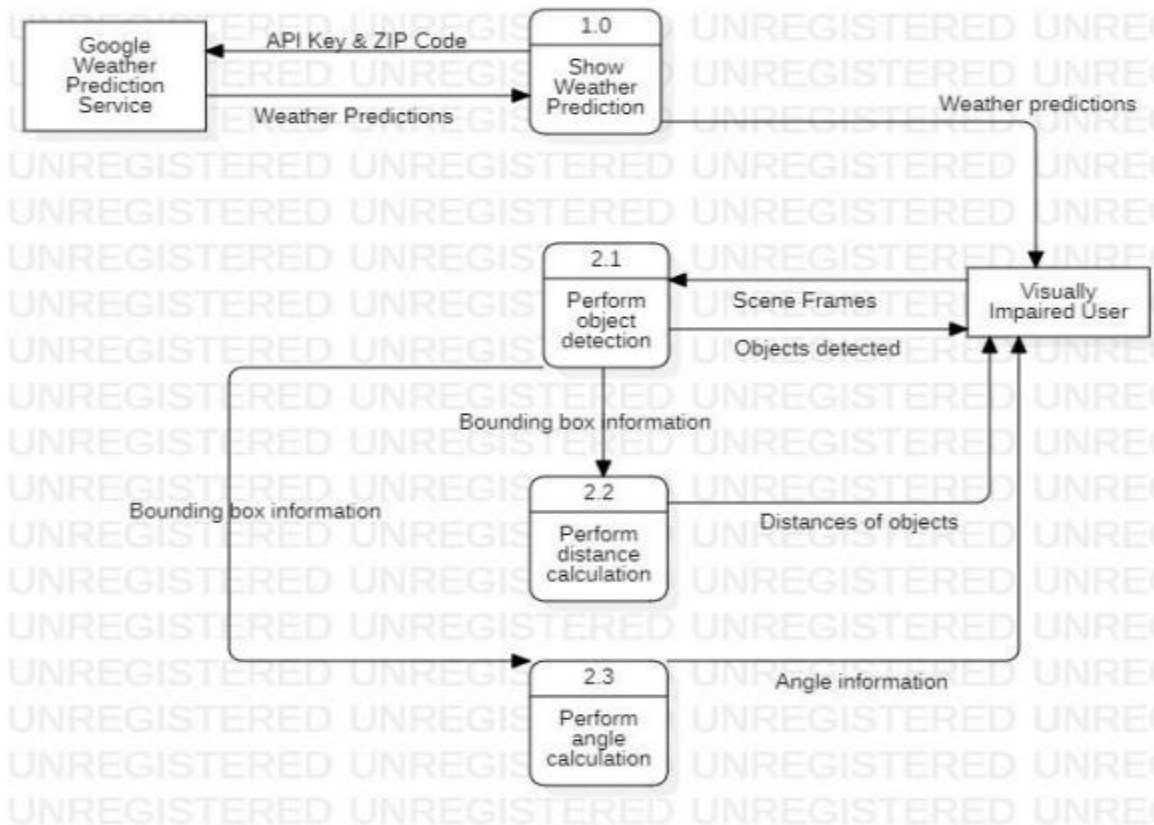


Figure 17: Dataflow level – 1

The data flow diagram for our system when considered in level-1 can be seen in Figure 17 where the sub-processes of the major processes are displayed. The “Analyze scene” process 2.0 in level-0 is divided into three sub-processes namely, Perform object detection, Perform angle calculation, and Perform distance calculation.

As previously mentioned, the distance calculation method requires the perceived width of the detected objects, hence the flow of the Bounding box information from the Perform object detection process to the Perform distance calculation process can be seen. Likewise, Bounding box information also needs to get to the Perform angle calculation process. Note that the bounding box, which is one of the outputs of the ML model, contains information about the coordinates of the location of the detected objects and their width & height, which will be used in distance calculation. Process 2.1 is performed by the ML model, whereas processes 2.2 and 2.3 use optical physics algorithms hence they are separated into three sub-processes of the major process Analyze scene.

Additionally, it is important to mention that the Show Weather Prediction process has not been separated into various parts in the dataflow level - 1 because it can't be decomposed into further processes.

### 4.3 Development View

The development view shows how the software is decomposed for development.

#### 4.3.1 Component Diagram

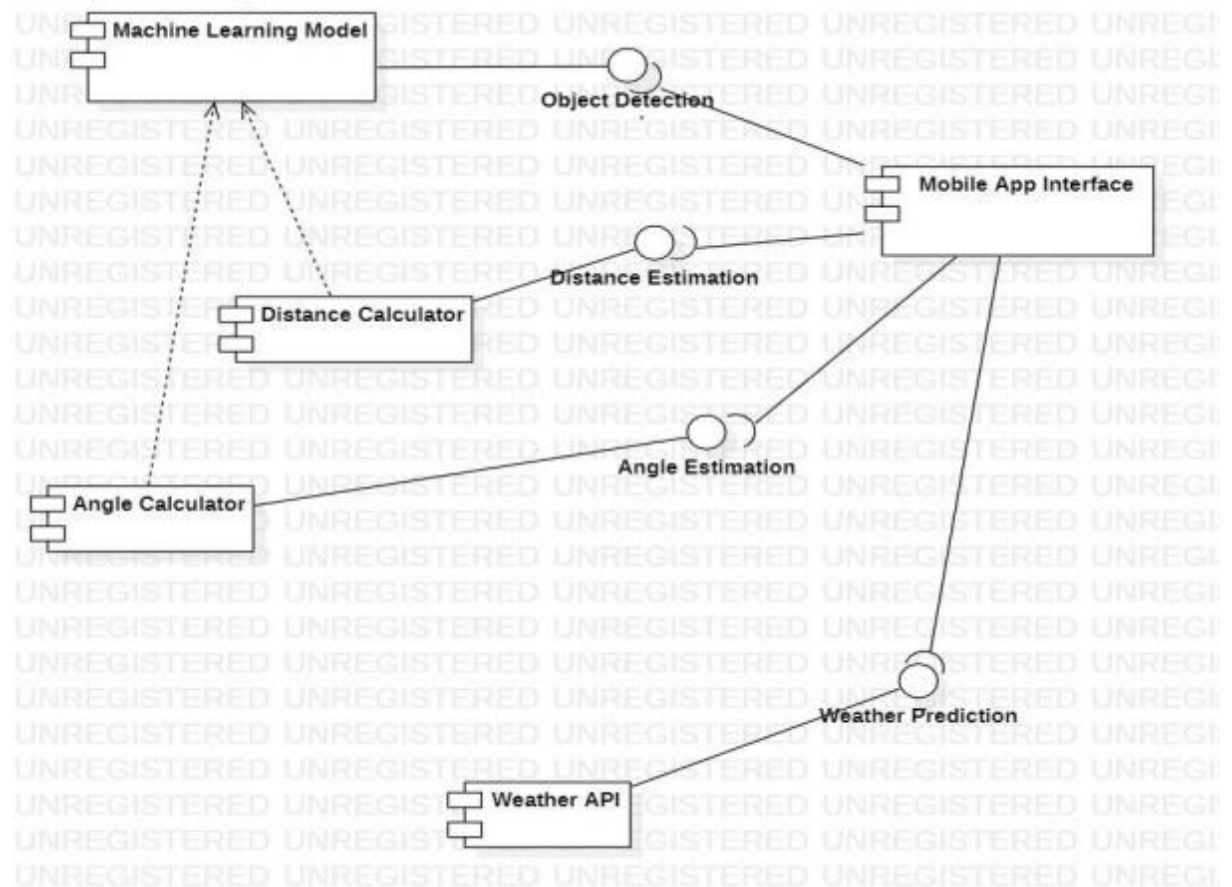


Figure 18: Component Diagram

The scene in front of the visually impaired, which will be considered as a frame, will be described by three separate components. The object detection will be performed by the ML model, whereas the distance estimation and angle estimation will be performed by the Distance Calculator and Angle Calculator respectively that will be using some optical physics equations/algorithms. The Distance Calculator and Angle Calculator are dependent on the ML model because the algorithms to be used for distance estimation and angle estimation need to know the perceived width, which can be found in the bounding box of the detected object. Note that we can obtain the bounding box of the detected object because the ML model to be used is for object detection (localization) and not only object classification

In brief, the data to be used for scene description will be coming from three components and needs to be combined to give meaningful information. They will be combined in the Mobile App Interface. For example, the ML model will give data such as cars, bicycles, and trash-can, the Distance Calculator will give data such as 5 meters, and 4 meters and the Angle Calculator will give data such as 45 degrees to the left, and 60 degrees to the right, finally, these three separated data will be combined to convey meaningful and useful information such as a car in 5 meters away, 45 degrees to the left of the user there is a bicycle.

Finally, the Weather API component will provide the Weather Prediction Interface, which is part of the frontend, with weather predictions.

#### 4.4 Physical View

The physical view shows the mapping of software onto hardware.

##### 4.4.1 Deployment Diagram

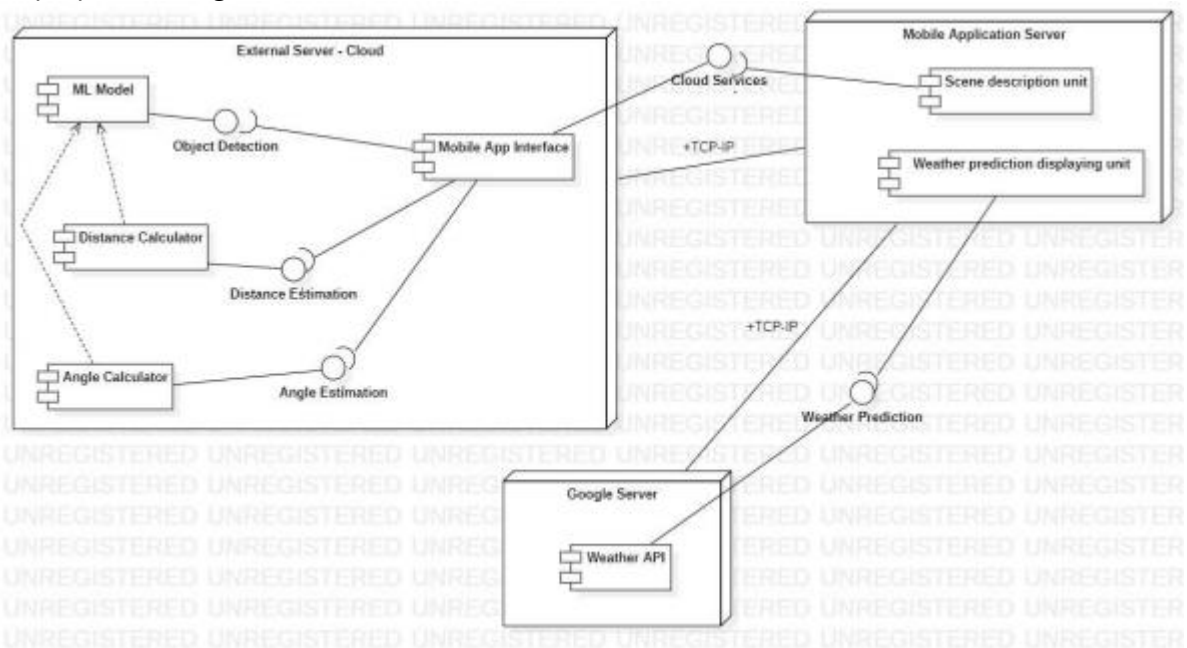


Figure 19: Deployment Diagram

There are three main servers being used in our project. Namely, they are the external server, Google Server, and mobile application server. These servers will be communicating over the Internet that uses the TCP/IP protocol.

- External Server: This is the most functionally and computationally dense server because it will be hosting the ML model and the distance calculation method to perform object detection and distance calculation respectively. The responses from the ML Model, distance calculation, and angle calculation method will be combined together and fed into the Mobile App interface because they provide information that is related to one another. Moreover, the distance calculation and angle calculation methods depend on the ML model because the model will be providing one of the function parameters, the width of the object, to the methods. The model is able to provide the width data to the distance calculation method because YOLOv5 also predicts the bounding boxes of the detected object.
- Google Server: It should be made clear that only one of the functionalities of the Google Server which is weather prediction will be used in our project. Moreover, the Weather Prediction functionality will be accessed using an API provided by Google. We are using Google's weather prediction service because this is not one of the main functionalities of our project, hence, we did not find it necessary to reinvent the wheel since there is already an available trustworthy implementation.
- Mobile Application Server: This is the server that will be hosting the frontend. Consequently, the mobile app is basically a thin client [1] that communicates with the resource-rich servers to provide services such as object detection, distance calculation, and weather prediction to the end users of N.A.V.I. There are external servers used because the end-user has no easy access to servers that can perform dense computations, instead, the end-user will be utilizing a thin client, a mobile app that communicates with external servers, to access services. The thin client is divided into two parts, namely Weather Prediction Displaying Unit and the Scene Description Unit. Weather Prediction Displaying Unit will communicate with the Google Weather API and the Scene Description Unit will communicate with the external server.

## 5 Software Implementation

- The back-end was implemented in Python.
- The API was built using Flask, a lightweight Python web framework that provides useful tools and features for creating web applications in the Python.
- Pre-trained YOLOv8, which is the core of our application, was loaded and prepared using the ultralytics library, which is a Python library made by YOLO developers in as a sort of a catch-all solution to accessing and using their projects.
- API testing was done via the Locust library, which is a Python-based testing tool used for load testing and user behavior simulation.
- Construction of the dist-yolo model was done using the torch library, a Python open source ML library used for creating deep neural networks and is written in the Lua scripting language.
- Data Pre-processing on the images was done via Albumentations, Albumentations is a Python library for image augmentation.
- The front-end was implemented in Flutter, an open-source UI software development kit created by Google for developing mobile applications for IOS, Android, and Linux.

- The front-end was coded in the Dart programming language, The programming language can be used to develop web and mobile apps as well as server and desktop applications. It is an object-oriented, class-based, garbage-collected language with C-style syntax.
- Various Dart and Flutter packages were used that each serve a specific purpose: camera access, image compression, testing, etc...

## 6 Software Testing

Testing the system was divided into testing the front-end via Unit Testing and Integration Testing, these were tested using flutter's built-in testing libraries. The back-end was used for testing the accuracy and performance of the ML model, as well as the two chosen characteristics: stress testing and performance testing, both of which were done on the API in conjunction with the backend, this was done using the Locust framework. Finally, the back-end and front-end were combined together for system testing, which was done manually. The following items summarize the focus of our tests:

- User Interface/Front-end, this consists of the mobile app itself, which will undergo unit testing as well as integration and system testing.
- Machine Learning Model/Back-end, this consists of the machine learning model and the encapsulating code that exposes an endpoint, it will be subject to stress testing, and performance testing.
- API, the link between the front-end and back-end, it handles passing images to the backend and returning detected object to the front-end. The API we developed will undergo stress testing as well as performance testing.

The following features were tested:

1. Validate Object Detection and Distance Estimation
  - 1.1. Ensure proper interfacing with phone's camera and sending the image via the API
  - 1.2. Ensure the API returns correct responses based on the provided image
2. Validate Image Compression
  - 2.1. Ensure that image compression is performed adequately such that image size is manageable while not losing features
  - 2.2. Ensure no loss in model accuracy due to image compression
3. Validate Screen Reader Accessibility
  - 3.1. Ensure that all components of the front-end are readable to the screen reader

### 6.1 Unit Testing

#### 6.1.1 Test Procedure

##### **ID1**

This test case builds the OnBoardingScreen widget and verifies that the expected text and widgets are present on each page of the onboarding process by performing swipe gestures and making assertions on the screen contents.



```

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:navi/pages/onBoarding.dart';
import 'package:navi/pages/navigation.dart';

void main() {
  testWidgets('OnBoardingScreen displays correctly', (WidgetTester tester) async {
    await tester.pumpWidget(
      MaterialApp(
        home: OnBoardingScreen(),
      ), // MaterialApp
    );

    // Verify that the first page is displayed correctly
    expect(find.text('Welcome To N.A.V.I'), findsOneWidget);
    expect(find.text('(Navigation Assistance For Visually Impaired)'), findsOneWidget);
    expect(find.byType(Image), findsOneWidget);
  });
}

```

The test case sets up a `testWidgets` function that takes a description ("OnBoardingScreen displays correctly") and an asynchronous callback function with a `WidgetTester` parameter. Then the widget tree is built using `pumpWidget` with a `MaterialApp` as the root widget and the `OnBoardingScreen` as the home.

```

// Swipe to the next page
await tester.drag(find.byType(PageView), const Offset(-400.0, 0.0));
await tester.pumpAndSettle();

// Verify that the second page is displayed correctly
expect(find.text('What is N.A.V.I?'), findsOneWidget);
expect(
  find.text(
    'Navigation Assistance for the Visually Impaired, is a mobile application aimed to assist visually impaired',
  ),
  findsOneWidget,
);

```

Assertions are made using the `expect` function to check specific elements on the screen, The tester then performs a drag gesture on the `PageView` to simulate swiping to the next page.

```

// Swipe to the last page
await tester.drag(find.byType(PageView), const Offset(-400.0, 0.0));
await tester.pumpAndSettle();

// Verify that the last page is displayed correctly
expect(find.text('How to Use?'), findsOneWidget);
expect(
  find.text('When using the app, the user needs to hold their phones on chest level and allow the app to access',
  ),
  findsOneWidget,
);

```

Assertions are made to verify the content on the last page, then the test case completed.



## ID2

This test case ensures that the Navigation widget renders a Scaffold with an AppBar containing the correct title. It also verifies the presence of the 'Start Navigation' and 'Stop Navigation' buttons on the screen. These assertions help ensure the proper rendering and content of the Navigation widget.

```
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter/material.dart';
import 'package:navi/pages/navigation.dart'; // Replace with the correct import path

void main() {
  testWidgets('Navigation widget test', (WidgetTester tester) async {
    // Build the widget
    await tester.pumpWidget(
      MaterialApp(
        home: Navigation(),
      ), // MaterialApp
    );

    // Verify that the Scaffold widget is rendered
    expect(find.byType(Scaffold), findsOneWidget);
  });
}
```

The test case is set up using the testWidgets function, which takes a description ("Navigation widget test") and an asynchronous callback function with a WidgetTester parameter.

```
4 // Verify that the Scaffold widget is rendered
5 expect(find.byType(Scaffold), findsOneWidget);
6
7 // Verify that the AppBar is rendered
8 expect(find.byType(AppBar), findsOneWidget);
9
10 // Verify the title of the AppBar
11 expect(find.text('NAVI'), findsOneWidget);
12
13 // Verify the presence of the Start Navigation button
14 expect(find.text('Start Navigation'), findsOneWidget);
15
16 // Verify the presence of the Stop Navigation button
17 expect(find.text('Stop Navigation'), findsOneWidget);
18 });
19 }
```

Inside the callback function, the widget tree is built using pumpWidget with a MaterialApp as the root widget and the Navigation widget as the home. Assertions are made using the expect function to check specific elements on the screen

### 6.1.2 Test cases

| <i>I<br/>D</i> | <i>Descriptio<br/>n</i>  | <i>Steps</i>  | <i>Test<br/>Dat<br/>a</i> | <i>Pre-<br/>conditio<br/>n</i> | <i>Expected<br/>Output</i>   | <i>Passed/Faile<br/>d</i> |
|----------------|--|---|---------------------------|--------------------------------|--|---------------------------|
| 1              | Verify that the OnBoarding Screen displays the correct content on each page. | <ol style="list-style-type: none"> <li>1. Pump the widget tree with the MaterialApp containing the OnBoardingScreen.</li> <li>2. Verify that the first page is displayed correctly.</li> <li>3. Check if the text "Welcome To N.A.V.I" is present.</li> <li>4. Check if the text "(Navigation Assistance For Visually Impaired)" is present.</li> <li>5. Check if an Image widget is present.</li> <li>6. Swipe to the next page.</li> <li>7. Verify that the second page is displayed correctly.</li> <li>8. Check if the text "What is N.A.V.I?" is present.</li> <li>9. Check if a specific description text is present.</li> <li>10. Swipe to the last page.</li> <li>11. Verify that the last page is displayed</li> </ol> | N/A                       | N/A                            | All the specified elements and texts are present on each page of the OnBoardingScreen. | Passed                    |

|   |   |   |     |     |  |        |
|---|---|---|-----|-----|--|--------|
|   |   | <p>correctly.</p> <p>12. Check if the text "How to Use?" is present.</p> <p>13. Check if a specific usage description text is present.</p> <p>14. Check if an Image widget is present.</p> <p>15. Swipe to the next page.</p> <p>16. Verify that the second page is displayed correctly.</p> <p>17. Check if the text "What is N.A.V.I?" is present.</p> <p>18. Check if a specific description text is present.</p> <p>19. Swipe to the last page.</p> <p>20. Verify that the last page is displayed correctly.</p> <p>21. Check if the text "How to Use?" is present.</p> <p>22. Check if a specific usage description text is present.</p> |     |     |  |        |
| 2 | Verify that the appBar title,start and stop navigation button is present. | Similar approach like test ID1  | N/A | N/A | <p>1.The text value of the appBar title should be 'NAVI'.</p> <p>2.The start and stop navigation button should be present.</p> | Passed |

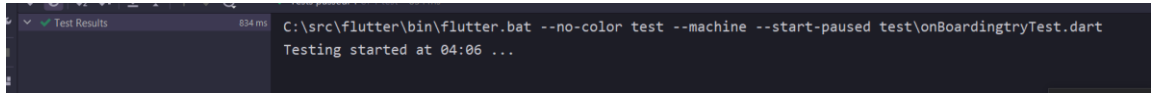
### 6.1.3 Test Results

Based on the expected outputs in table, the test for the OnBoardingScreen has been marked as 'Passed', indicating that all specified elements and texts were successfully displayed on each page. Furthermore, the Navigation widget in the app has also passed, as it successfully

presented the appBar title, start, and stop navigation buttons, confirming the presence of essential navigation controls.

#### 6.1.4 Test Log

The below figures show that the tests have been passed successfully.



```
Test Results 834 ms
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused test\onBoardingtryTest.dart
Testing started at 04:06 ...
```

*On boarding screen displays the correct content on each page.*



```
Test Results 576 ms
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused test\navigation_TEST.dart
Testing started at 04:08 ...
```

*Verify that the navigation page components present.*

## 6.2 Integration Testing

### 6.2.1 Test Procedure

As we mentioned in our test plan report, we are using Flutter SDK for integration testing because Flutter provides the test package that provides core framework for writing integration tests. Specifically, we are testing whether the Navigation widget includes two helper/custom widgets that are ObjectDistancePositionHeader and FrameInformation widgets. Their interactions will be tested specifically how they behave when used as custom widgets in the parent widget. Egemen, Hevra and Ahmed are responsible for integration testing.

#### ID<sub>1</sub>

This integration test allows for testing the interaction and behavior between different components, such as the ElevatedButton, Scaffold, and alertPopUp function, to ensure they integrate correctly within the widget hierarchy of the app.



```
void main() {
  testWidgets('AlertPopUp displays alert dialog with message',
    (WidgetTester tester) async {
      // Build the widget tree
      await tester.pumpWidget(
        MaterialApp(
          home: Scaffold(
            body: Builder(
              builder: (BuildContext context) {
                return ElevatedButton(
                  onPressed: () {
                    alertPopUp(context, 'Test message');
                  },
                  child: const Text('Show Alert'),
                ); // ElevatedButton
              },
            ),
          ),
        );
    },
  );
}
```

The widget tree is built using tester.pumpWidget, starting with a MaterialApp as the root widget. Inside the MaterialApp, a Scaffold is set as the home widget, which defines the main screen structure.

After tapping the button, `tester.pumpAndSettle` is used to wait for any animations or transitions to complete. The test then verifies that the alert dialog is displayed correctly by using expect statements. It checks if the expected texts ('Alert', 'Test message', 'OK') are present in the widget tree. Simulates dismissing the alert dialog by tapping the 'OK' button using `tester.tap`.

```

25     );
26
27     // Tap the button to show the alert dialog
28     await tester.tap(find.byType(ElevatedButton));
29     await tester.pumpAndSettle();
30
31     // Check that the alert dialog is displayed with the correct message
32     expect(find.text('Alert'), findsOneWidget);
33     expect(find.text('Test message'), findsOneWidget);
34     expect(find.text('OK'), findsOneWidget);
35
36     // Tap the OK button to dismiss the alert dialog
37     await tester.tap(find.text('OK'));
38     await tester.pumpAndSettle();
39
40     // Check that the alert dialog is no longer displayed
41     expect(find.byType(AlertDialog), findsNothing);
42   });

```

```

4 void alertPopUp(BuildContext context, message) {
5   showDialog<String>(
6     context: context,
7     builder: (BuildContext context) => AlertDialog(
8       title: const Text('Alert'),
9       content: Text(message),
10      actions: <Widget>[
11        TextButton(
12          onPressed: () async {
13            Navigator.pop(context, 'OK');
14          },
15          child: const Text('OK'),
16        ), // TextButton
17      ], // <Widget>[]
18    ), // AlertDialog
19  );
20 }

```

## 6.2.2 Test cases

Specify your test cases as shown below:

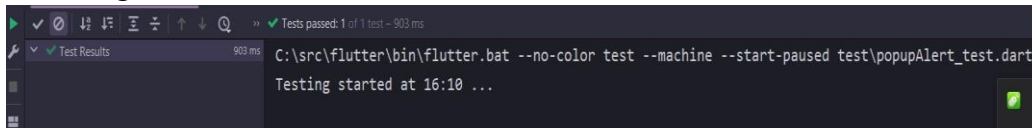
| ID | Description                             | Steps   | Test Data | Pre-condition | Expected Output   | Passed/Failed |
|----|---|---|-----------|---------------|---|---------------|
| 1  | Triggers the display of an alert dialog | Build the widget tree with an ElevatedButton triggering the alertPopUp function | N/A       | N/A           | The alert dialog is displayed with the title "Alert", the message "Test message", and an "OK" button. | Passed        |

### 6.2.3 Test Results

The test for triggering the display of an alert dialog was marked as 'Passed'. This indicated that the widget tree with an ElevatedButton successfully triggered the alertPopUp function, and the alert dialog was displayed with the expected title "Alert", the message "Test message", and an "OK" button.

All the specified tests have passed, confirming that the alert dialog was triggered and displayed correctly with the expected content.

### 6.2.4 Test Log



## 6.3 System Testing

### 6.3.1 Test Procedure

The system testing has been mostly done manually due to the absence of tools that can cover diverse scenarios. The activities performed are dependent on which scenario is being tested. For testing unusual or rare scenarios, the backend is intentionally brought down in order to test the frontend's reaction. On the other hand, for more expected application usage flow, the application is being used by the developers to act out the scenarios. Hevra and Ahmed are responsible for performance testing.

### 6.3.2 Test Scenarios

Initially, when the user opens the application for the first time, they will be introduced to what the application is and how it is expected to behave. Later, when the introduction is done, they will be asked to give permission for camera access. Once the permission is given and the user clicks on the start navigation button, the phone's camera will take snapshots every 3 seconds and send the snapshot as a compressed image to the server. Next, the server will forward the image to the machine learning model, where object detection will be performed. Later, distance estimation and position estimation will take place. Finally, the server will combine all these data and send it back to the frontend. The received data will be read by the screen reader to the user. This process will keep continuing until the user clicks on the stop navigation button. Once that button is clicked, the frontend will confirm with the user that they have in fact stopped the navigation and it wasn't done by mistake. This might seem unnecessary; however, considering that the user is blind and can have an accident if they aren't aware that navigation has been stopped, it becomes clear why it is a crucial to have this. We can also have incidents where the server is down. In that case, the user is informed about it.

Below are the items that need to be tested based on the above scenario.

- 1) When the server is down, the user should be informed about it to prevent any accidents.
- 2) When the user opens the application the introduction pages are shown to the user only once.
- 3) When the user starts the navigation, there should be continuous flow of information, which is detected objects,

distances, and positions, from the backend to the frontend.

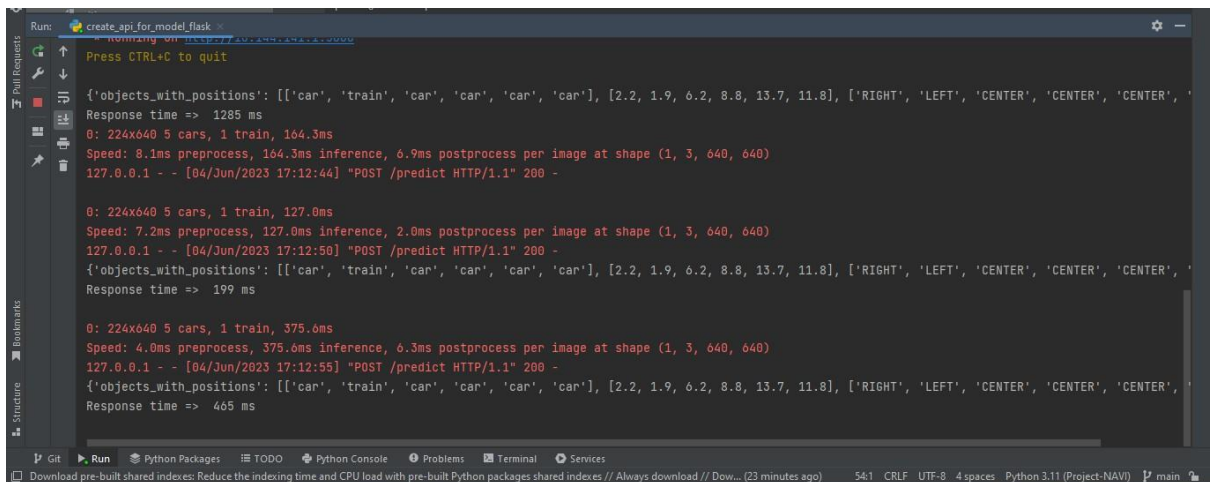
- 4) When the user clicks on the stop navigation button, the user should be alerted/warned that the navigation has actually been stopped.

### 6.3.3 Test Results

- 1) Passed. A connection attempt was made before the server was started. Hence, the user was informed about the server being down.
- 2) Passed. The introduction pages were only shown once when the user started using the application.
- 3) Passed. The information was received from the backend, displayed by the frontend, and read by the screen reader. The information was flowing continuously with no interruption.
- 4) Passed. The user was alerted/warned when the navigation was stopped by them.

### 6.3.4 Test Log

As it can be seen in Figure 20, a request was sent to the server from the virtual emulator, and it was received by the server successfully. The compressed image that was attached to the request was processed by the server and a response was returned to the frontend including the detected objects, their distances, and positions. Note that we are not displaying this information on the screen, rather the screen reading is reading them.



```
Run: create_api_for_model_flask
Press CTRL+C to quit

{
  "objects_with_positions": [
    [
      "car",
      "train",
      "car",
      "car",
      "car",
      "car"
    ],
    [
      2.2,
      1.9,
      6.2,
      8.8,
      13.7,
      11.8
    ],
    [
      "RIGHT",
      "LEFT",
      "CENTER",
      "CENTER",
      "CENTER",
      "CENTER"
    ]
  ]
}
Response time => 1285 ms
0: 224x640 5 cars, 1 train, 164.3ms
Speed: 8.1ms preprocess, 164.3ms inference, 6.9ms postprocess per image at shape (1, 3, 640, 640)
127.0.0.1 - - [04/Jun/2023 17:12:44] "POST /predict HTTP/1.1" 200 -

0: 224x640 5 cars, 1 train, 127.0ms
Speed: 7.2ms preprocess, 127.0ms inference, 2.0ms postprocess per image at shape (1, 3, 640, 640)
127.0.0.1 - - [04/Jun/2023 17:12:50] "POST /predict HTTP/1.1" 200 -

{
  "objects_with_positions": [
    [
      "car",
      "train",
      "car",
      "car",
      "car",
      "car"
    ],
    [
      2.2,
      1.9,
      6.2,
      8.8,
      13.7,
      11.8
    ],
    [
      "RIGHT",
      "LEFT",
      "CENTER",
      "CENTER",
      "CENTER",
      "CENTER"
    ]
  ]
}
Response time => 199 ms
0: 224x640 5 cars, 1 train, 375.0ms
Speed: 4.0ms preprocess, 375.0ms inference, 6.3ms postprocess per image at shape (1, 3, 640, 640)
127.0.0.1 - - [04/Jun/2023 17:12:55] "POST /predict HTTP/1.1" 200 -

{
  "objects_with_positions": [
    [
      "car",
      "train",
      "car",
      "car",
      "car",
      "car"
    ],
    [
      2.2,
      1.9,
      6.2,
      8.8,
      13.7,
      11.8
    ],
    [
      "RIGHT",
      "LEFT",
      "CENTER",
      "CENTER",
      "CENTER",
      "CENTER"
    ]
  ]
}
Response time => 465 ms
```

Figure 20: Server receiving request and responding

Furthermore, Figure 21 shows the server being down and Figure 22 and 23 shows the code that can handle it.



```
1 POST /predict ConnectionRefusedError(10061, [WinError 10061] No connection could be made because the target machine actively refused it.)
```

Figure 21: Unresponsive server



```

// If you want to send images/videos/files to the server, use MultipartRequest
var request = http.MultipartRequest(
  'POST', Uri.parse('http://10.0.2.2:5000/predict'));

// adding the image file
request.files.add(multipart);

// taking into consideration that the server may go down or face a problem
// let the user know about it accordingly.
try {
  // send the api request for object detection
  var streamedResponse =
    await request.send().timeout(const Duration(seconds: 5));
  // converting the streamed response to a casual response
  var response = await http.Response.fromStream(streamedResponse);
  var response_decoded = json.decode(response.body);
  return response_decoded["objects_with_positions"];
} catch (_) {
  return [
    [-222],
    [],
    []
  ];
}
}

```

Figure 22: Catching the unresponsive server

```

lib > pages > navigation.dart
// make API requests every 5 seconds
await Future.delayed(const Duration(seconds: 6), () async {
  scene = await handle_scene_description();
});

// no objects detected and no server error
if (scene[0].isEmpty) {
  continue;
}

// objects detected and no server error
else if (scene[0][0] != -222) {
  // use setState so that the widget gets rerendered to display
  // the objects, distances, and angles
  setState(() {
    objects_with_positions = scene;
  });
}

// server error
} else {
  // show an alert to the user since the server is irresponsive and can't detect objects
  var message =
    "Currently the app is unable to detect objects.\nConsequently, please take cautions accordingly and try again later.";
  SemanticsService.announce(message, TextDirection.ltr);
}

```

Figure 23: Warning getting announced

## 6.4 API Performance Testing

### 6.4.1 Test Procedure

As we mentioned in our test plan report, we are using Python's Locust [1] package for performance testing please refer to Test Tools section for more details. Specifically, we are testing the machine learning model's API's response time and its performance as the number of users increase. Hevra and Ahmed are responsible from API performance testing.

As it can be seen in Figure 24, there is a custom user class 'MyUser' that extends the 'HttpUser' class provided by Locust. The wait\_time attribute is set to 'between(3,3)'. This means that each virtual user will wait for exactly 3 seconds before sending the next request. It simulates the assumption that each user will make an API request every 3 seconds. Furthermore, the @task decorator is used to define a task called 'make\_api\_request'. This task represents the behavior of the virtual user when it performs an API request. Finally, the virtual user sends a POST request using the 'self.client.post()' method. An image is attached to the request for object detection to be performed by the machine learning model in the server.

```
el.py × API_response_time_testing.py × create_api_for_model_flask.py × utils.py × mini_labels.txt × train.py × config.py × explanation_of...
from locust import HttpUser, between, task

# Create a Locust User class that represents the behavior of the virtual users in your test
class MyUser(HttpUser):
    # By specifying between(3, 3), it means that each virtual user will wait for
    # 3 seconds before sending the next request.
    # This is in consistent with our assumption that each user will make an api request every 3 seconds.
    wait_time = between(3, 3)

    @task
    def make_api_request(self):
        # since the ml model needs an image for performing predictions,
        # we need to send an image, which is from our dataset, in the post request.

        file_path = "../Dataset/images/000000.png"
        # Open the image file
        with open(file_path, "rb") as file:
            # Create a dictionary with the file data
            files = {"image": file}

        # Send the POST request with the image file
        self.client.post("/predict", files=files)

> make_api_request() > with open(file_path, "rb") as f...
```

Figure 24: Performance test code using Locust for endpoint

We are using the command in Figure 25 to run the Locust tests

```
locust -f tests/performance-tests/locust_test.py
```

Figure 25: Command to run Locust tests

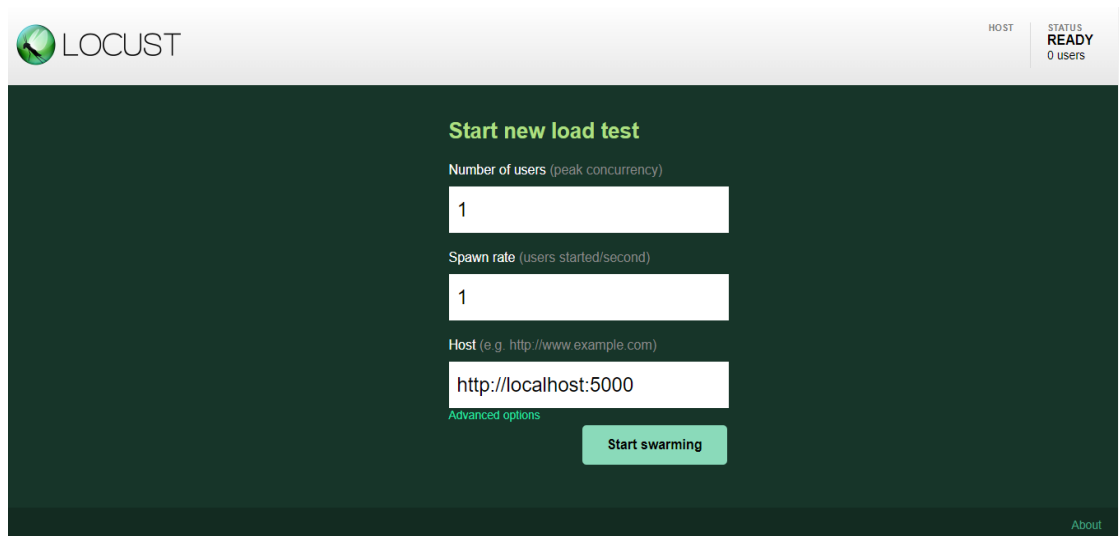
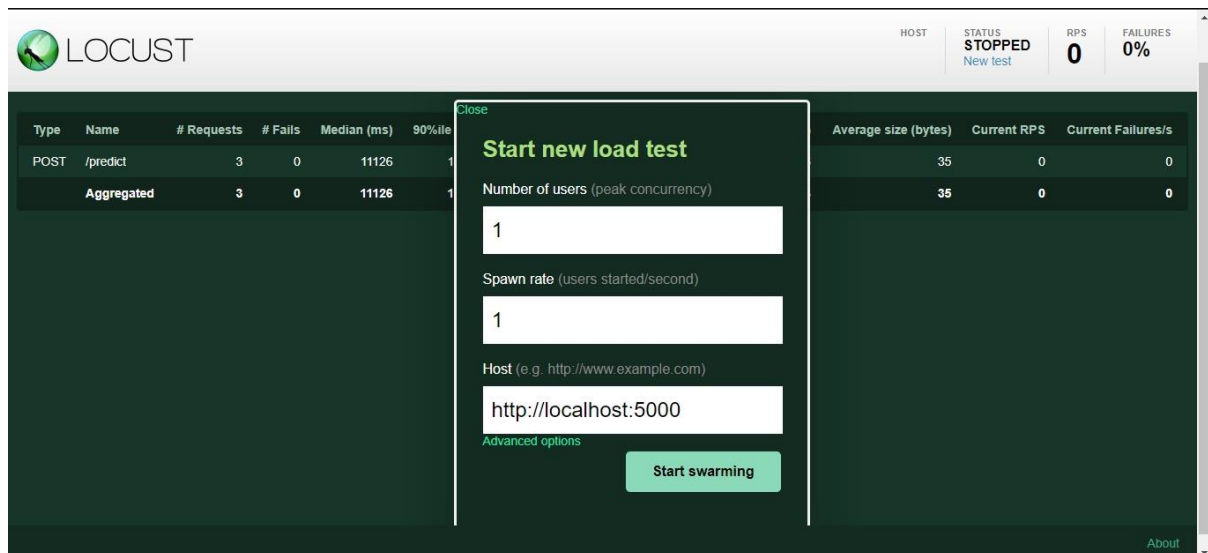


Figure 26: Locust's web interface



Note that “Start new load test” screen pops up when we click on “New test” under “STATUS”. It is used to test the API with various conditions.

In order to test the API response time, the number of users (peak concurrency) is limited to one user. Note that initially we are not performing load testing that is why we keep the number of users as 1. However, later on, we will also perform load testing and stress testing where we will need to increase the number of users. The reason for performing both stress and load testing is that they serve different purposes and show our API performance through different perspectives. Stress testing will be used to determine how the system will behave in extreme or unfavorable conditions. Whereas, load testing involves testing the API under normal or expected usage scenarios to measure its response time, throughput, which is the amount of requests it is able to serve, and overall performance.

The machine learning model is part of the core functionality of the mobile application, hence, it is shared by all the users. The memory requirement for multi-threading the model is too high for our systems, thus it is out of the question; i.e. keeping track of a model per user. Under ideal circumstances, the model would be hosted through a Cloud Service Provider, which would handle load optimization and would have the hardware necessary for multi-threading such a complex model

#### 6.4.2 Test cases

| ID | Description                  | Steps | Test Data                 | Pre-condition  | Expected Output           | Passed/Failed |
|----|------------------------------|-------|---------------------------|--|---------------------------|---------------|
| 1  | ML API response time testing |       | Image sent from the user. | The model architecture should be ready.<br>Should have internet access.<br><br>Image should be compressed. | response_time < 2 seconds | Passed        |

|   |                       |  |               |  |  |        |
|---|-----------------------|--|---------------|--|--|--------|
| 2 | ML API load testing   | Test it with different number of users | Virtual users | Images should be compressed.<br><br>Backend server is single threaded. | Expecting linear increase in response time.<br>User_x_response time = 2 + (2*number_of_users_in_queue).<br>Assuming that the response time for an API call is max 2 seconds. | Passed |
| 3 | ML API stress testing |  | Virtual users | The system should be already warmed up, no cold start.                 | Expecting the system will not freeze or crash under extreme conditions.  | Passed |

#### 6.4.3 Test Results

It is crucial to note that the machine learning model suffers from cold start. Hence, in some of the reports, the initial response time is dramatically high. Assuming that the system will be continuously running, in the long run, this will not be a problem.

In order to see the baseline response time statistics, please check the Test Log section. Figure 36 shows the baseline response time performance testing meaning that how fast the application's response is when a single user is using it and sending API requests every 3 seconds, please check the Test Log section. Figure 27 and Figure 29 show that the number of users is kept as 1 and the API request is being made every 3 seconds respectively. In other words, the frequency is  $1/3 = 0.33$  but the period is 3.

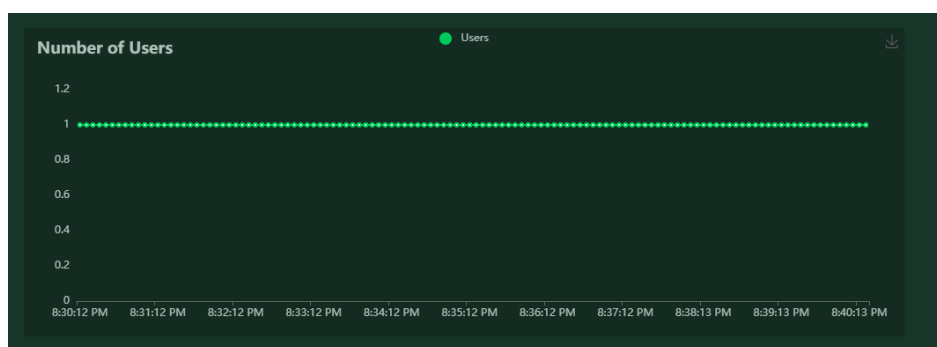


Figure 27: Baseline response time testing

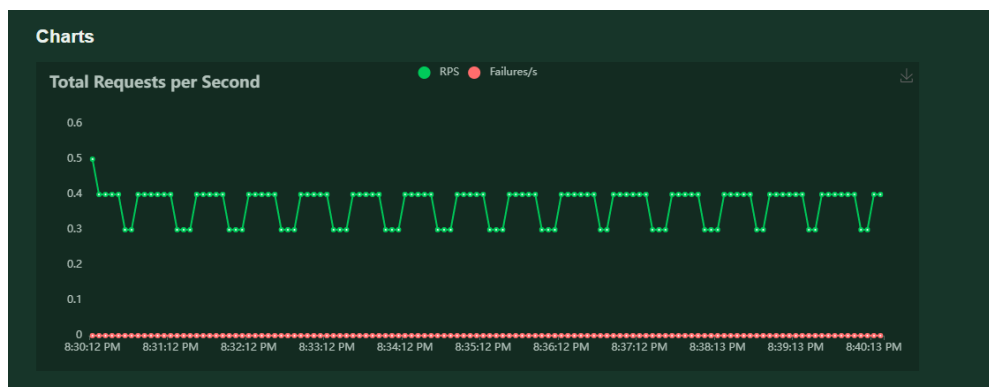


Figure 28: Baseline response time charts

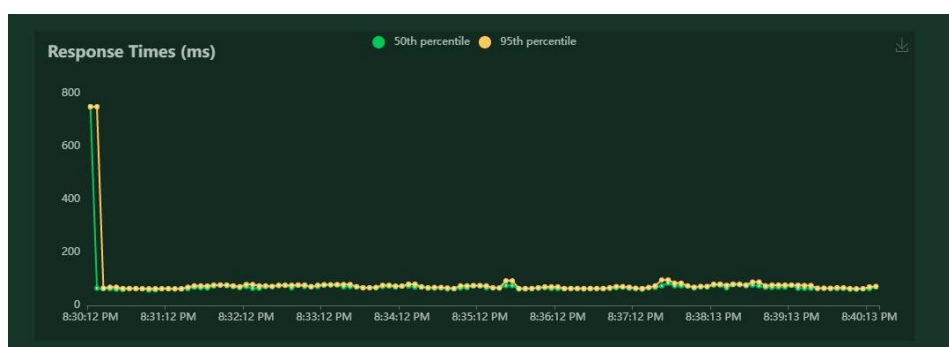


Figure 29: Baseline response time - over a continuous period

Figure 31 shows the stress testing. It is a type of software testing that evaluates the stability, reliability, and performance of a system under extreme or unfavorable conditions. The purpose of stress testing is to identify the system's breaking point. During stress testing, the system is subjected to high levels of user traffic.

As a start, we randomly decided to choose 100 users, as can be seen in Figure 31, where each user is sending a request every 3 seconds. Please refer to the Test Log to see the test report statistics. The average response time is around 4.4 seconds, which is not optimal.

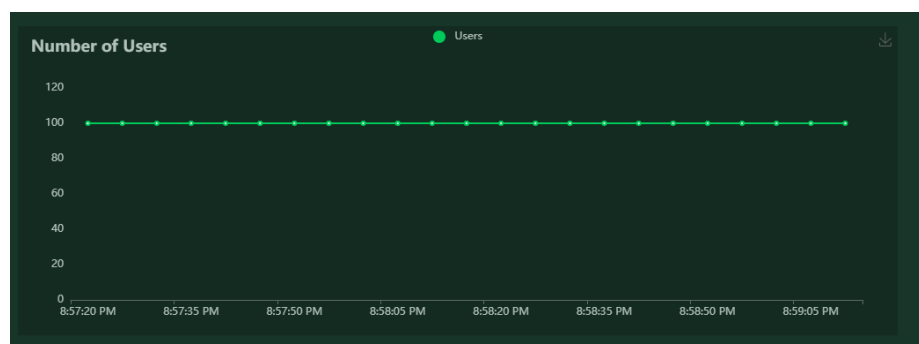


Figure 30: Stress testing - fixed 100 users

In order to see how the response time increases as the number of users increases, we increased the number of users linearly and expected a linear increase in the response time. As can be seen in Figure 33, that is in fact the case.



Figure 31: Stress testing - Response time increase as number of users increase

Additionally, we decided on a threshold for the response time so that we can see how many users the server can handle, where the response time does not increase the decided threshold, which is 1 second. It should also be noted that the response time is more directly dependent on the average request per given time rather than average user per given time. Hence, we will check the number of requests per second that cause the response time to increase beyond one second.

As it can be seen in Figures 34 and 35, beyond 9 requests per second, the response time increases to be above 1 second, hence, we consider 9 requests per second as our limit, which implies that the server can serve 27 users (1 request per 3 seconds) at once with no significant delay.



Figure 32: Increase in request as users incrementally increased

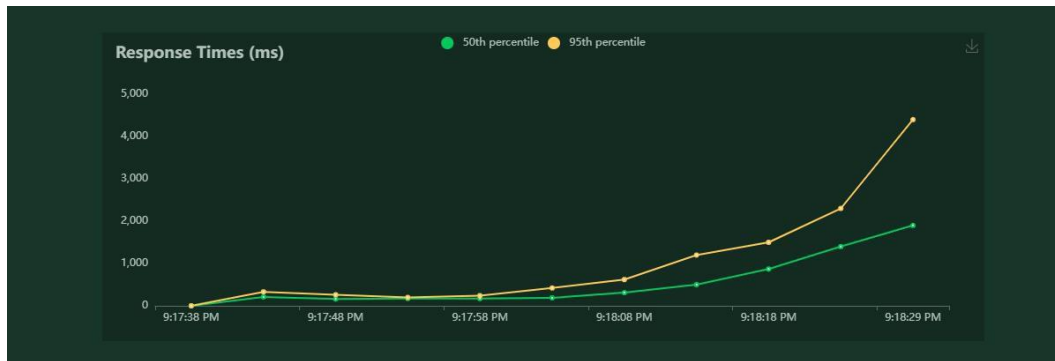


Figure 33: Response time - 100 concurrent users incrementally increased

The stress testing is considered to be passed because even when the number of requests increased, the server was able to respond without crashing or freezing irrespective of how high the response time was. We can safely ignore the response time because we already considered that when deciding on the number of users that can be handle by the system with no significant delay.

#### 6.4.4 Test Log

Figure 36 shows the baseline response time performance testing meaning that how fast the application's response is when a single user is using it and sending API requests every 3 seconds.

| Locust Test Report                                  |          |            |         |              |          |          |                      |     |            |
|---|----------|------------|---------|--------------|----------|----------|----------------------|-----|------------|
| During: 6/3/2023, 8:30:07 PM - 6/3/2023, 8:40:20 PM |          |            |         |              |          |          |                      |     |            |
| Target Host: http://127.0.0.1:5000                  |          |            |         |              |          |          |                      |     |            |
| Script: performance_testing.py                      |          |            |         |              |          |          |                      |     |            |
| Request Statistics                                  |          |            |         |              |          |          |                      |     |            |
| Method  | Name     | # Requests | # Fails | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | RPS | Failures/s |
| POST  | /predict | 200        | 0       | 71           | 55       | 751      | 150                  | 0.3 | 0.0        |
| Aggregated  |          | 200        | 0       | 71           | 55       | 751      | 150                  | 0.3 | 0.0        |

Figure 36: Baseline response time statistics

Figure 37 shows the Stress testing statistics with 100 users. It shows an average response time of around 4.4 seconds. Please refer to the Test Results for further explanation.

| Locust Test Report                                  |          |            |         |              |          |          |                      |      |            |
|---|----------|------------|---------|--------------|----------|----------|----------------------|------|------------|
| During: 6/3/2023, 8:57:16 PM - 6/3/2023, 8:59:12 PM |          |            |         |              |          |          |                      |      |            |
| Target Host: http://127.0.0.1:5000                  |          |            |         |              |          |          |                      |      |            |
| Script: performance_testing.py                      |          |            |         |              |          |          |                      |      |            |
| Request Statistics                                  |          |            |         |              |          |          |                      |      |            |
| Method  | Name     | # Requests | # Fails | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | RPS  | Failures/s |
| POST  | /predict | 1591       | 0       | 4440         | 71       | 8011     | 150                  | 13.7 | 0.0        |
| Aggregated  |          | 1591       | 0       | 4440         | 71       | 8011     | 150                  | 13.7 | 0.0        |

Figure 37: Stress testing statistics with 100 users

## 6.5 Model Performance Testing

### 6.5.1 Test Procedure

We planned on using cross-validation due to it being a better performance indicator since the test is done over different folds and hence is more realistic. However, due to YOLO's huge architecture and huge dataset, it would've taken 14 days to complete the testing because each epoch was taking 7 minutes and given that we have 10 folds, where each fold should run through 200 epochs. The number of epochs is 200 because that is when the model accuracy starts to converge

Consequently, we decided to go with the traditional 80% train and 20% test technique for model performance testing. As can be seen in Figure 38, we passed the "test\_loader", which holds the test dataset, to the custom "check\_model\_accuracy" function; for the code output, please refer to the Test Log section. Furthermore, we tested two different models. First, we tested Dist-YOLO and then we tested YOLOv8.

```
if __name__ == '__main__':  
    with open('epoch number 60.pk', 'rb') as file:  
        model = pickle.load(file)  
        file.close()  
  
    test_loader = get_loaders()  
    check_model_accuracy(model, test_loader, config.CLASS_CONFIDENCE_THRESHOLD, config.CONF_DIST_THRESHOLD)
```

Figure 38: Dist-YOLO model performance test code

Our test metric is class accuracy. For Dist-YOLO we used Top-1 accuracy; however for YOLOv8, we used Top-1 and Top-5 accuracies. To find more information about what Top-1 and Top-5 are please refer to the Test Results section.

Hevra and Ahmed are responsible from Model performance testing.

### 6.5.2 Expectations

For Dist-YOLO, we expected to get an accuracy level that is better than the traditional YOLO family. That is because based on the research paper that we followed and tried to implement Dist-YOLO accordingly, it was stated that based on their analysis, Dist-YOLO performs better than the YOLO family mainly due to the multi-tasking and curriculum learning principles. However, when we tried to follow these principles and build the model architecture and loss functions accordingly, we did not get satisfactory results. For more explanation about the reason, please refer to the Test Results section.

YOLOv8 was trained on COCO80 dataset. A random guess of class prediction will give an accuracy of  $1/80 = 0.0125$ . Consequently, any value above this would imply that some learning has been performed. However, that would be a very low expectation because we need to take some other variables into consideration, model complexity, baseline and state of art performances, task complexity, and dataset characteristic. Firstly, regarding the model complexity, given that YOLOv8 is the latest version of the YOLO family, we suspect that its architecture holds the most advanced layer structure. Nevertheless, given that YOLOv8 is more concerned with detection rather than classification, we do not expect a very high classification accuracy.



### 6.5.3 Test Results

For Dist-YOLO, the result we got is an indication of failure because it is very low compared to the expectations raised by the Dist-YOLO paper authors. Coming to the possible reason, we are suspicious of the model architecture, loss functions implementation, or, data pre-processing before sending it to the model.

We have concerns about the architecture of the model as we believe that an additional convolutional layer needs to be incorporated into the standard YOLO architecture. The reason behind this is our expectation for the model to learn a new parameter, specifically the distance. Introducing a new parameter to be learned necessitates the inclusion of new filter weights, and the most appropriate way to achieve this is through a convolutional layer. Despite implementing the necessary changes as described, we suspect that there may still be an issue within the architecture, resulting in unexpected behaviour.

In addition, we have concerns about the loss function. While YOLO already incorporates loss functions for class accuracy and objectness accuracy, we have included a distance loss to enable the model to learn and adjust its weights accordingly. However, we suspect that even a minor issue with the loss function could lead to incorrect weight updates, which is a critical aspect of model training.

One of the final concerns lies in the data pre-processing phase of YOLO. To enable accurate bounding box predictions, anchors need to be properly prepared. These anchors serve as reference points for predicting bounding box coordinates. Additionally, the labels we have (bounding boxes) are transformed into three different labels, each corresponding to a different scale, based on the prepared anchors. During evaluation, these predictions are compared with the model's predictions, and class accuracy and objectness accuracy are calculated accordingly. It's important to note that even a minor error in anchor preparation or the transition from one label to three labels can have a significant impact on loss calculation. We already explained how incorrect loss calculation can affect the training process in the above paragraph.

Next, we can discuss the accuracy of YOLOv8. When dealing with YOLO class accuracy, the terms "top1" and "top5" refer to different ways of evaluating the accuracy of object detection models, such as YOLOv8.

1. Top-1 accuracy: Top-1 accuracy measures the percentage of objects that are correctly detected and classified by the model when considering only the most confident prediction for each object. It means that the model's prediction for an object must be the correct class with the highest confidence score. The model's prediction is considered correct if the predicted class matches the ground truth class.
2. Top-5 accuracy: Top-5 accuracy is a more lenient metric that considers the model's prediction to be correct if the correct class is present among the top 5 predictions with the highest confidence scores. In other words, if the ground truth class appears within the top 5 predicted classes by the model, the prediction is considered correct.

In summary, top-1 accuracy is a stricter metric that requires the model to provide the correct class with the highest confidence score, while top-5 accuracy allows for more flexibility by considering the correct class to be present within the top 5 predictions. The output of the test code can be seen in the Test Log section.

### 6.5.4 Test Log

Figure 39 shows the output of the YOLOv8 classification accuracy test code. For explanation about what top1\_acc and top5\_acc are please refer to the Test Results section. Top1\_acc is 0.35; whereas, the top5\_acc is 0.68 that is 35% and 68% respectively.

```
classes top1_acc top5_acc: 100% | 7/7 [00:07<00:00, 1.05s/it]
all      0.35      0.68
```

Figure 39: YOLOv8 - Top\_1 and Top\_5 accuracies

Moreover, Figure 40 shows Dist-YOLO model performance test code's output, please refer to the test procedure to see the code screenshot. Note that the distance accuracy is calculated using a threshold, meaning that if the difference between the model's distance prediction and the actual distance of the object is below a threshold, the prediction is counted as true. For the explanation of the reasons behind getting these results, please refer to the Test Results section.

```
Class accuracy is: 0.218226%
No obj accuracy is: 0.056686%
Obj accuracy is: 97.132500%
Distance accuracy is: 0.374251%
```

Figure 40: Dist-YOLO model performance test code output

## 7 Project Scheduling

### 7.1 Milestones and Tasks

| ID | Milestone           | Date    |
|----|---------------------|---------|
| M1 | Project Proposal    | Week 5  |
| M2 | Back-end Prototype  | Week 11 |
| M3 | Front-end Prototype | Week 14 |
| M4 | Presentation        | Week 15 |
| M5 | Back-end Release    | Week 25 |
| M6 | Front-end Release   | Week 26 |
| M7 | Demonstration       | Week 28 |

| #  | Task Name                      | Duration | Members      |
|----|--------------------------------|----------|--------------|
| 1  | Finalize SRS                   | 3        | All          |
| 2  | Finalize SDD (M1)              | 2        | All          |
| 3  | Develop ML Model               | 2        | Hevra/Ahmed  |
| 4  | Interview Blind Girl           | 1        | Hevra        |
| 5  | Revise ML Model (M2)           | 3        | Hevra/Ahmed  |
| 6  | Develop Front-end (M3)         | 3        | Egemen/Hevra |
| 7  | Prepare Presentation (M4)      | 1        | All          |
| 8  | Troubleshoot Dist-Yolo         | 6        | Hevra/Ahmed  |
| 9  | Front-End Refinement & Testing | 6        | Egemen       |
| 10 | Revise Back-end                | 3        | Hevra/Ahmed  |
| 11 | Test Back-end (M5)             | 1        | Hevra/Ahmed  |
| 12 | Front-end Revision (M6)        | 5        | Hevra/Ahmed  |
| 13 | Prepare Demo (M7)              | 1        | All          |

## 8 Conclusion

During the progress of the project, concessions had to be made in terms of the machine learning model. Due to the complexity of YOLO's architecture and its demanding training procedure, we were not able to augment the architecture to add our own layers and make the model capable of distance estimation. Due to hardware inefficacy and the limited time frame, we had no choice but to discard our own model and use a pretrained version of YOLOv8, coupled with the old method of distance calculation we used during the prototyping stage. This resulted in the inclusion of an extra hyperparameter, the focal point, which we had to guesstimate based on publicly available data about smartphone camera architecture. The front-end was subject to radical changes during the final leg of our development timeline, as we realized that we were capable of greatly simplifying the UI, making the application more intuitive for screen-readers, while also reducing the number of components we needed to keep track of. For future work, we can think about integrating the phone's gyroscope, this would allow us to constantly monitor the orientation of the phone, ensuring that the user has the phone in the right orientation, as well as alerting them in case it needs adjustment. It is possible to also revisit the dist-yolo model, a lenient timeframe coupled with stronger hardware might prove the model viable. For stretch goals, we can look into integrating a navigation module that would give users directions to their set destination while simultaneously monitoring for obstacles. Overall, the project was a rich source of experience in the world of machine learning and mobile app development. A valuable lesson in resource and expectation management was learnt, as well as the importance of pre-planning. While we were very eager to start developing the project, we will likely adopt a more cautious and calculated approach in the future.

## 9 References

- "An introduction to unit testing," [Online]. Available: <https://docs.flutter.dev/cookbook/testing/unit/introduction>.
- "An introduction to integration testing," Flutter, [Online]. Available: <https://docs.flutter.dev/cookbook/testing/integration/introduction>.
- C. B. J. H. H. Jonatan Heyman, "An open source load testing tool," [Online]. Available: <https://locust.io/>.
- s.-l. developers, "3.1. Cross-validation: evaluating estimator performance," [Online]. Available: [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html).
- "Accessibility," [Online]. Available: <https://docs.flutter.dev/development/accessibility-andlocalization/accessibility>.
- J. Juviler, "API Response Time, Explained in 1000 Words or Less," 2022. [Online]. Available: <https://blog.hubspot.com/website/api-responsetime#:~:text=Generally%2C%20APIs%20that%20are%20considered,begin%20to%20not ice%20s ome%20delay..>
- "flutter\_image\_compress," 2023. [Online]. Available: [https://pub.dev/packages/flutter\\_image\\_compress](https://pub.dev/packages/flutter_image_compress).
- "Providing Accessible Names and Descriptions," 2023. [Online]. Available: <https://www.w3.org/WAI/ARIA/apg/practices/names-and-descriptions/>.
- shawn.blais, "Flutter: Crafting a great experience for screen readers," 2022. [Online]. Available: <https://blog.gskinner.com/archives/2022/09/flutter-crafting-a-great-experience-for-screenreaders.html>.
- [10] "Web Content Accessibility Guidelines (WCAG) 2.1," 2018. [Online]. Available: <https://www.w3.org/TR/WCAG21/>.
- Cooper, B. (n.d.). Retrieved May 13, 2019, from <https://zapier.com/blog/prioritize-task-listmethods/>
- Everything You Need to Know About White Canes. (2021). Retrieved from <https://lhblind.org/everything-you-need-to-know-about-white-canes/>
- Flutter: the framework for cross-platform applications. (2021). Retrieved from Appify Digital.
- Girshick, R. (n.d.). Fast R-CNN. Retrieved from [https://www.cvfoundation.org/openaccess/content\\_iccv\\_2015/papers/Girshick\\_Fast\\_RCNN\\_ICCV\\_2015\\_paper.pdf](https://www.cvfoundation.org/openaccess/content_iccv_2015/papers/Girshick_Fast_RCNN_ICCV_2015_paper.pdf)
- GUIDE DOGS VS. WHITE CANES: THE COMPREHENSIVE COMPARISON. (2020). Retrieved from <https://clovernook.org/2020/09/18/guide-dogs-vs-white-canes-thecomprehensivecomparison/>
- Joseph Redmon, A. F. (n.d.). YOLOv3: An Incremental Improvement. Retrieved from <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- Marek Vajgl, P. H. (2022). Dist-YOLO: Fast Object Detection with Distance Estimation. Institute for Research and Applications of Fuzzy Modeling. Retrieved from <https://www.mdpi.com/2076-3417/12/3/1354/htm#B49-applsci-12-01354>
- Page, T. (2020). A robot suitcase could replace canes and guide dogs for blind people. Retrieved from <https://edition.cnn.com/travel/article/ai-suitcase-blind-chieko-asakawaspc-intl/index.html>

Python vs other programming languages. (2022). Retrieved from vilmate software developmet.

Renotte, N. (n.d.). Deep Drowsiness Detection using YOLO, Pytorch and Python. Retrieved from 30 <https://www.youtube.com/watch?v=tFNJGim3FXw&t=4159s>

Rosebrock, A. (2021). Find distance from camera to object/marker using Python and OpenCV. Retrieved from <https://pyimagesearch.com/2015/01/19/find-distance-cameraobjectmarkerusing-python-opencv/>

S. Kiruthika Devi, C. N. (2021). Deep Learning Based Audio Assistive System for Visually Impaired People. SRM Institute of Science and Technology. Tech Science Press. Retrieved from <https://www.techscience.com/cmc/v71n1/45390/html>

Shah, D. (n.d.). Mean Average Precision (mAP) Explained: Everything You Need to Know. Retrieved from

[https://www.v7labs.com/blog/meanaverageprecision#:~:text=let%27s%20dive%20in!-,What%20is%20Mean%20Average%20Precision%20\(mAP\)%3F,values%20from%200%20to%201](https://www.v7labs.com/blog/meanaverageprecision#:~:text=let%27s%20dive%20in!-,What%20is%20Mean%20Average%20Precision%20(mAP)%3F,values%20from%200%20to%201)

Shaoqing Ren, K. H. (2016). Faster R-CNN: Towards Real-Time Object. Retrieved from <https://arxiv.org/pdf/1506.01497.pdf>

Sharma, A. (2022). Introduction to the YOLO Family. Retrieved from <https://pyimagesearch.com/2022/04/04/introduction-to-the-yolo-family/>

SreerajM, J. A. (2020). VIZIYON: Assistive handheld device for visually challenged. Elsevier B.V. Retrieved from [https://www.sciencedirect.com/science/article/pii/S1877050920312618?ref=pdf\\_download&fr=RR-2&rr=76220dd80ff0c2f2](https://www.sciencedirect.com/science/article/pii/S1877050920312618?ref=pdf_download&fr=RR-2&rr=76220dd80ff0c2f2)

Taylor Mordan, N. T. (n.d.). Revisiting Multi-Task Learning with ROCK: a Deep. Retrieved from <https://proceedings.neurips.cc/paper/2018/file/7f5d04d189dfb634e6a85bb9d9adf21ePaper.pdf>

Team, Q. E. (2021). How Camera Resolution In Machine Vision Plays An Important Role: 2021. Retrieved from QUALITAS TECHNOLOGIES: [https://qualitastech.com/imageacquisition/camera-resolution-in-machinevision/#:~:text=So%2C%20we%20need%20a%20camera,%20t.%20\(n.d.\).%20How%20Camera%20Resolution%20In%20Machine%20Vision%20Plays%20An%20Important%20Role:%20202](https://qualitastech.com/imageacquisition/camera-resolution-in-machinevision/#:~:text=So%2C%20we%20need%20a%20camera,%20t.%20(n.d.).%20How%20Camera%20Resolution%20In%20Machine%20Vision%20Plays%20An%20Important%20Role:%20202)

World Health Organization. (2022). Retrieved from <https://www.who.int/news-room/factsheets/detail/blindness-andvisualimpairment#:~:text=Globally%2C%20at%20least%202.2%20billion,near%20or%20di stance%20vision%20impairment>

Yoshua Bengio, J. L. (n.d.). Curriculum Learning. Retrieved from <https://dl.acm.org/doi/pdf/10.1145/1553374.1553380>

## 10 Appendices

### 10.1 Acronyms and abbreviations

- API => Application Programming Interface
- ML => Machine Learning
- YOLO => You Only Look Once

- NAVI => Navigation Assistance for the Visually Impaired

## 10.2 Glossary

Thin Client: A computer that lacks the resources needed to run some functionalities and hence communicates with external servers to access those functionalities.