Android Driving Assistant
Final Year Project Report

DT228
BSc in Computer Science

Yit Chee Chin

Mark Foley

School of Computing

Dublin Institute of Technology

12/04/2019

Abstract

Recent advances in technology have allowed for new car models to be designed with AI features built n. The objective of the project is to develop a mobile application for use in older vehicles that incorporate some of these features. When it is set up, it will act as a passive tool to help users drive in a safe manner by informing them of certain conditions in real-time.

The result of the project will be a native Android application that uses image processing and text recognition to detect lanes and speeds signs to understand what it needs to alert the user about. It will use the mobile device as the only medium where data retrieval and processing takes place. No connections to any external devices such as servers mean that the delay in the app will be minimal.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Yit Chee Chin

Yit Chee Chin

12/04/2019

Acknowledgements

Firstly, I would like to thank my project supervisor Mark Foley for his help and guidance throughout the course if the project over the last few months. I would also like to thank all the lecturers I had over the years that have provided me with the skills necessary to undertake this project. Finally, I would like to thank my friends that tested the application and gave advice on design choices.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Overview and Background

The idea for Android Driving Assistant was inspired by developments in the autonomous vehicles industry. Recent advances in technology have allowed for new car models to be designed with AI features built in however, they are only implemented in newer and higher-end models. Android Driving Assistant is a native application designed to use the phone's camera to compensate for the lack of AI features in older vehicles. By mounting the phone in a place where it has a view out of the windscreen, it uses some methods found in autonomous vehicles to alert the user about the vehicle speed, speed limits, and lane departure.

Android Driving Assistant is intended to be a passive tool. When it is set up, the user will not have to interact with the device until the end of a drive. The app will continuously process the video data and provide audio feedback to the user where necessary. It also runs locally on the device, no external connections are required except a GPS signal, which is embedded in all of the target devices of my project.

1.2 Project Objectives

The overall objective of this project is to design and develop an Android application that aids drivers in traffic by providing some signals to help them to drive in a safe manner. This app also helps to bridge the gap between modern vehicles with AI features and older vehicles where these features are absent. I had to design the app in a way so that the delay between receiving input and producing a result is minimal. This was a key point to consider since it is meant to be used in a real traffic environment and any delay could cause an error leading to an accident.

Another aim was to minimise the amount of user interaction with the app. Audio feedback can be implemented so that it will not require the user to look at their device. The driver should be focused solely on driving, even when the app is being used. The setup and user interface had to be very simple in order to achieve this.

1.3 Project Challenges

There are many challenges involved in developing an application that will involve computer vision and artificial intelligence, while also incorporating design and third-party APIs. Below is a short discussion of the major challenges faced in the project.

1.3.1 Intensive Processing Tasks

Image processing involves tasks that need a considerable amount of processing power. This project revolves around using the Android device as the only component to carry out the processing of data. This made it hard to test the performance and accuracy of the app on lower and mid-tier phones. Higher end phones with more capable CPUs are needed to run smoothly. Fortunately, the Android API lets developers take advantage of threads, this helped to spread the task load and allowed the app to achieve more desirable performances on weaker devices.

1.3.2 Machine Learning

Inexperience in machine learning coupled with a lack of learning resources online for the specific method and data used proved to be a challenge which slowed development significantly in the early stages. Similarly developing a machine learning method that needed to run on Android only added to the challenges since this it was only recently supported.

1.3.3 Inexperience with Image Processing

Before starting this project, I had very minimal knowledge about computer vision and its libraries. This lack of familiarity in this area caused me to spend a significant amount of time trying to learn how to use the OpenCV library which may have impacted other parts of the project in a negative manner.

However, as the project progressed, I felt more confident using this library. The documentation available proved to be very useful getting me more familiar with how this library works and different approaches I can take to certain problems.

1.3.4 Limited Data and Inaccurate AI

Originally, I had planned to train a custom AI model to recognise different types of signs however, I could not find datasets specific to Ireland and had to create my own. My dataset had an insufficient amount of data which caused the AI to mislabel test images of signs. This high inaccuracy made it unusable in the app.

Next was to try using the MNIST handwritten digits dataset to train the model since the app only need to recognise speed signs containing only numbers. Unfortunately, MNIST focuses exclusively on handwritten digits in low resolution, making the use of it on the fonts of the signs difficult. This AI model also yielded poor results which could not be used. The failure of these two approaches forced me to find another method that will be capable of understanding the speed limit signs.

This problem was solved by introducing optical character recognition into the project. This technology allowed me very accurately extract the text from the signs, producing results that were far beyond what the other method was able to generate.

1.3.5 Delay Between Obtaining Data and Producing Results

The first draft of the project was to send data received from the phone and send it to a server that will perform the necessary processing. The advantage of using this method is that it would work on less powerful devices since it only needs to grab video data and send it to the server. The downside is the delay to get a result back. This external connection added several seconds to the data transfer. This method is unsuitable for the purposes of my project.

To get around this problem, I had to use a library, OpenCV, that is available on Android to allow the phone to process the data locally. Although this will impact the performance, it will allow the app to run in real time as there won’t be any data transfer delays.

1.3.6 Time Management

Due to the magnitude of the final year this degree, an increased effort was made for each module and as the weeks passed, goals in the project may have been obstructed by the continuous assessment assigned by each module. This challenge constantly presented itself throughout the development of the project however, as the project progressed, I gained more experience in this area and felt I managed to improve this skill.

1.4 Structure of Document

Research

The research section contains valuable research carried out throughout the course of the project. It was used to determine the problems and the technologies available to solve it after evaluating the advantages and disadvantages of each. It also reviews some existing solutions that cover a similar area of this project.

Design

This section will describe the design methodologies and approaches selected for the project. It goes through the prototypes of the app and details why some designs were chosen. Architectural and use case diagrams are also illustrated to give an insight as to what the application is composed of and its uses.

Architecture & Development

The architecture and development sections explore the implementation of the final project. It details how the app works, the APIs and libraries involved, and the key elements of the project. There will be an in-depth explanation as to how the app can differentiate different signs, get the speed, and detect lanes.

System Validation

Everything related to the testing or validation of the project is found in this section. Various types of testing are explored throughout the section that assisted in the proper evaluation of the application. Identification of features that can be demonstrated along with the screenshots of the features will be shown.

Project Plan

This section analyses and reviews how the project plan changed from the initial proposal to the deadline the reasons behind these changes. It also addresses how they would be confronted if the project was repeated.

Conclusion

A review of the project is found in the conclusion. It goes through the goals and whether they were met, and the reasons why. Key learning outcomes, recommendations and suggestions for how the work can be improved will also be discussed here.

Appendix

A user guide can be found in this section. It will include how to set up the device and what the different features available do.

2 Research

2.1 Background Research

Before starting the development of my project, it was important to understand how a self-driving car detects different objects, the technologies involved, and what is available for me to use.

The SAE International (Society of Automotive Engineers) has defined levels of automation for self-driving cars and what they mean (1). Fully autonomous cars (Level 5) that don't require any interaction from drivers are becoming more popular every year. Many companies have already started developing these vehicles and will be ready to deploy them into the real world within the next few years. There are also companies dedicated solely to autonomous vehicles, one of them being Waymo, the successor to Google's Self-Driving Car Project. The company has already deployed their designs onto public roads for testing and it is estimated that there will be 10 million of these vehicles on roads by 2020 (2).

Waymo's self-driving cars need an array of sensors to work properly. Three of the key sensors are the camera, a LiDAR (light emitting radar), and the radar. The cameras take images of the road that are interpreted by a computer. This allows the vehicle to differentiate between different objects but is limited by what the camera can "see". They also user LiDARs which send out lasers that reflect off objects. This can define road markings and works in the dark. Another key sensor is a radar. Using this, radio waves are sent out and bounces off objects. Radars can the distance the car is from the objects around it and can work in any weather but cannot distinguish between objects. These are also the standard amongst most self-driving vehicles today, they are essential in creating a fully autonomous vehicle (3,4).

For self-driving cars to process all of the data gathered at the same time, with minimal latency, they need a very powerful computer. Even partially autonomous cars need a considerable amount of computing power. For my project, the only sensor I will be using is the camera on the smartphone. Although this limits the features I can implement, features that only need video data, it can allow vehicles that aren't capable of autonomous functions to have similar ways to understand the environment around it and provide feedback helpful to the driver.

2.2 Alternative Existing Solutions

There are some apps on the Google Play Store that can be downloaded and used by anyone with an Android device. I found three applications that were designed to do some of the things I intend to implement in my project. They all run locally on the Android smartphone and do not require connections to any external devices. I think that the lack of requirements makes them very convenient to use, for example, if the device does not have a Wi-Fi connection, it would have no effect on the performance of the applications.

2.2.1 Traffic Sign Detector (5)

This app is designed to find and describe road signs using the phone's camera. It scans the current frame and looks for the signs. When a sign is detected it will extract a copy of it and pass it through a trained AI classifier to try to predict what it means. It will then provide a sound cue to let the user know the prediction it has made.

After some testing with this app using still images, I found that it can accurately detect a road sign but the definitions of them are not always right. For example, in the screenshots, it can correctly describe the '50' speed limit but labels the '80' sign as 50. This app would not be suitable for use in a real-world environment because of unreliable results.

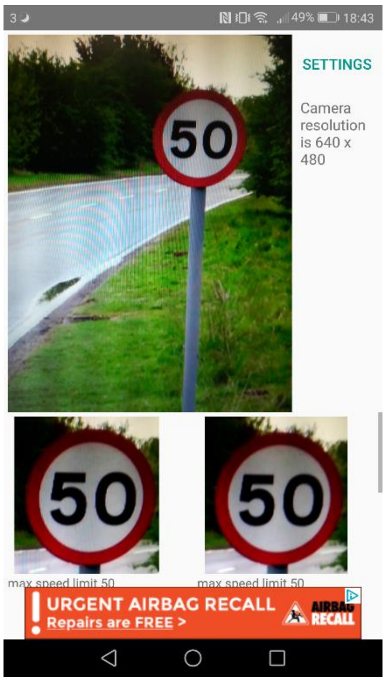


Figure 1 Traffic Sign Detector screen 1

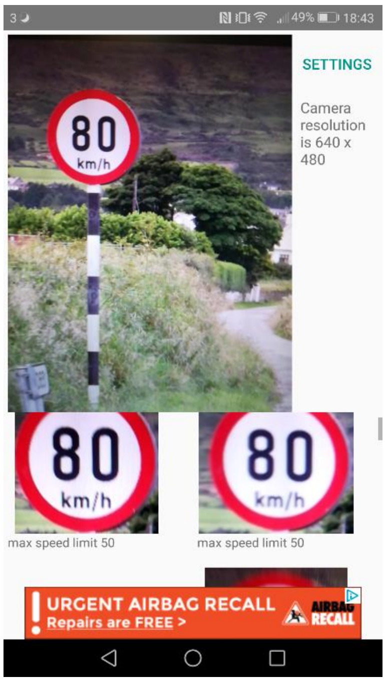


Figure 2 Traffic Sign Detector screen 2

2.2.2 Lane Detector (6)

This app uses the device to detect the lane a driver is currently in using video data from the camera. It will show where the lane's edges are and calculate the optimal position for the car. The red lines represent the outer lines of the vehicle's current lane, and the blue shows the centre. The turquoise box is the area that the app will perform the processing on, this can be adjusted by the sure with a simple tap on the screen to update the height of the window. This application works best with minimal obstructions in the lane but also tends to give false positives by detecting some random objects as lanes. It also lacks other features, the only function it can perform is identifying lanes.

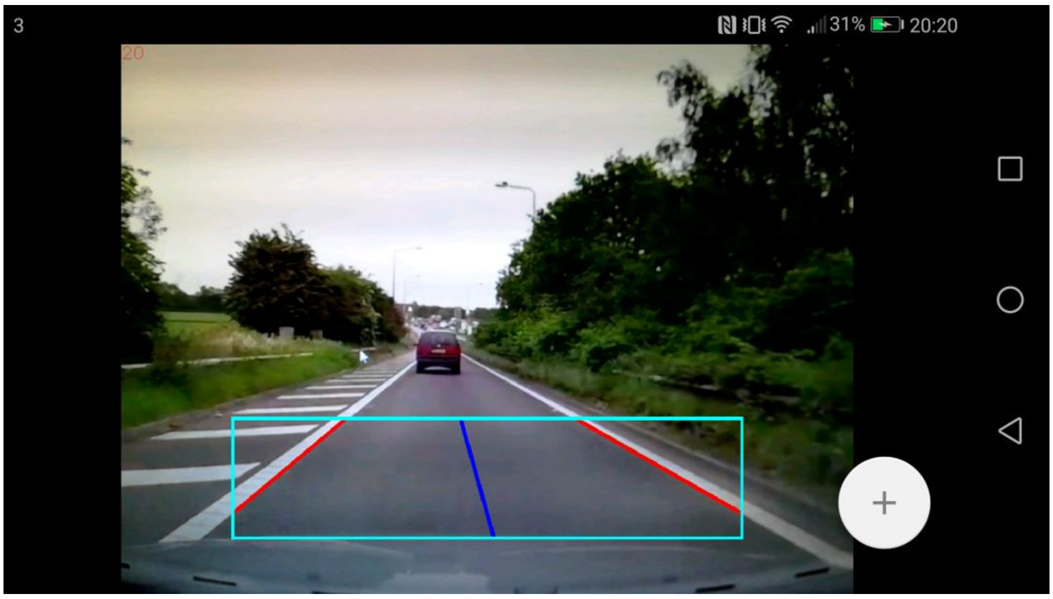


Figure 3 Lane Detector app screen

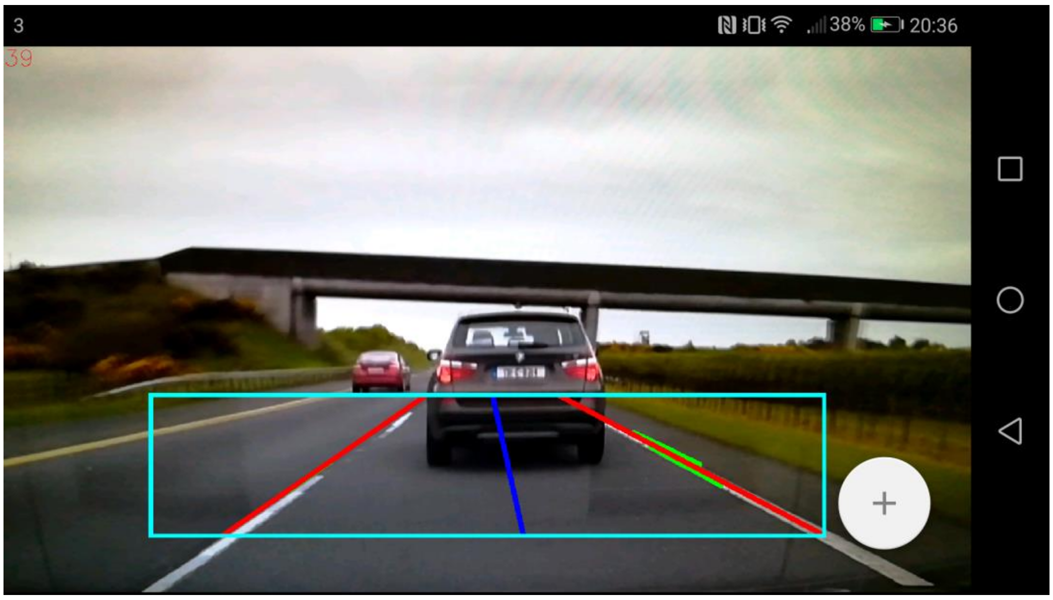


Figure 4 Lane Detector app screen 2

2.2.3 Lane Identification Pro (7)

This application is very similar to the other lane detection application. It also focuses solely on lane detection. The blue box marks the area that the app will use to find the current lane, the red and green lines mark the outsides of the lane, and the yellow area shows what the app thinks is the lane. It can automatically calibrate these settings to try to improve the performance.

The differences are that this app automatically sets the region to perform the detection on and will produce a sound alert when it thinks the driver is drifting into another lane. It also lets the user tweak the algorithm for detecting lanes if the auto optimisation doesn't work properly.

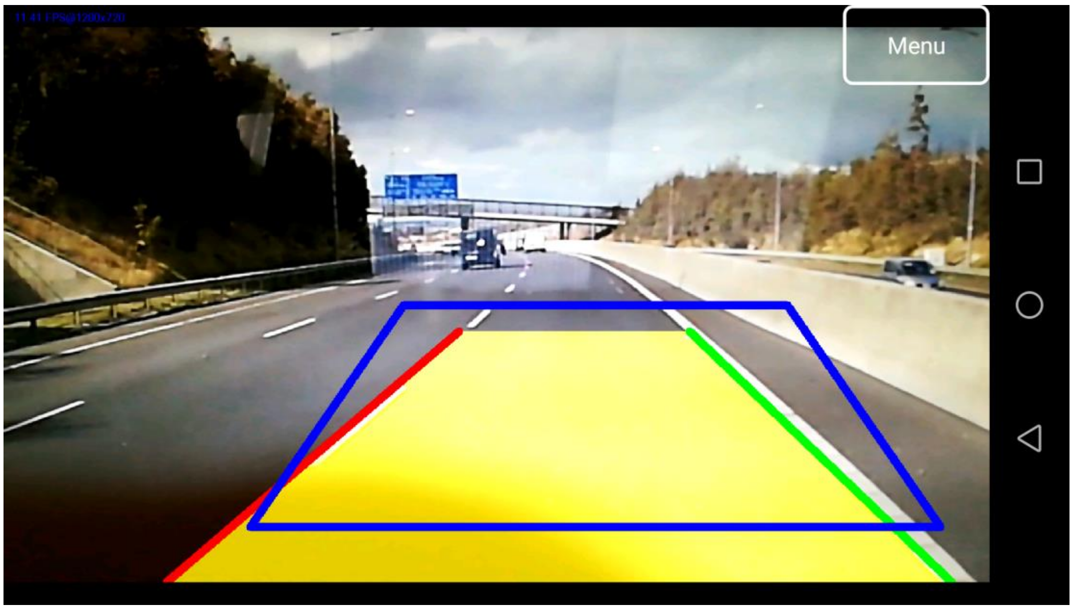


Figure 5 Lane Identifier Pro screen 1

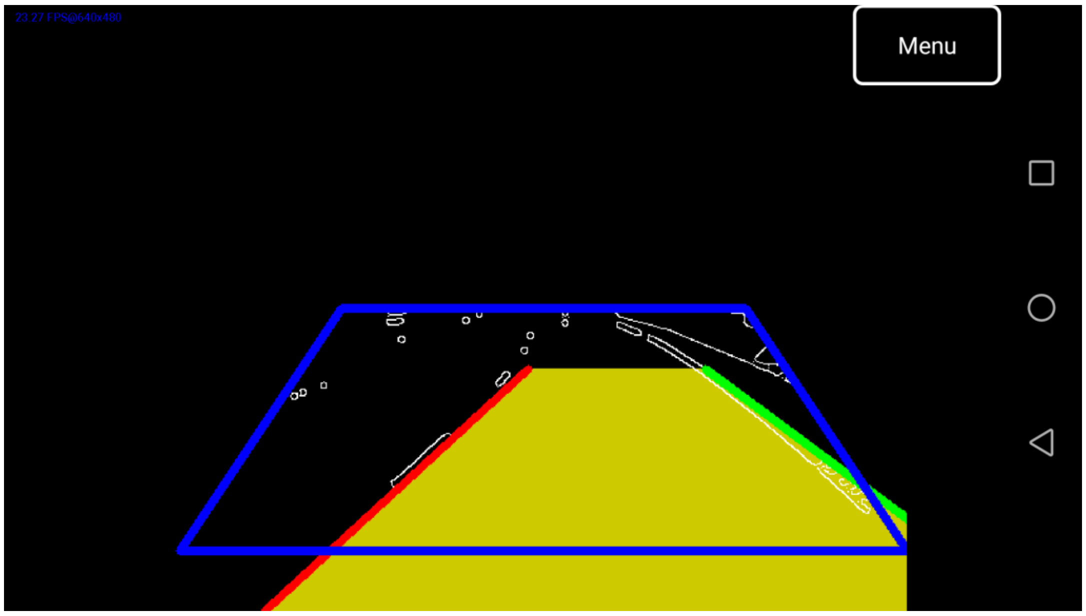


Figure 6 Lane Identifier Pro screen 2

2.3 Technologies Researched

2.3.1 OpenCV (Open Source Computer Vision)

OpenCV is a library that is mainly aimed at real-time computer vision. It is compatible with multiple programming languages and supported operating systems. I will be using this since it is the most popular open source image processing library available. There is plenty of existing documentation online which will make it easier to find solutions to problems I may run into while using OpenCV.

OpenCV has an SDK available for Android (8). This will allow me to develop an application that can process the image data natively on the device. Doing this will eliminate the need to send the data to an external medium such as a server for processing. It will also reduce the delay between retrieving data and providing feedback to the user at the cost of requiring more computing power from the device.

However, running OpenCV on Android either requires the device to have the OpenCV Manager application installed (8) or requires embedded OpenCV .so files (compiled C or C++ libraries) into the application.

The advantage of using OpenCV Manager is that if there are several apps based on OpenCV, the total occupied storage will be less than having internal libraries since only one instance of this app is needed. The disadvantages are that it’s not available on the Play Store anymore, and if the user does not have the Manager installed an OpenCV app will not work. As a result, I will have to embed the required OpenCV libraries into my application. This will increase the storage size required by the application but gets rid of dependence on a separate application.

2.3.2 JavaCV

JavaCV uses wrappers from commonly used libraries by researchers in the field of computer vision, including OpenCV, and provides utility classes to make their functionality available to use on the Java platform (9). The JavaCV API can also be used for Android development but for the purposes of this project, many of the libraries included will not be used. A disadvantage of using this API is that it lacks documentation from the developers and communities who use it. If I were to implement this in my project, the drawback could slow down development stages if problems are encountered. Similarly, it also requires the OpenCV Manager application to run.

2.3.3 Android

Android is responsible for providing functionality to millions of mobile devices all over the world. It is the largest mobile platform that currently dominates the market with about 75% of devices using Android (10).

Android provides developers with its own developer kit, Android Studio, which uses Java programming language. Java programming language is the preferred language for Android studio however, it also supports C and C++ programming languages (11). Android Studio also comes with a set of built-in tools required for development and testing that makes the development work considerably easier. Android Studio allows developers to create multiple emulators of Android devices. These devices can have different versions of Android that have been made available to the public (12). I can use this to test my application on different versions that are older or newer than the version of my actual testing device and assess the compatibility.

I think developing this project in Android would be better since I have experience with Android and other operating systems, such as iOS and Windows, don’t have nearly as much presence as Android on the mobile market.

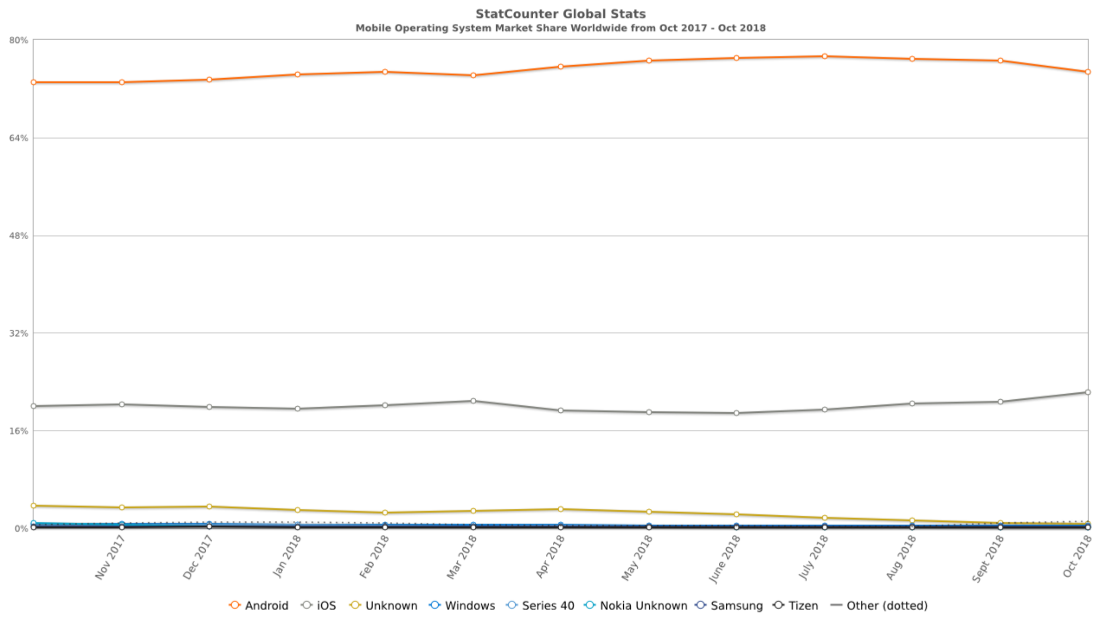


Figure 7 Mobile OS market share

2.3.4 Android SDK Versions

For choosing the version I want to develop for, I have considered the number of phones running each version of Android and the libraries available for each version. I have chosen the minimum API level to be 6.0 with SDK level 23 to maximise libraries and utilities I can use while still targeting a large market. Since around 73% of Android devices are using Android 6.0 or above, this will make the application available to most of the users (13).

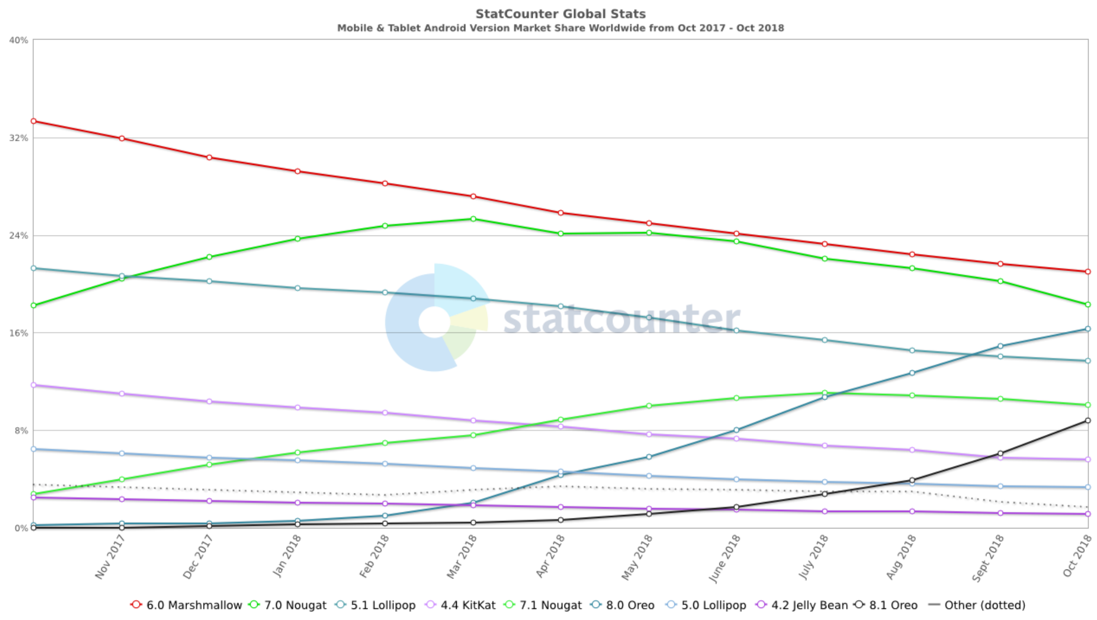


Figure 8 Android version market share

2.3.5 Optical Character Recognition

Optical character recognition (OCR) is the conversion of images of handwritten, typed, or printed text into a digital format that can be used by computers. This

involves scanning the text character-by-character and translating it into character codes such as ASCII.
Before OCR is carried out, some pre-processing must be done on the image. This includes straightening the text, removal of non-text objects like dust or random lines and converting to grayscale. This makes the document cleaner and easier for the algorithm to detect characters accurately.

The next step of OCR is to apply a feature matching on the binary image. It splits each character into individual lines that makes it up and compares it against a database that contains all types of fonts. The one with the highest match will be shown in the result.

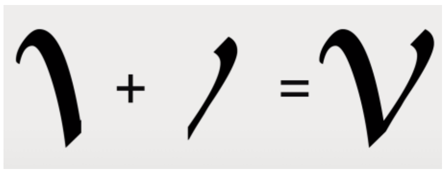


Figure 9 OCR example 1

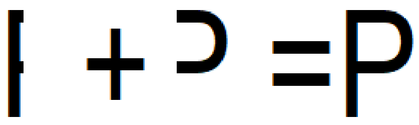


Figure 10 OCR example 2

In relation to my project, this technology would be very useful for understanding what some road signs mean. There are many APIs that are supported by Android like Tesseract OCR and Google Mobile Vision (14).

2.3.6 TensorFlow

TensorFlow is an open-source library developed by Google with strong support for machine learning. It is compatible with multiple operating systems and languages including Python, Windows and Android which I plan to use during the development of my project.

TensorFlow allows developers to create artificial neural networks. These neural networks are intended to replicate how humans learn. They consist of input and output layers, as well as hidden layers that transform the input into data the output layer can use. These neural networks can be trained to understand and predict data using this library. It also allows the trained models to be imported into another OS such as Android.

One of the best features of TensorFlow is that a GPU can be easily integrated into the training process of a neural network. This will reduce training times significantly on large datasets (15) due to large memory bandwidths. A possible use for this would be to train a model that can recognise different road signs, and then embed it into the Android application.

2.3.7 Google Compute Engine

Google Compute Engine is a cloud platform that provides infrastructure to its users. It offers many services such as web hosting, database storage, user storage, networking etc.
This platform was considered for this project because it allows me to host a server that would be programmed to take video data directly from an Android phone. It would then process it and send the output back to the device, so it can display the results to the user. Another major reason is that it offers NVIDIA GPU instances. This

could be used for the machine learning part of my project. Since training deep learning models contains intensive compute tasks on large datasets that will take days to run on a single processor, the use of these GPUs can significantly accelerate the training process by reducing to hours instead of days (16). Though this will be faster than training on my own computer, the price to run these instances can get very expensive for better performing hardware (17).

Another possible use for Google Compute Engine was to create a virtual machine on the platform as a server. This server would listen for connections from the mobile device for video data, process it, and send the result back to the phone. However, after testing the latency between sending some data to the server and receiving results on the client, I decided to not include the use of servers in my project. This transfer of data took about 9 seconds to complete, and in order to minimize the delay, the application needs to process data locally. I want my application to provide feedback in real-time based on the current state of the environment, so I created a simple OpenCV app to test the speed of processing all the data on the device. This app was able to take data from the camera and perform some simple manipulations, and then return the results in about 0.5 seconds.

2.3.8 CUDA Module for OpenCV

Originally, I had planned to use this CUDA module alongside OpenCV. In order to benefit from this library, a Nvidia GPU is needed. CUDA allows developers to use the OpenCV framework to greatly increase the performance by running the processing on the GPU instead of the CPU (18). Using this would have allowed me to process more video data in the same amount of time because the GPU can carry out calculations at a much higher frame rate.

Unfortunately, deciding to develop natively in Android rules out the use of this module for me since there are no Android devices that use a CUDA compatible GPU. Doing this will reduce the performance of my project but it is necessary since it needs to have minimal latency between receiving input and producing the output. This will be achieved by not having to send data to be processed by an external GPU but instead use the Android’s onboard CPU.

2.4 Other Relevant Research

2.4.1 Android Threads

When an application starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single execution thread. All components an application run in the same thread by default. This thread is called the ‘main’ or ‘UI’ thread as it is responsible for updating the UI as the app is being used (19). So, if this thread is used to process multiple aspects of the app, it will be slowed down and might crash. Using Android Threads will allow processes to be sent to a worker thread to carry out the operation in the background. This frees up the UI thread for focusing on the interface and other tasks.

2.4.2 Design Methodologies

For this project, I have considered two possible methodologies to implement, Agile and Waterfall. Waterfall is a traditional method and works in a sequential process. Developers work on each step continuously and only move to the next if the previous has been perfected. If there is a feature that needs to be added, they must start from the beginning. This is a very straightforward model to follow that can be very beneficial for simple products that will not encounter a need for change to the original design (20). The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

The other methodology is the Agile Methodology, specifically the feature-driven approach. It is one of the most popular methodologies in use in software development. Agile development methods allow for the project requirements to develop and evolve as time goes on, and change is expected. Agile depends on customer collaboration, this aspect is important as it allows for design changes over time as features are included (21).

2.5 Resultant Findings and Requirements

2.5.1 Findings

Following the research completed, this section will cover the reasons why the technologies were chosen. The application will be developed natively for Android using Google's official integrated development environment Android Studio. Benefits of this IDE are that it allows virtual Android devices to be created so that the application can be tested on multiple versions of the Android OS. It also comes with a plugin for the build system Gradle, which lets developers easily generate APKs (application package used to install an app) that can be tested, deployed, signed, and distributed (22).

Between JavaCV and OpenCV, I have decided to use the OpenCV library for Android. The library was originally released in 2011 (23) and the developers have been continuously supporting and updating it. This support comes in the form of documentation and communities that actively participate in exchanging information. JavaCV does not have this to the extent of OpenCV and if I were to this instead and encountered a problem, I would most likely have a harder time coming up with a solution making it the less viable option for image processing. Another reason is that while trying to set up both libraries for some tests, OpenCV proved to be easier and faster to set up.

In order for the app to be useable in a real-world situation, there should be minimal delay between getting data and returning a result to the user, this means that there will be no external connections implemented. The Android application will process all the data locally on the device and will not need to send data to any servers. At the cost of needing a more powerful CPU to run the app, the app won't need to wait for a response from a server. This also has the added benefit of being able to run without needing a Wi-Fi connection.

For the sign recognition aspect, I feel that OCR is the better option after carrying out some accuracy tests with OCR and trained TensorFlow AI models. Due to the lack of training data, the AI performed poorly. This could be improved but will require a larger dataset of training data, i.e. pictures of speed limit signs, which are not readily available. I would have to gather a lot of images myself which will take a lot of time. Since I only need the app to recognise numbers on the signs, OCR will be enough to achieve this. It is intended for use on images of text and is far more accurate in my testing than a custom AI model.

2.5.2 Requirements Table

The table below outlines the set of requirements for the application. Each requirement is given a stage and a priority. The stage corresponds to the stage it was implemented in the development cycle. The priority defines how important it is to the overall project and the general order I will take to implement them into the project.

Req. Number	Stage	Name	Description	Priority
1	1	Speed Sign Detection	Enable app to detect and interpret speed limit signs	HIGH
2	1	Lane Detection	Enable app to detect the bounds the vehicle's current lane	HIGH
3	1	Get vehicle speed	Enable app to get the speed of the vehicle using GPS	HIGH
4	2	Audio feedback	Using the retrieved data, audio alerts will be produced to inform users. i.e. new speed limits	HIGH
5	2	Calculate limits	Determine if the driver is within limits. i.e. not over the speed limit and drifting into another lane	HIGH
6	2	Visual Feedback	Provide some visual feedback for the user. Display the detected speed limit on screen and highlight the current lane	HIGH
7	2	Optimisation	Optimise the app so that it will run at a reasonable frame rate and lower battery consumption	MEDIUM
8	3	Resolution options	Allow users to change the resolution of the camera to accommodate different tier devices	MEDIUM
9	3	GPS options	Allow users to disable the GPS service to stop the app from getting speed (lower battery consumption)	MEDIUM
10	3	Sound options	Allow users to disable the audio alerts	LOW

Table 1 Requirements table

3 Design

3.1 Approach and Methodology

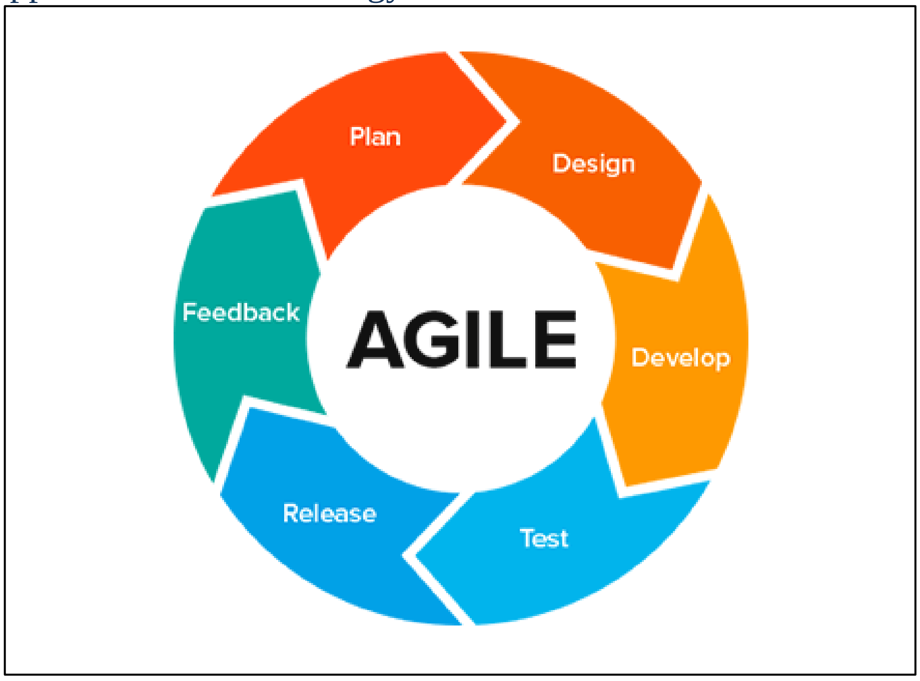


Figure 11 Agile methodology diagram

The methodology I am implementing in my project is an agile model. I feel that a feature driven development model is best suited for my project as it focuses on delivering working features repeatedly at regular intervals. The repetition allows for more thorough designs, implementations, and can provide a better way of exposing major flaws before they manage to make their way into the system. In addition, the flexibility of this methodology will allow me to easily go back to previous designs and features and adjust them if needed.

Design and code will be delivered in phases. As a new feature is ready to be implemented the design can be drawn up and then developed, with redesigns likely to occur during the development. Completing all design up front could potentially result in a loss of time because aspects chosen at the time may not be needed further down the development cycle or may need to be redesigned. This will also allow me to keep working on and testing a feature until I feel that it performs well. Then I can start development on the next feature and repeat until all aspects have been completed.

The approach for the project was firstly to carry out some general research that would help me understand what would be involved, this was discussed in the research section. Then I will design the overall UI, use cases, and the architecture. Next was to start development on one of the highest priority features as outlined in the requirements table (*Figure 11*). The feature being worked on will be iterated over until it passes the tests created for it. When a feature is complete, I will move to the next unimplemented feature with the highest priority. I plan to continue this cycle until all the requirements have been satisfied. If this is completed before the due date, I plan to use the extra time to revisit and redesign the user interface if needed.

3.2 Use Cases

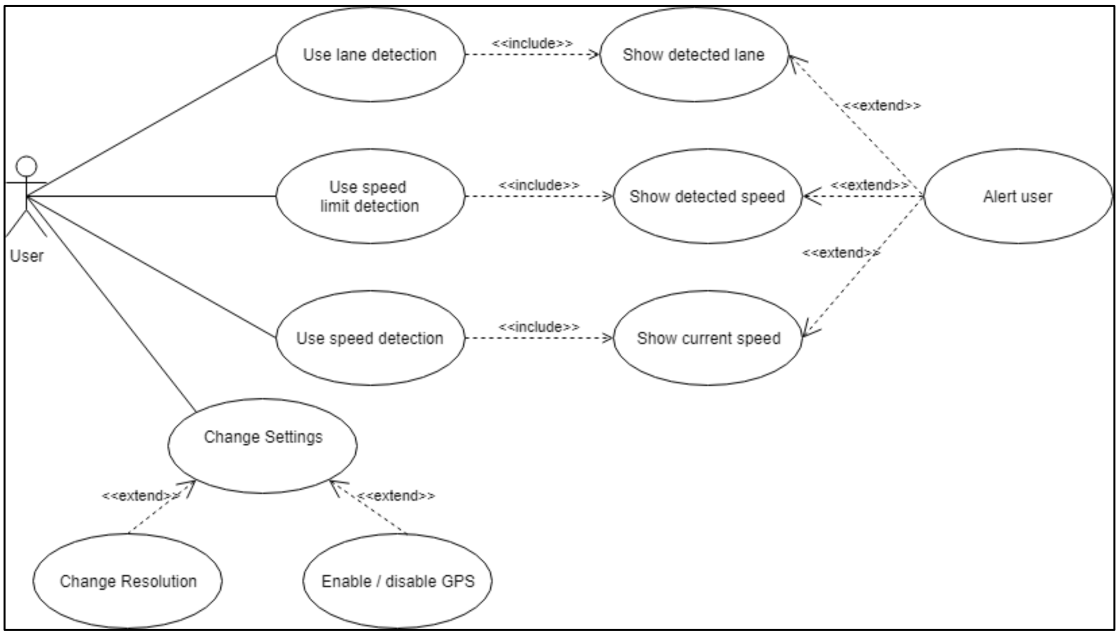


Figure 12 Use case diagram

The use case diagram above outlines what features are available to the user. The main features are lane, speed, and speed limit detection. They will also provide visual feedback as demonstrated by the 'include' relations. All the detection components will also have the option to alert the user by using audio signals as shown by the 'extends' relations. Another use case would be the user's ability to change settings within the application. They would be allowed to change the resolution of the camera (to accommodate devices with different processing capabilities) and enable or disable the use of GPS (uses less power is disabled but cannot get the speed of the vehicle).

3.3 User Interface Design

3.3.1 Low Fidelity Prototype

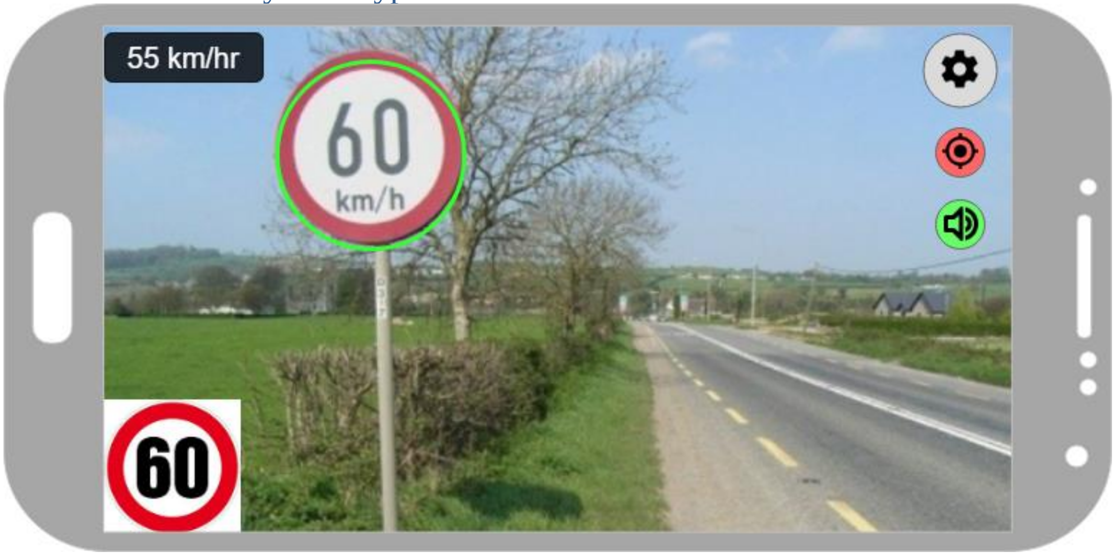


Figure 13 Prototype design for sign detection

The figure above is the prototype design for the interface of detecting road signs. I decided to go with a simple design with minimal clutter. When a sign is identified, a picture the sign will be displayed at the bottom left of the screen and there will a sound alert e.g. “Speed Limit: 60”.

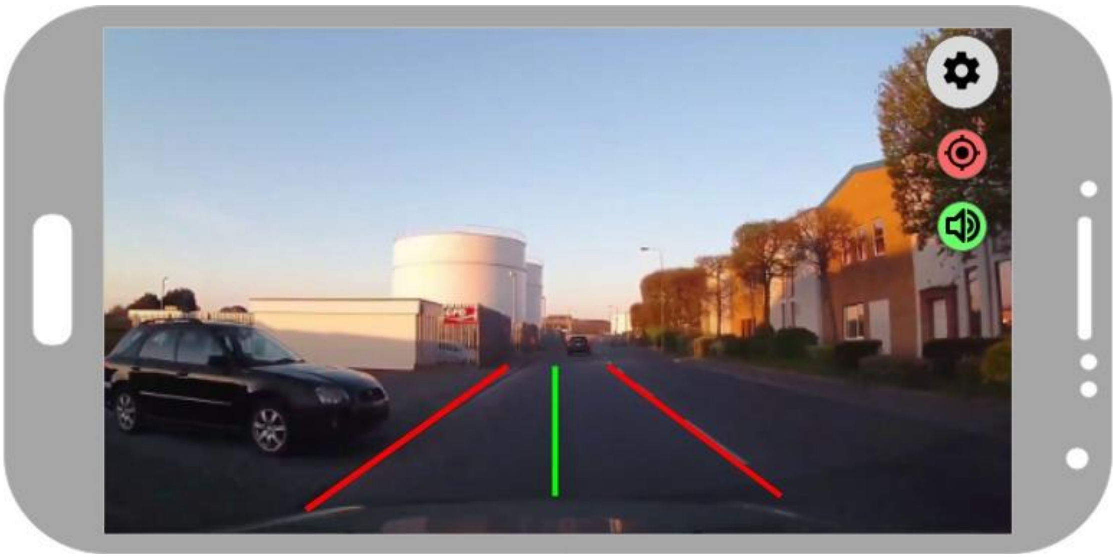


Figure 14 Prototype for lane detection

For the lane detection interface, I wanted a clean look. The red lines mark the outside of the lane and the green marks the middle. Another possible design is to fill the region being processed with a solid colour, I feel that both are viable options because they are plain and uncluttered.

3.3.2 Final Design

Since the main objective is to create an app that won’t need user input when the app is set up, the UI will be very simplistic. In the final design, there is an expandable options menu in the top right corner of the screen, this shows the user different settings they can adjust. The buttons use icons instead of text to represent different options.

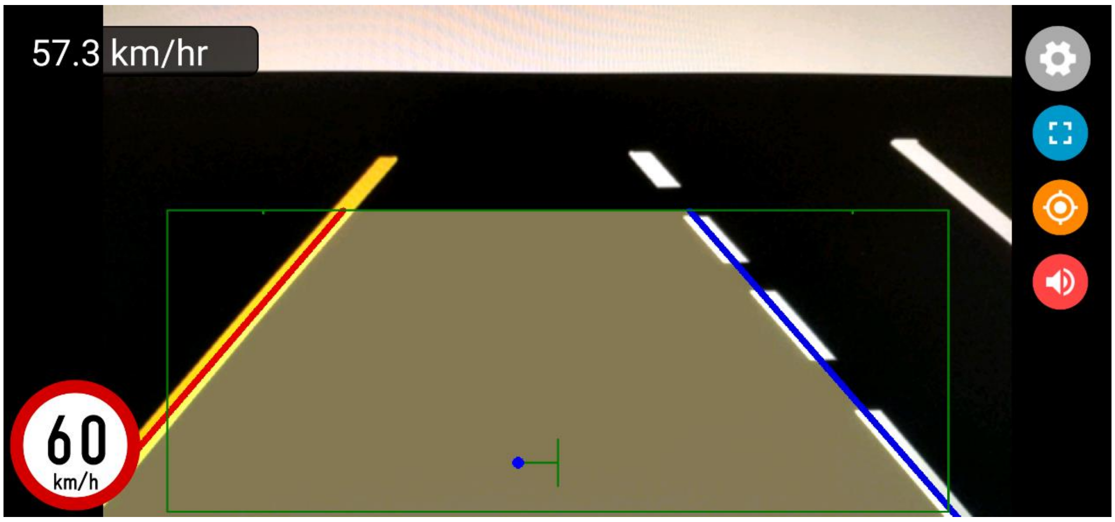


Figure 15 Final screen design

There is a simple box outlined in green to show where the user should place the phone so that lanes appear in that area. When a lane is detected, the left will be shown with a red line and blue will be shown for the right. The blue dot serves as the centre of the detected lane with the vertical green line as the centre of the vehicle. An opaque white layer is used to highlight the valid lane for the driver.

The speed limit is shown in the bottom left corner of the screen and will update depending on the signs are found. The speed of the vehicle is shown in the top left corner and changes every second.

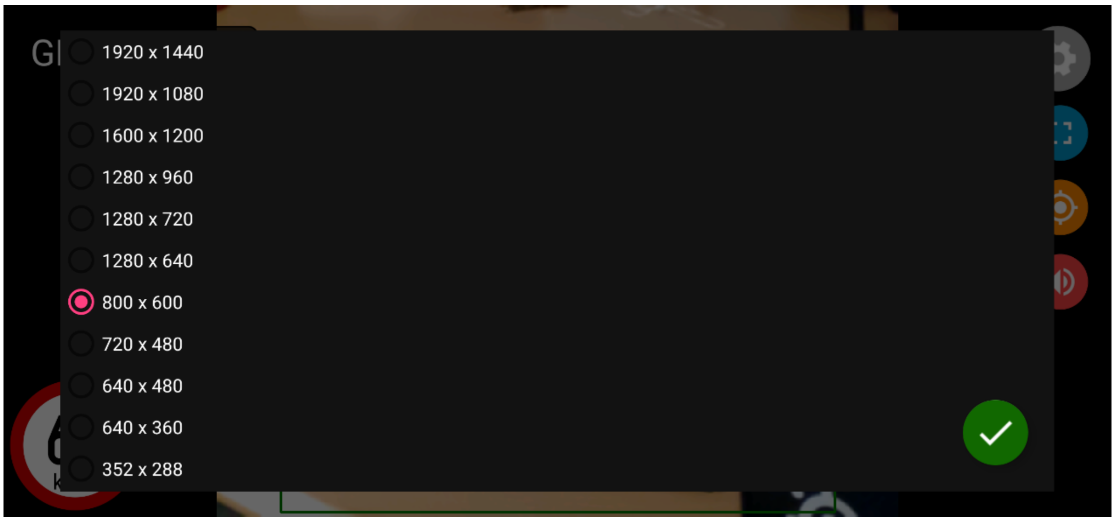
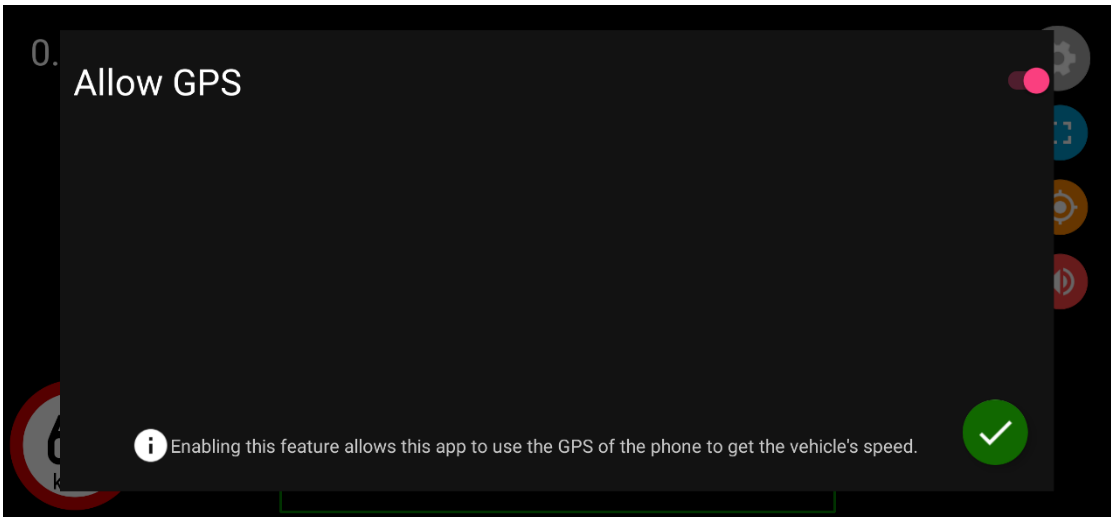


Figure 16 Resolution selection screen

When the user presses the blue button, a new screen will be displayed. This lists the supported resolutions of the device's camera and the current resolution and allows the user to change it with a single click.



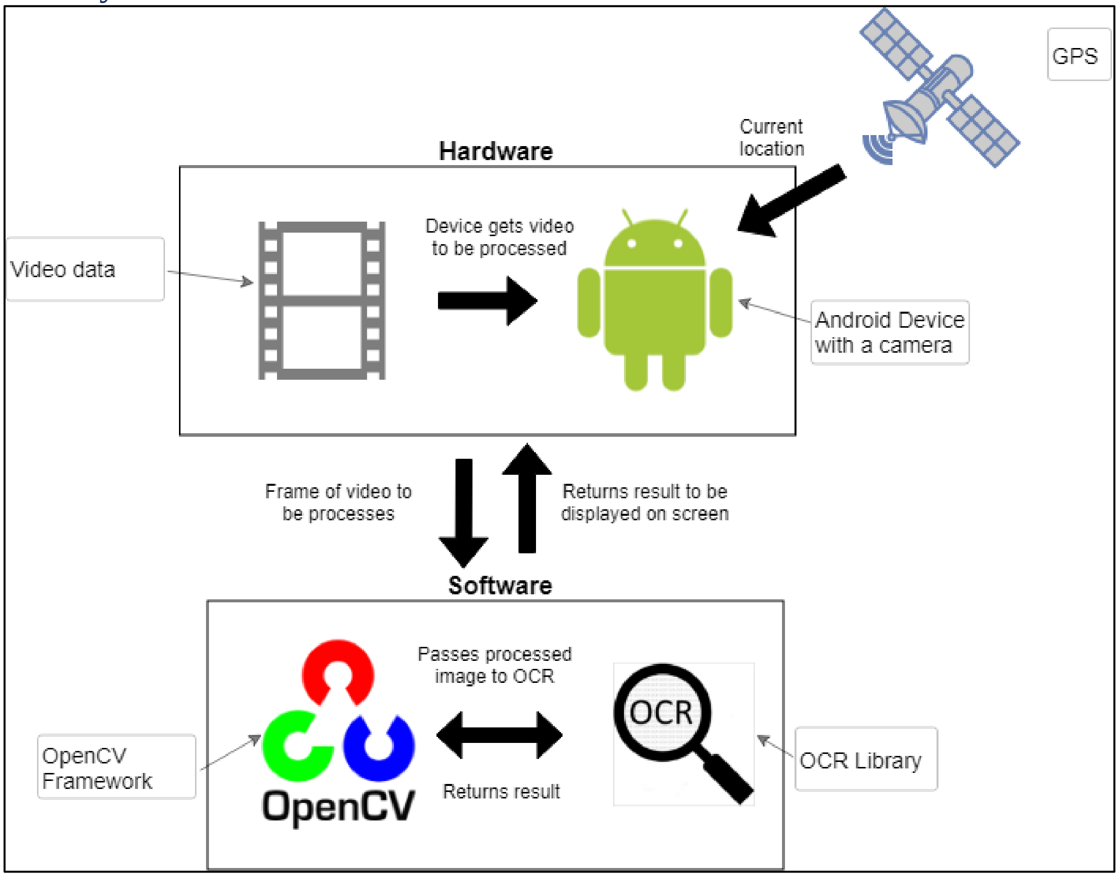
Enabling this feature allows this app to use the GPS of the phone to get the vehicle's speed.

Figure 17 GPS screen

The orange button opens the GPS screen. It lets the user enable or disable the GPS and provides information about the feature. Figure 15 and 16 both show a green check button which allows to user to easily exit to the main screen.

4 Architecture & Development

4.1 System Architecture



GPS
Current location
Hardware
Video data
Device gets video to be processed
Android Device with a camera
Frame of video to be processes
Returns result to be displayed on screen
Software
Passes processed image to OCR
OCR
OpenCV Framework
OpenCV
Returns result
OCR Library

Figure 18 Technical architecture diagram

The technical architecture of the application is all contained within the device itself. There are no connections to external components such as servers. This minimises the time it takes between retrieving data and producing results. The architecture can be divided into two distinct parts:

- Hardware – camera and GPS
- Software – APIs and SDKs used

The hardware part of the system is used to gather information to be processed. They are the device's camera and GPS. The camera grabs video data that will be processed for results. The device communicates with the GPS to get the location of the vehicle which is needed to calculate the speed.

The camera continuously captures frames that will be dealt with by the OpenCV framework. It searches the frame for object that looks like speed limit signs and crops that section. It sends that new image through an OCR operation which will try to extract the contents on the sign if present. OpenCV is also used for lane detection, it searches for white and yellow lines within the region of interest (green rectangular box), as shown in Figure 16. If lanes are found, the library is used to highlight the results.

4.2 Project Setup

The project required the following hardware and software items:

- Computer / Laptop
- Android devices (different OS versions)
- Android Studio
- GitHub
- GitKraken
- OpenCV 3.4.3 for Android
- Google Mobile Services
- Google Play Console

The laptop and Android device were required for the development and the testing of the application.

Android Studio IDE was installed and used to develop and build the application using the Java programming language. Virtual devices were also created using this IDE for the application testing on Android versions I don’t have physical access to.

The GitHub account and repository were used for the control of the development process and code versions developed.

GitKraken is a Git client used to push all code changes to the GitHub repository.

OpenCV is the image processing library that will is used for lane and sign detection.

Google Mobile Services is an API that is used for character recognition of text on speed limit signs.

Google Play Console is used to publish the application to the Google Play Store.

4.3 Project Components

The following section details some of the files, methods and Classes within the project that generates the lane, sign and speed detection functionality and provides a usable experience to users of the application. For most, a short description of the functionality of the Class is all that is required however, there are Classes that deserve a more detailed discussion where the complexity, challenges and solutions will be outlined. The following list is ordered based on the priority specified in Table 1, Requirements table, from highest to lowest.

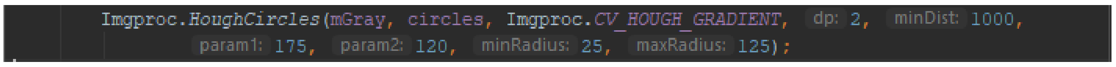
4.3.1 MainActivity.java

An activity in Android is the presentation layer of an Android application, e.g. a screen which the user sees. An Android application can have several activities and it can be switched between them during runtime of the application. This class is the main screen the user will see. It shows the results of the lane and sign detection and provides audio feedback is needed. It allows the user to navigate to the settings activities.

On the first launch, the app sets all settings to a default value that can later be changed by the user. It also checks for permissions it needs to use the device's camera and location and gives the user an explanation as to why.

4.3.1.1 Speed Sign Detection

This section will explain the process the application takes to detect and alert the user of a speed limit sign. The following code runs inside the onCameraFrame() function which will be called continuously from when the app is started. Every frame of the video, the following code will be executed:



```
Imgproc.HoughCircles(mGray, circles, Imgproc.CV_HOUGH_GRADIENT, dp: 2, minDist: 1000,
        param1: 175, param2: 120, minRadius: 25, maxRadius: 125);
```

Figure 19 Circle detection method

Using the OpenCV library, the method above is using on a pre-processed frame in which image noise reduction and conversion to grayscale had taken place. It tries to find circles by looking at the colour value of each pixel and the ones adjacent to it. If circles are found, that area will be cropped and passed to the method 'analyzeObject' shown below.

```
public void analyzeObject(final Mat img, final Rect roi, final int radius) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            isRunning = true;
            Mat copy;
            try {
                copy = new Mat(img, roi);
                // Creates a bitmap with size of detected circle and stores the Mat 
into it
                bm = Bitmap.createBitmap(Math.abs((radius * 2) + 20),
Math.abs((radius * 2) + 20), Bitmap.Config.ARGB_8888);
                Utils.matToBitmap(copy, bm);
            } catch (Exception e) {
                bm = null;
            }

            if (bm != null) {
                Frame imageFrame = new Frame.Builder().setBitmap(bm).build();
                SparseArray<TextBlock> textBlocks = 
textRecognizer.detect(imageFrame);

                for (int i = 0; i < textBlocks.size(); i++) {
                    TextBlock textBlock = textBlocks.get(textBlocks.keyAt(i));

                    if (!signValue.equals(textBlock.getValue())) {
                        signValue = textBlock.getValue();
                        setUISign(signValue);
                    }
                }
            }
            isRunning = false;
        }
    };

    if (!isRunning) {
        Thread textDetectionThread = new Thread(runnable);
        textDetectionThread.run();
    }
}
```

Figure 20 AnalyzeObject method

This method creates a ‘Runnable’, which is code to be executed on another thread, so the main thread won’t have to wait for this method to finish. This prevents the feedback on screen from lagging. If a circle is detected and the thread is not already running, the cropped image will be converted to a bitmap. Then it is passed through the OCR framework to retrieve any text in the bitmap and depending on the result, the app will alert the user and update the UI.

4.3.1.2 Lane Detection

Like the speed sign detection functionality, the feature runs inside the onCameraFrame() function. It also uses a pre-processed frame after image noise reduction. The image is then split into different colour spaces, RGB and HSV, getting the Red of RGB and the Value of HSV. This converts them to grayscale, the brighter a pixel is, the more of the value is detected.

```
    public void splitRGBChannels(Mat rgb_split, Mat hsv_split, Mat hls_split) {
        List<Mat> rgbChannels = new ArrayList<>();
        List<Mat> hsvChannels = new ArrayList<>();

        Core.split(rgb_split, rgbChannels);
        Core.split(hsv_split, hsvChannels);

        rgbChannels.get(0).copyTo(mRed);
        hsvChannels.get(2).copyTo(mVal_hsv);

```

Figure 21 Splitting image into different channels

In Figure 21 a yellow line is present, in the RGB colour space red and green makes up this colour so in Figure 22 that line is shown as white, meaning a high amount of red was detected in that area. The same concept is applied to the HSV split shown in Figure 23 below. The diagram in Figure 24 shows that both yellow and white will have high Value in HSV making them perfect for lane detection since all road lane markings are one of these two colours.

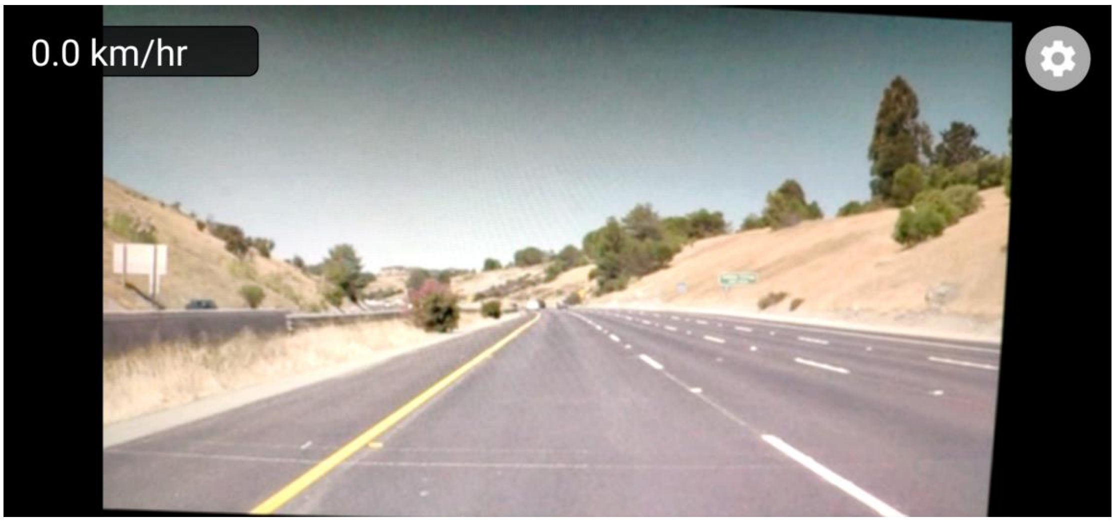


Figure 22 Normal frame before split

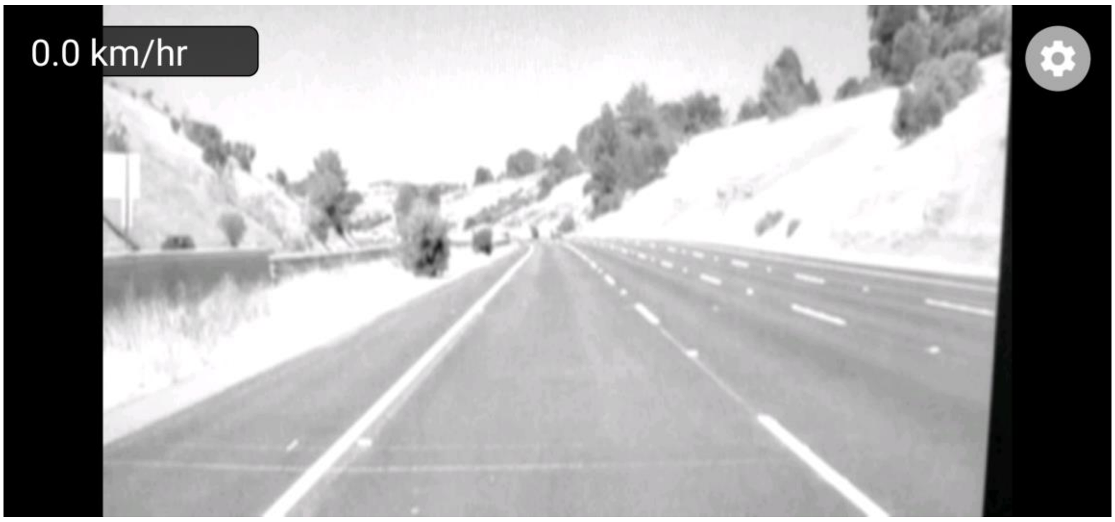


Figure 23 Red channel of RGB

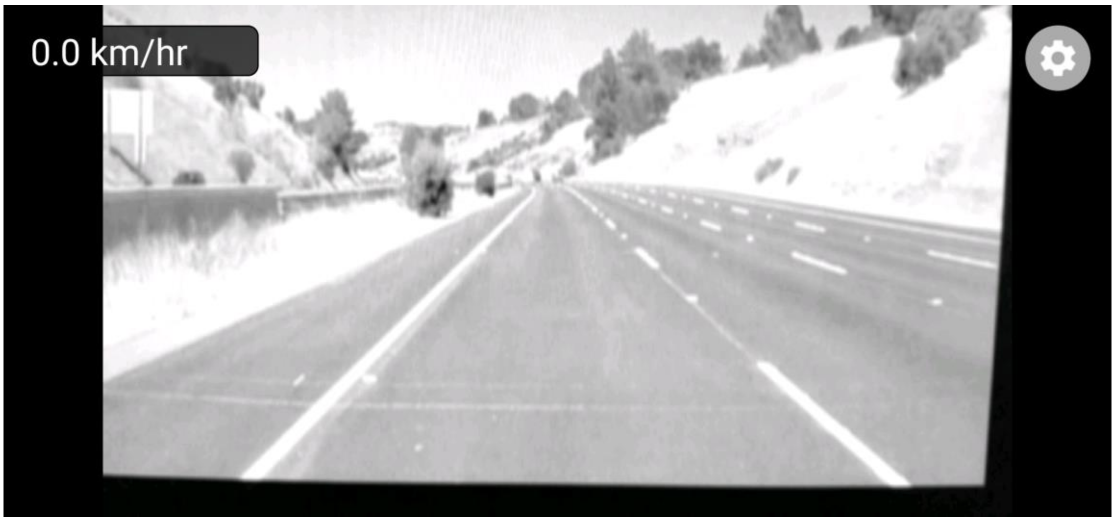


Figure 24 Value channel of HSV

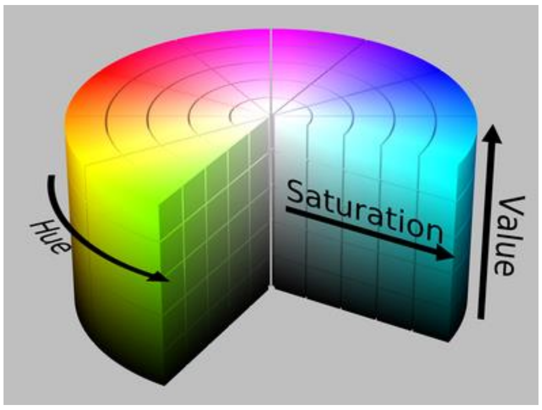


Figure 25 HSV colour space

After extracting the two colours, a mask is then created. The mask turns all values into either pure black or pure white. The code below checks if the frames produced have values within certain thresholds and will only show those pixels in the mask. This is used to reduce interference from other objects that are not lanes.

```
public void applyThreshold() {
    Scalar lowerThreshold = new Scalar(210), higherThreshold = new Scalar(255);

    Core.inRange(mRed, lowerThreshold, higherThreshold, mRed);
    Core.inRange(mVal_hsv, lowerThreshold, higherThreshold, mVal_hsv);

    Core.bitwise_and(mRed, mVal_hsv, mask);
}
```

Figure 26 Applying mask to split channels

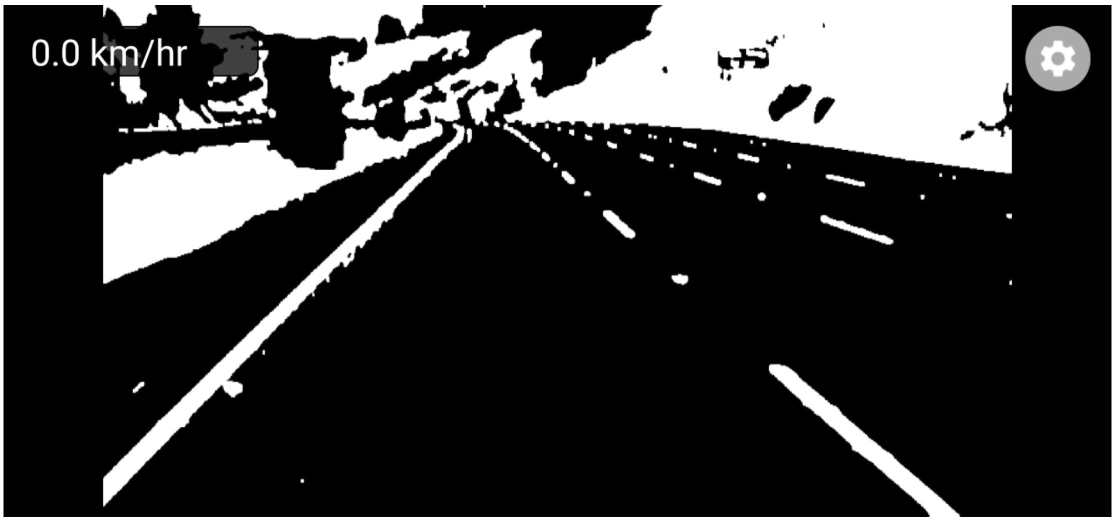


Figure 27 Mask result

From this, the app tries to find lines that are within a certain region which could represent lane lines. Those lines are then checked if they are within a certain range of slopes. This is because the lanes on either side will always look like they are converging towards a vanishing point from the perspective of the driver. This is illustrated in the figure below. Setting a limit will reduce the number of false positives for lane lines and produce more accurate calculations.

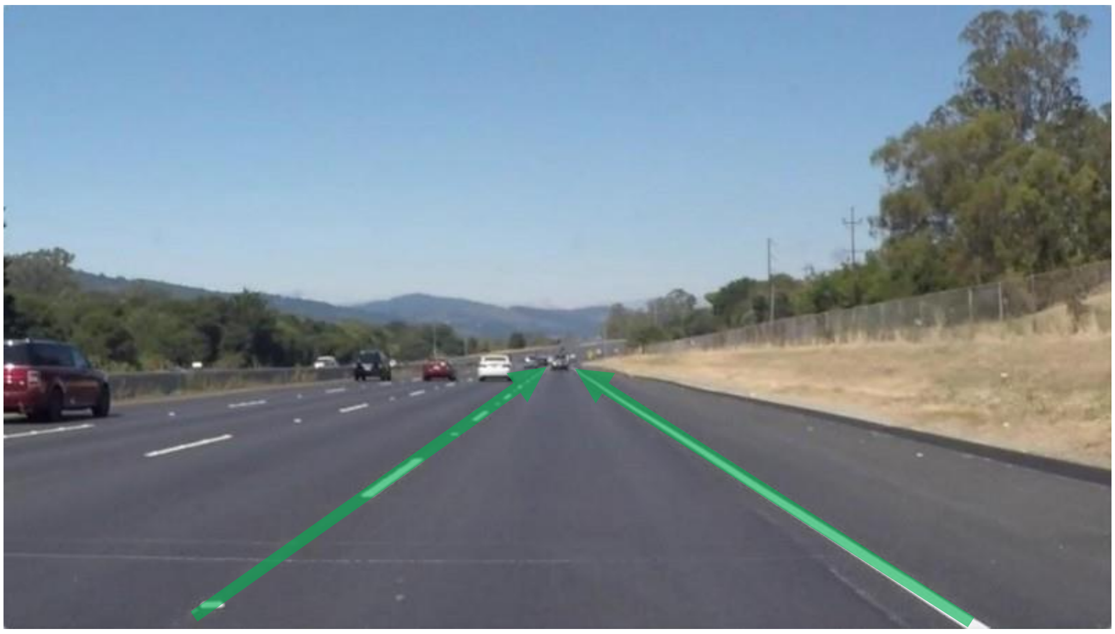


Figure 28 Lane vanishing point

4.3.2 SignUiRunnable.java

This class is used to update the image on the bottom left of the screen is a new speed limit is detected. It implements the Runnable class, a built-in Android class, which allows it to run on a different thread. There is only one instance of this class which is created every time the app is started.

4.3.3 LocationService.java

This class acts as a 'Service' or a task that runs in the background of the application. This service is responsible for getting the location of the device using the GPS, which is used to calculate the speed of the vehicle. At timed intervals, this service broadcasts the speed back to the main activity which will display it on the screen for the user. The code below shows that the location should be retrieved every 1.5 seconds and broadcasted to the main activity.

```
/* A service that runs in the background to get location updates and send it to the
main activity */
public class LocationService extends Service {
    private static final String TAG = "LocationService";
    private LocationManager locationManager;
    private static final int LOCATION_INTERVAL = 1500;
    private static final int LOCATION_DISTANCE = 0;

    private class LocationListener implements android.location.LocationListener {
        Location lastLocation;
        double speed, distance;
        long curTime, prevTime;

        LocationListener(String provider) {
            Log.e(TAG, "LocationListener " + provider);
            lastLocation = new Location(provider);
        }
         .
         .
         .
         .
    }

    LocationListener[] locationListeners = new LocationListener[]{
            new LocationListener(LocationManager.GPS_PROVIDER),
            new LocationListener(LocationManager.NETWORK_PROVIDER)
    };

    @Override
    public void onCreate() {
        Log.e(TAG, "onCreate");
        initializeLocationManager();
        try {
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
LOCATION_INTERVAL, LOCATION_DISTANCE, locationListeners[0]);
        } catch (java.lang.SecurityException ex) {
            Log.i(TAG, "fail to request location update, ignore", ex);
        } catch (IllegalArgumentException ex) {
            Log.d(TAG, "network provider does not exist, " + ex.getMessage());
        }
    }
}
```

Figure 29 Location update code

4.3.4 ResolutionSettingsActivity.java

This is the activity that lets the user change the resolution the camera is using. It's a simple screen where it lists the supported resolutions and highlights the one currently being used. This was implemented to accommodate for less powerful phones. The lower the resolution, the less CPU power needed to process the video data.

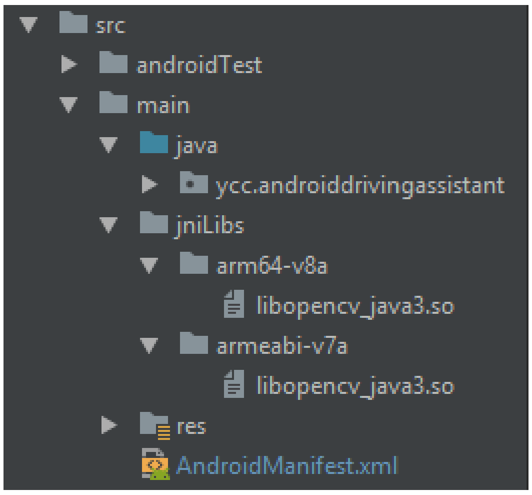
4.3.5 GpsSettingsActivity.java

This activity is used to enable or disable the LocationService class. Doing this will stop the app from getting the location of the device from the GPS, pausing the vehicles speed updates. I decided to include this feature because the constant query of the GPS for data is very demanding. The will help reduce the load put on the device.

4.3.6 JNI Libraries

The Java Native Interface (JNI) is a foreign function interface programming framework that enables Android applications compiled from managed code (written in the Java or Kotlin) to interact with native code (written in C/C++) (24). The OpenCV library is written in C++ so JNI libraries are needed for the app to work. Fortunately, the OpenCV SDK for Android includes these libraries that are compatible with multiple CPU architectures.

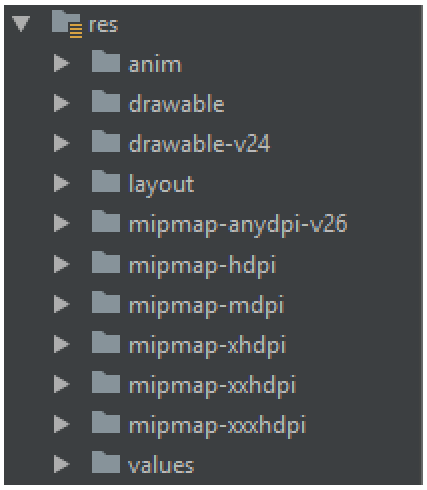
Most modern Android phones run on CPUs with ARMv7 or ARMv8 instruction sets therefore, I have embedded the corresponding libraries into the application. This allows the devices to use the OpenCV methods. If this is not done, the user would need to install an external application 'OpenCV Manager' (25) on the device which contains the libraries mentioned. Embedding them will make the storage size for the application larger but makes the third-party app redundant and keeps everything packaged together.



```
▼ src
    ▶ androidTest
    ▼ main
        ▼ java
            ▶ ycc.androiddrivingassistant
        ▼ jniLibs
            ▼ arm64-v8a
                  libopencv_java3.so
            ▼ armeabi-v7a
                  libopencv_java3.so
        ▶ res
          AndroidManifest.xml
```

Figure 30 JNI libraries in source code

4.3.7 Resources



```
▼ res
    ▶ anim
    ▶ drawable
    ▶ drawable-v24
    ▶ layout
    ▶ mipmap-anydpi-v26
    ▶ mipmap-hdpi
    ▶ mipmap-mdpi
    ▶ mipmap-xhdpi
    ▶ mipmap-xxhdpi
    ▶ mipmap-xxxhdpi
    ▶ values
```

Figure 31 Resources directory contents

Resources are the additional files and static content that the code uses, such as bitmaps, layout definitions for activities, animation instructions, and more. Storing files here externalize app resources from my code so that I can maintain them independently. These can be accessed using resource IDs that are generated in the project's R class, which is a dynamically generated class created during the build process to dynamically identify all assets.

Images of speed limit signs that are used to update the UI are stored here along with the animations for opening and closing the options, layouts for each screen present in the app, and the app icon that the user sees on the home screen.

4.3.8 External APIs

4.3.8.1 OpenCV

OpenCV is a major part of this project. It provides image processing capabilities for Android applications. This library is needed for sign and lane detection. It was imported into the application as a module making it available to use in Android as long as the necessary JNI libraries were present.

There were other alternatives like JavaCV, mentioned in the research section, but there are more advantages for OpenCV e.g. easier to set up, more documentation.

4.3.8.2 Google Mobile Services OCR

Google offers an OCR API to developers for mobile devices. It can be easily added to an application and if perfect for what I needed it for. The final line of the figure below shows how the API was added to the project.

```
dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
    implementation project(':openCVLibrary343')
    implementation 'com.android.support:design:27.1.1'

    implementation 'com.google.android.gms:play-services-vision:17.0.2'
}
```

Figure 32 Application dependencies

This API is used for understanding the signs picked up by the app. The area where the digits should be is cropped and a text recognition method is used to identify the contents.

5 System Validation

5.1 Testing

5.1.1 Black Box Testing

To test the application, black box testing was performed to test the expected behaviour and the actual behaviour. The test table was created to log all the tests completed and their results. The black box testing was chosen because automated testing wouldn't really be able to show and justify the test results as the results mostly were driven by audio feedback and visual on the device.

An observation made was that the application has a hard time carrying out the following functions in a real-world test where conditions involved poor weather and a lot of obstructions. The tests were carried out using a driving simulator instead so that the conditions can be controlled.

Test	Expected Result	Pass	Fail	Comments
Get user permission at first launch	A pop dialogue should ask for user permission to use the camera and location	🟩		When the app is launched, the permission dialogue is showed and asks the user for permission
Inform the user about why permission is needed	A toast should be displayed with information explaining why access to camera and location is needed		🟥	Was tested on Android version 9.0 and 7.1. Toast is displayed properly for v7.1 but is displayed after acceptance on v9.0
Detect circles and crop region of interest	The app should find circles and crop them for later use in the speed limit detection aspect. This should happen for every frame.	🟩		
Use the cropped image to get the speed limit if present	The app should use the image from the previous test to get the contents of it and display the necessary updates to the user	🟩		
Alert user through audio about a new speed limit	If a new speed limit is detected, the app should provide feedback through to inform the user	🟩		The app plays an audio clip saying what speed limit has been detected e.g. "50 kilometres per hour"
Detect only white and yellow lanes that is visible in the frame	The app should only display lane lines that are either white or yellow and show it on the screen in a highlighted colour	🟩		Detects what it should but if there is shine on the road, the results are less accurate.
Show centre of current lane	The centre of the vehicle's current lane should be calculated and showed continuously.	🟩		A blue dot is used to represent the centre of the lane which moves depending on the position of the vehicle

Alert user about lane departure through audio	If the vehicle drifts too far away from the current lane, the app should produce an audio warning repeatedly until the vehicle is back to a safe position	🟩		
Delay warnings	Don’t alert the user instantly if speeding or lane departure is detected to allow for errors in calculations and to give user time to correct themselves first	🟩		This was done by starting timers of ~3 seconds when faults in the user’s driving is detected. Alerts are given when timer runs out.
Change resolution of the camera	The resolution should be able to be changed by the user through the settings menu.	🟩		Can be changed successfully and will show what is currently being used.
Enable / disable GPS via options menu	Doing this should start /stop the location service from updating the app on the location of the device	🟩		
Change the app ability to get vehicle speed	When the GPS is disabled, the speed should not be updated and should inform the user. It should also be able to be restarted.	🟩		

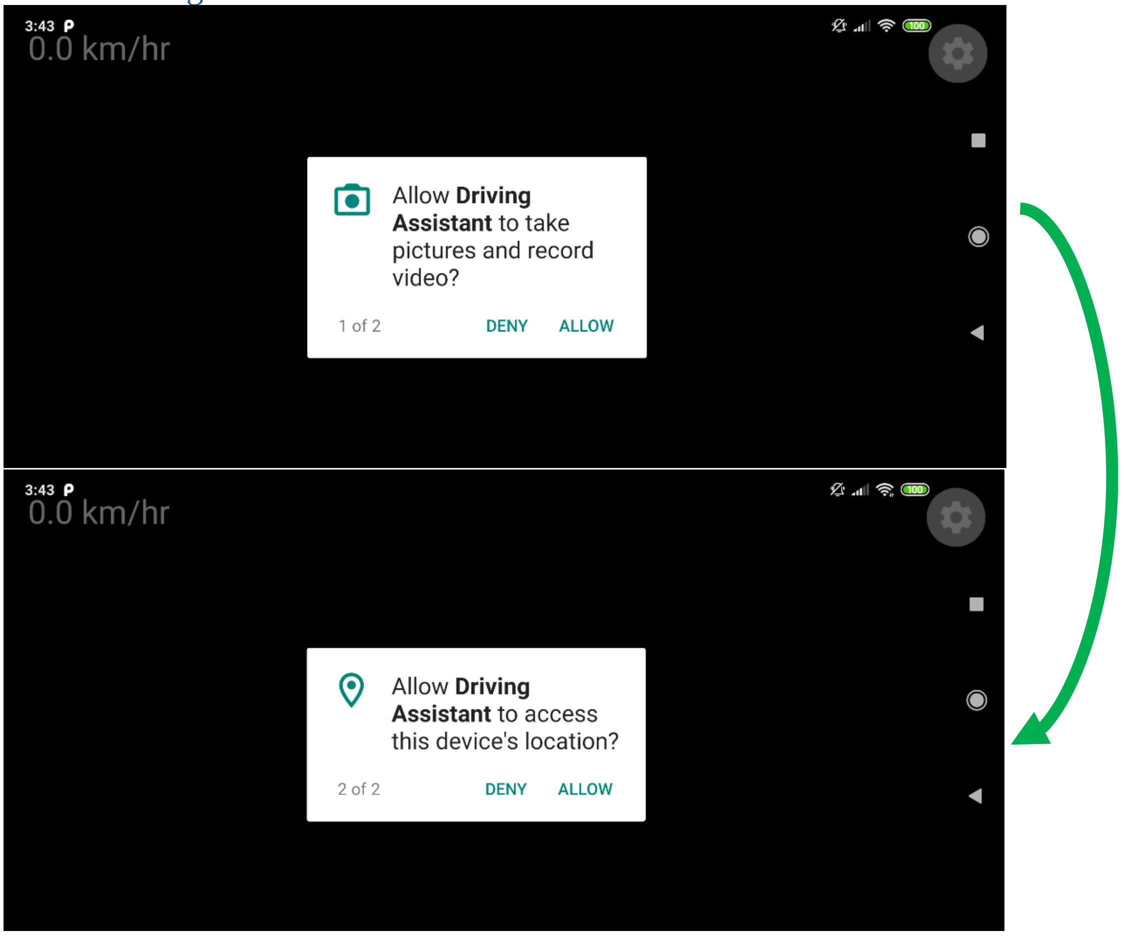
Table 2 Black box test results table

5.1.2 User Testing

To test the usability of the application, users with different levels of technical expertise was asked to try to use the application without any assistance. This was done to test the UI of the application and see if anyone found it difficult to navigate. Originally the settings menu for the app was to open a new screen and display the options there. This version of the settings used text instead of icons and the results of this test showed that users had taken more time to navigate than I would have liked. This prompted me to redesign the options menu using icons and reducing the number of screens to go through. The new UI design was tested and produced better results. Users were able to navigate and adjust setting quicker than before.

5.2 Demonstrations

5.2.1 Getting Permissions



Allow **Driving Assistant** to take pictures and record video?

1 of 2 DENY ALLOW

Allow **Driving Assistant** to access this device's location?

2 of 2 DENY ALLOW

Figure 33 Getting permissions prompts

When the app is first launched and permission has not yet been given, it prompts the user to let the app access these components. If denied, the app will not work and will

5.2.2 Changing Resolutions

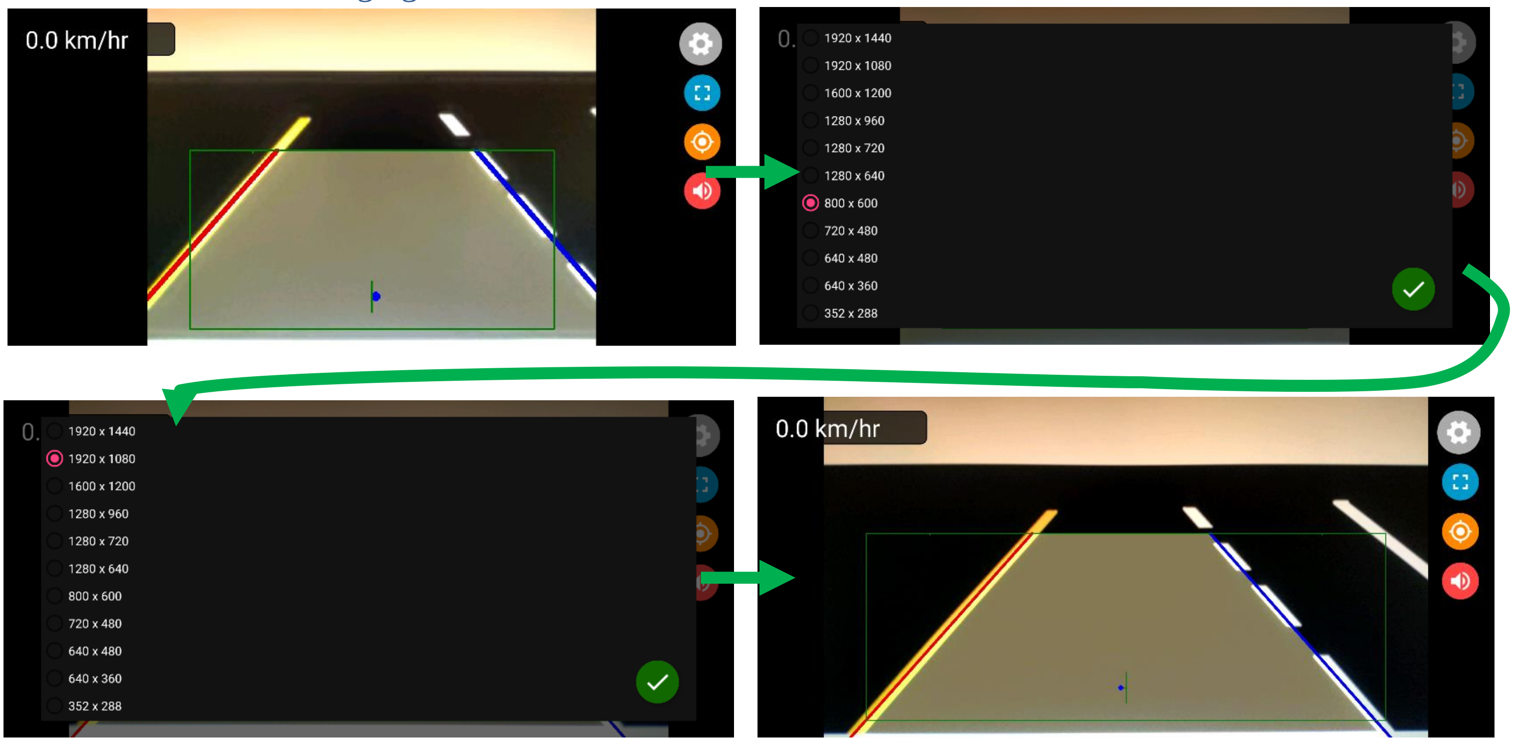
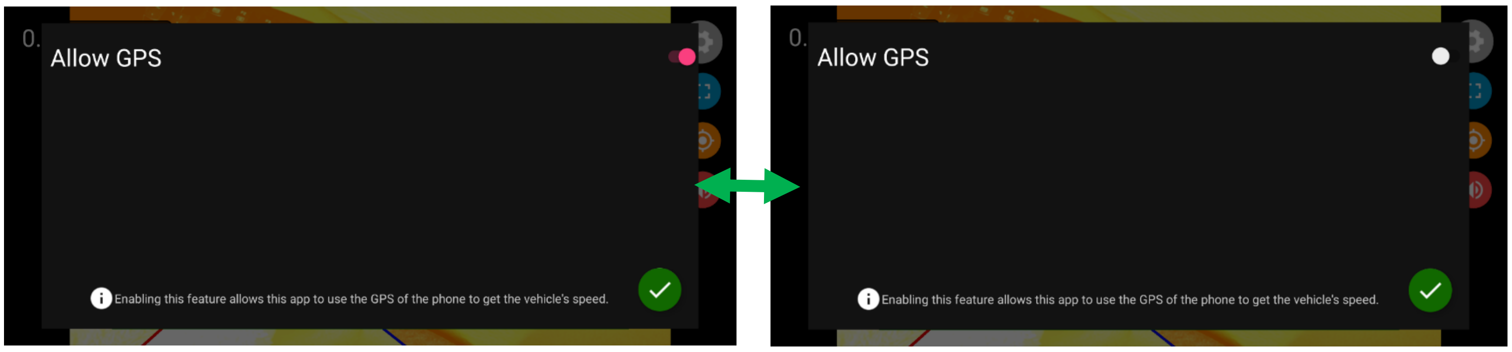


Figure 34 Changing resolution demo

Resolution is changed by firstly expanding the options menu (grey gear icon). Then a list of supported resolutions is shown, and the desired one can be chosen.

5.2.3 Enabling/Disabling GPS



Enabling this feature allows this app to use the GPS of the phone to get the vehicle's speed.

Enabling this feature allows this app to use the GPS of the phone to get the vehicle's speed.

Figure 35 GPS option demo

A simple switch is used to enable / disable the GPS.

5.2.4 Demonstration Videos

The demonstration videos of the application can be found at:
https://www.youtube.com/watch?v=QauqjXFVUd0
https://www.youtube.com/watch?v=ZBF2aWVOTCE
https://www.youtube.com/watch?v=fe4ENsGITdc&

The features demonstrated in the videos are:

- Speed limit detection
- New speed limit warning
- Lane detection
- Lane departure warning
- Speed of vehicle

6 Project Plan

6.1 Initial Project Plan

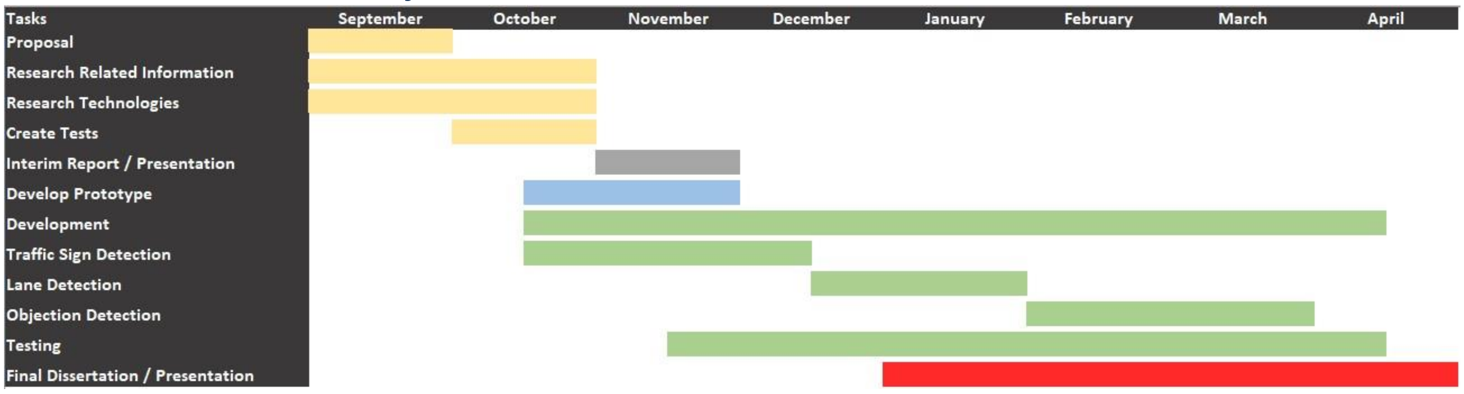


Figure 36 Initial Gannt chart

A simple Gannt chart was used to lay out the possible development plan and to create the guidelines for the project. At the beginning of the project, the plan only included the assumed main points of the development process however, the plan did change over the lifecycle of the project. The initial project plan assumptions were mainly imprecise, vaguely defined and failed to consider the implementation of the UI and settings. As the project progressed throughout the development process, new issues and development plans did emerge.

The main planned features initially were:

- Traffic sign detection
- Lane detection
- Object detection
- Audio alerts
- Training a custom AI
- Settings to accommodate slower devices

6.2 Actual Project Plan

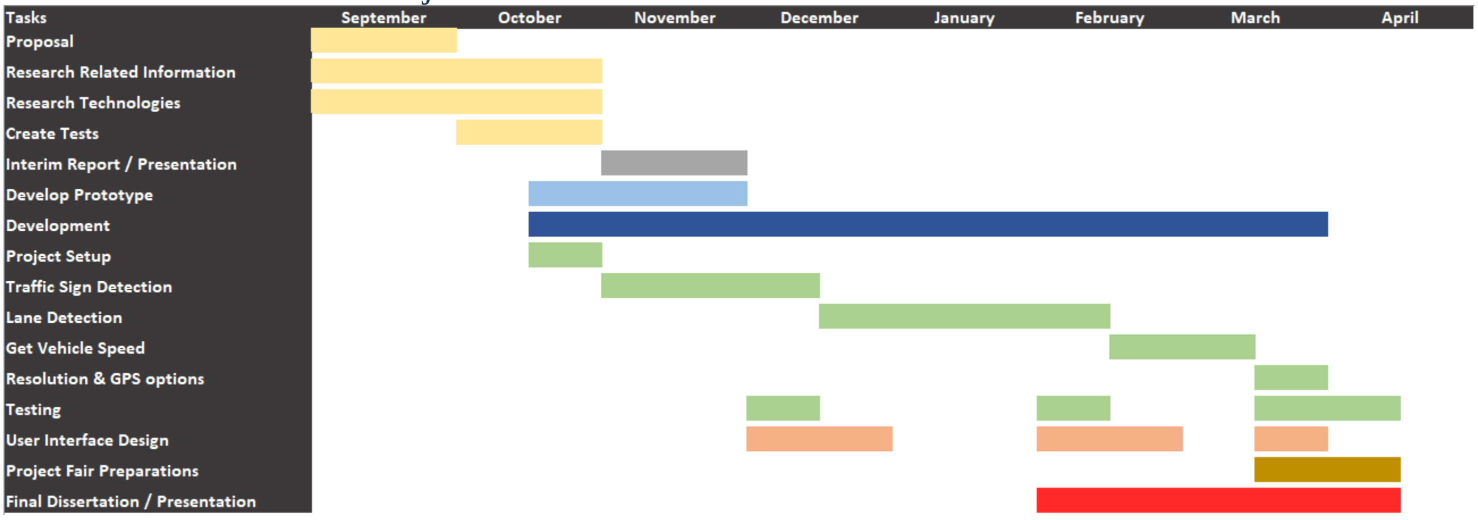


Figure 37 Actual Gannt chart

The actual project plan did include much more elements to be implemented and did dedicate a development time for UI and options. The plan, however, wasn't created in advance but instead was built and modified as the project progressed through its lifecycle and new requirements and ideas did arise. Difficulties when trying to set up the development environment caused delay to the project and left little time for training a custom AI for object detection. This hindered the progress of the project as I already spent about a week working on the feature and would take more to finish so I decided to discard this aspect to guarantee more time for the other main features.

Actual implemented features:

- Speed limit sign detection
- Lane detection
- Settings to accommodate slower devices
- Get speed of the vehicle with GPS
- Audio alerts

6.3 If the Project was to be Repeated

If the project was repeated, more time would be spent on analysis of the requirements to ensure that most critical parts of the project are identified and addressed early in the project’s lifecycle to avoid unexpected issues with critical elements, which may delay the project development. To save time during the development process and to ensure the reliability of the project, it would be advantageous to use development tools that the developer already has experience with. The knowledge of the tools and the capabilities of the tools would also help to determine the feature set of the project.

I would also manage time better than I did, poor time management allowed assessments from other modules overlap with the project’s weekly goals which had a bad effect on both aspects of my study.

6.4 Future Work

The project as it stands now has some functionality already available to it however there was some room left for future improvements from which the application will benefit even more. As of now, the application is basic in its functionality, but they worked well as shown by the testing.

One of the main things that the application can benefit from is the ability to recognise more road signs. Right now, it can only detect speed limit signs. Improving this aspect would make the application more useful in the real world.

Another feature I would add is a dashcam feature, where the app would keep recording over itself and when the user presses save, it would only keep the last few minutes of the video. I feel that this would be a very valuable feature since the app requires the device to be put in the dashcam’s position. This can be helpful in case an accident occurs. It would eliminate the need for a dashcam as it can be integrated into the app.

Currently, the application’s lane detection feature doesn’t work on lanes with large bends. Using a different method in the future like machine learning, it could predict the curvature of the lane and produce more accurate results about vehicle lane departure.

The app does not store any information about where speed limit signs are, it only detects it as it shows up in the camera frame. Future work could include making the app save the location of where a speed limit is detected and use it later if the user travels down that path again. This can reduce the amount of image processing the device needs to carry out.

7 Conclusion

The project developed is called Driving Assistant for Android and the purpose of the project was to provide older cars with some functionality of modern vehicles by only using an Android mobile device. These functionalities are; lane detection and whether the driver is drifting away from it, speed limit detection and if the driver is going over it. Any errors detected in the users driving will prompt the app to alert the user through audio.

To develop the project a variety of tools and different techniques and technologies were researched to decide which could be used for the project. Tools I have used before were paired with unfamiliar tools to develop the application.

The front-end side of the development used the Android framework. Layouts for screens were done through this and it is responsible for the parts a user interacts with. This was an easy to use framework that allowed me to design a simple UI.

The back-end side, where the data is processed, was done with the OpenCV and Google's OCR library. They were used to handle the data from the camera and returned the result to the front-end to inform the user.

The project was developed using an agile methodology with no set time given to any features. They were worked on until they were able to pass the tests outlined in Section 5 of this report. I would choose a different methodology if I were to do this project again. A methodology that designates a set amount of time to the different aspects of the project will be more effective in making sure deadlines are met.

While working on the project I learned about a variety of different technologies not being taught in the college. This project did equip me with skills which I utilise in the future as part of a company or in my own projects. During the lifetime of the project, I also learned how to think critically and prioritize tasks critical for a software project to be successful. If I had to do the same project again, I would perform more planning in the early stages of the project as this would make the development process easier in the later stages of a project.

The final state of the project performs well in the best road conditions i.e. lane lines are clear, no shine off the road, and minimal obstructions, however, the detection algorithms need optimisation for it to work in less desirable situations. Apart from this and the possible additions outlined in Section 6.4, I am satisfied with the overall outcome of the project and feel that it could have real-world use if improvements are made.

8 Bibliography

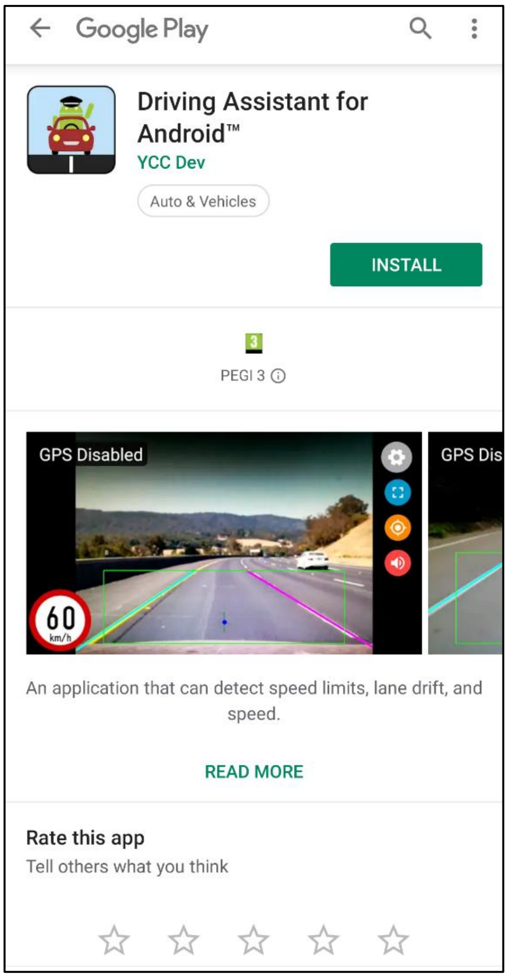
1. SAE J3016 automated-driving graphic [Internet]. [cited 2019 Apr 7]. Available from: https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic
2. Garret O. 10 Million Self-Driving Cars Will Hit The Road By 2020 [Internet]. Forbes. [cited 2018 Nov 27]. Available from: https://www.forbes.com/sites/oliviergarret/2017/03/03/10-million-self-driving-cars-will-hit-the-road-by-2020-heres-how-to-profit/
3. Technology [Internet]. Waymo. [cited 2018 Nov 21]. Available from: https://waymo.com/tech/
4. Surden H, Williams M-A. How Self-Driving Cars Work. SSRN Electron J [Internet]. 2016 [cited 2018 Nov 20]; Available from: https://www.ssrn.com/abstract=2784465
5. Jesmanczyk. Traffic Sign Detector [Internet]. 2018 [cited 2018 Nov 21]. Available from: https://play.google.com/store/apps/details?id=pl.jesmanczyk.android.driverassistant&hl=en
6. Childers J. Lane Detector [Internet]. 2018 [cited 2018 Nov 21]. (1.1). Available from: https://play.google.com/store/apps/details?id=me.streetvision.lanedetector&hl=en
7. Lane Identification Pro [Internet]. Vembar LLC; 2018 [cited 2018 Nov 21]. Available from: https://play.google.com/store/apps/details?id=com.vembarllc.laneidentificationpro&hl=en_US
8. Android - OpenCV library [Internet]. [cited 2018 Nov 27]. Available from: https://opencv.org/platforms/android/
9. Java interface to OpenCV, FFmpeg, and more. Contribute to bytedeco/javacv development by creating an account on GitHub [Internet]. Bytedeco; 2019 [cited 2019 Apr 6]. Available from: https://github.com/bytedeco/javacv
10. Mobile Operating System Market Share Worldwide [Internet]. StatCounter Global Stats. [cited 2018 Nov 27]. Available from: http://gs.statcounter.com/os-market-share/mobile/worldwide
11. Android Studio features [Internet]. Android Developers. [cited 2018 Nov 27]. Available from: https://developer.android.com/studio/features/
12. Run apps on the Android Emulator [Internet]. Android Developers. [cited 2018 Nov 27]. Available from: https://developer.android.com/studio/run/emulator
13. Mobile & Tablet Android Version Market Share Worldwide [Internet]. StatCounter Global Stats. [cited 2018 Nov 27]. Available from: http://gs.statcounter.com/os-version-market-share/android/mobile-tablet/worldwide
14. Nagy G, Nartker TA, Rice SV. Optical Character Recognition: An Illustrated Guide to the Frontier. In: Document Recognition and Retrieval. 1999.
15. Lazorenko A. TensorFlow performance test: CPU VS GPU [Internet]. Andriy Lazorenko. 2017 [cited 2018 Dec 1]. Available from: https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c
16. Using GPUs for Training Models in the Cloud | Cloud ML Engine for TensorFlow [Internet]. Google Cloud. [cited 2018 Nov 21]. Available from: https://cloud.google.com/ml-engine/docs/tensorflow/using-gpus
17. Pricing | Cloud Machine Learning Engine [Internet]. Google Cloud. [cited 2018 Nov 28]. Available from: https://cloud.google.com/ml-engine/docs/pricing

18. CUDA - OpenCV library [Internet]. [cited 2018 Nov 28]. Available from: https://opencv.org/platforms/cuda.html
19. Processes and threads overview [Internet]. Android Developers. [cited 2018 Dec 1]. Available from: https://developer.android.com/guide/components/processes-and-threads
20. What is waterfall model? [Internet]. SearchSoftwareQuality. [cited 2018 Nov 28]. Available from: https://searchsoftwarequality.techtarget.com/definition/waterfall-model
21. What is Agile Software Development? [Internet]. SearchSoftwareQuality. [cited 2018 Nov 28]. Available from: https://searchsoftwarequality.techtarget.com/definition/agile-software-development
22. Configure your build [Internet]. Android Developers. [cited 2018 Dec 4]. Available from: https://developer.android.com/studio/build/
23. Releases - OpenCV library [Internet]. [cited 2019 Apr 6]. Available from: https://opencv.org/releases.html
24. JNI tips | Android NDK [Internet]. Android Developers. [cited 2019 Apr 11]. Available from: https://developer.android.com/training/articles/perf-jni
25. Introduction — OpenCV 3.0.0-dev documentation [Internet]. [cited 2019 Apr 12]. Available from: https://docs.opencv.org/3.0-beta/platforms/android/service/doc/Intro.html#architecture-of-opencv-manager

9 Appendix

9.1 User Guide

The application can be downloaded and installed from
https://play.google.com/store/apps/details?id=ycc.androiddrivingassistant



Google Play

Driving Assistant for Android™

YCC Dev

Auto & Vehicles

INSTALL

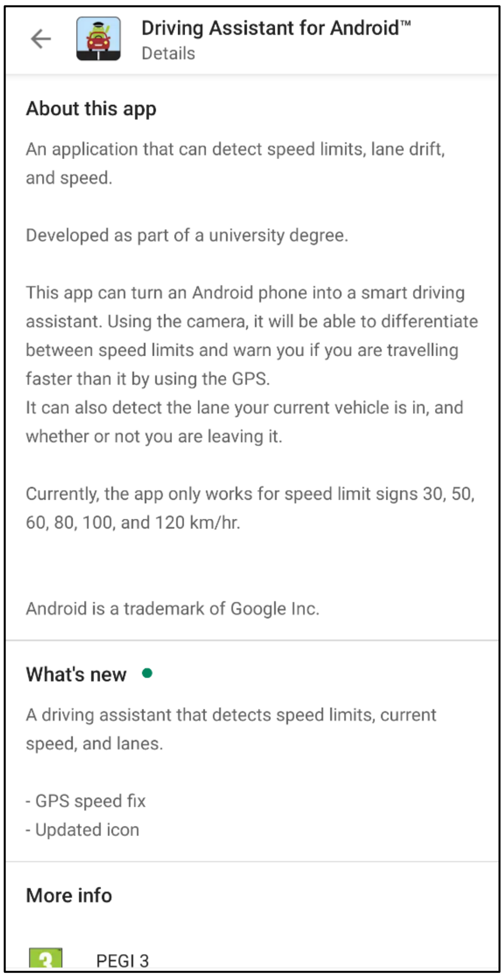
PEGI 3

An application that can detect speed limits, lane drift, and speed.

READ MORE

Rate this app

Tell others what you think



Driving Assistant for Android™

Details

About this app

An application that can detect speed limits, lane drift, and speed.

Developed as part of a university degree.

This app can turn an Android phone into a smart driving assistant. Using the camera, it will be able to differentiate between speed limits and warn you if you are travelling faster than it by using the GPS.
It can also detect the lane your current vehicle is in, and whether or not you are leaving it.

Currently, the app only works for speed limit signs 30, 50, 60, 80, 100, and 120 km/hr.

Android is a trademark of Google Inc.

What's new ●

A driving assistant that detects speed limits, current speed, and lanes.

- GPS speed fix
- Updated icon

More info

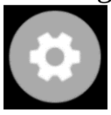
PEGI 3

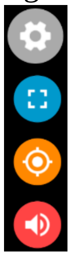
Figure 38 Google Play Store listing

For the app to work, camera permission must be given. The location services can be denied but the app will not be able to get the speed of the vehicle.

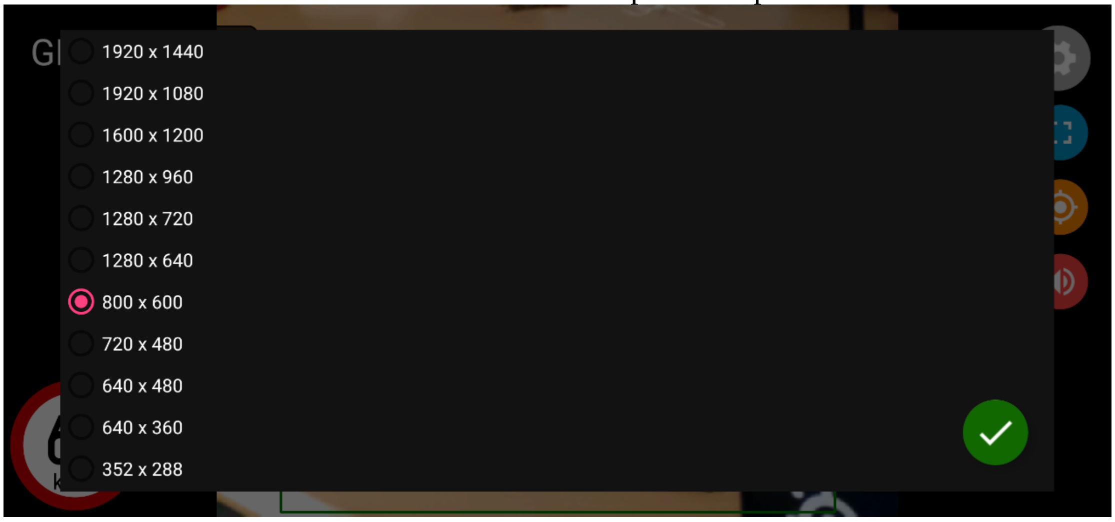
Mount the device, with the camera at the centre of the vehicle, so that it has a clear view out of the windshield. The green rectangular box should be lined up with where the lane will be. The size of the box can be adjusted by sliding up or down on the screen.

Pressing the following icon will expand the options menu.





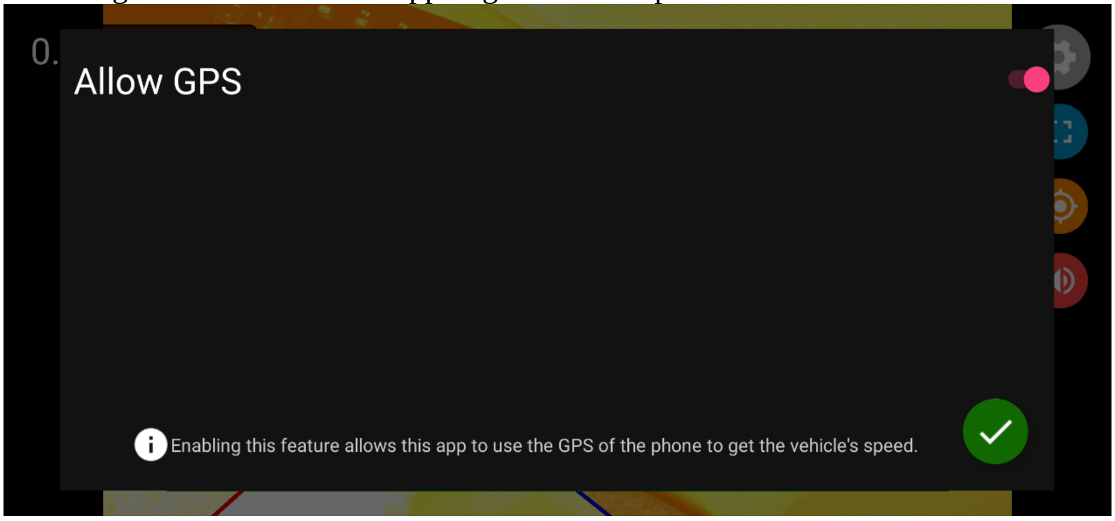
The blue button will open the resolution selection screen where the user can choose. Lower resolution runs smoother but less effective. Recommend resolutions are 720p for devices with fast CPUs and 640x480 for less powerful phones.



- 1920 x 1440
- 1920 x 1080
- 1600 x 1200
- 1280 x 960
- 1280 x 720
- 1280 x 640
- [✓ 800 x 600]
- 720 x 480
- 640 x 480
- 640 x 360
- 352 x 288

Figure 39 Resolution activity

The orange opens the GPS screen, it lets the user start/stop the location updates. Disabling it will not allow the app to get vehicles speed.



Allow GPS [✓]

Enabling this feature allows this app to use the GPS of the phone to get the vehicle's speed.

Figure 40 GPS activity

The red button enables/disables the audio alerts.

The detected speed limits will show up in the bottom left corner of the screen and will alert user every time a new one is found. The speed will be displayed in the top left corner, if enabled, in km/hr.

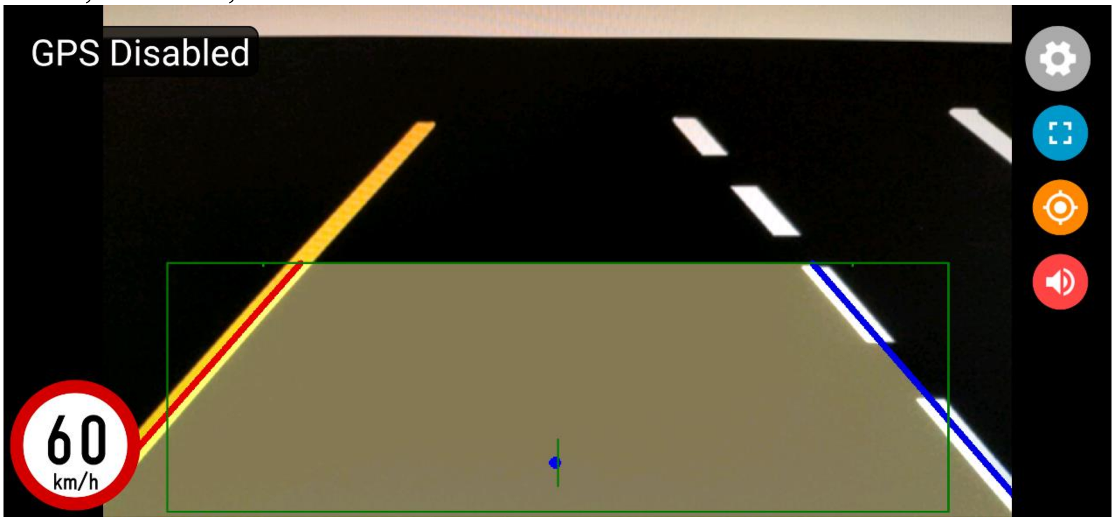


Figure 41 Speed and Sign location

Screen will flash orange is vehicle drifts from the lane, and audio warnings will be issued if not corrected within 3 seconds. The red line marks the left lane and the blue marks the right.

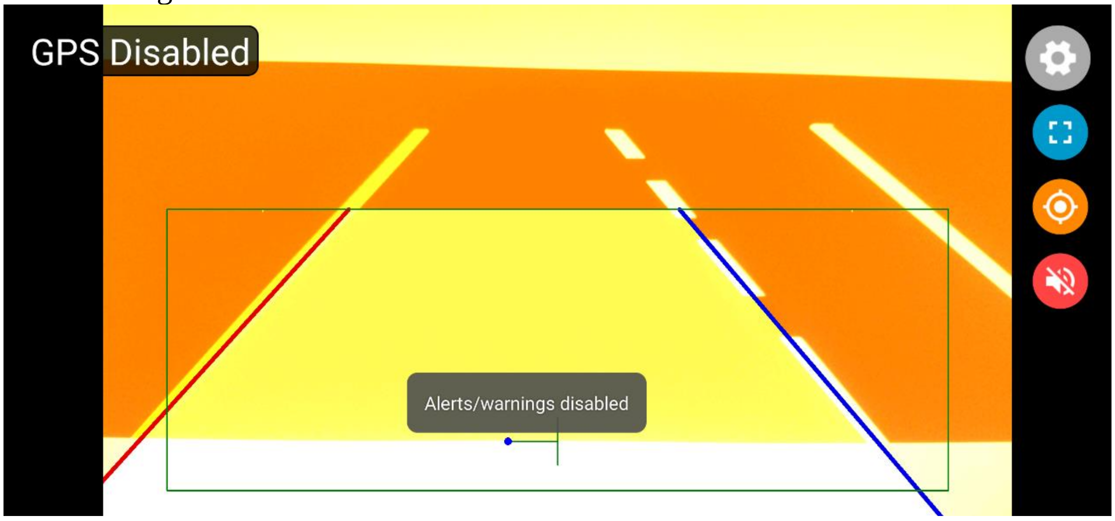


Figure 42 Lane detection info