Smart Kart

Final Report

Submitted for the BSc in

Computer Science

April 2022

by

George Jacob Anthony Kokinis

Word Count: ??

0.1 Abstract

Average speed check zones (ASC zones), typically enforced using SPECS[1] in the UK, are being increasingly deployed throughout the UK, doubling between 2013 and 2016 (BBC News, 2016). While useful for enforcing speed limits and increasing safety, with Owen et al. (2016) finding that fatal and serious collisions dropped by 36.4% at ASC zones installed purposely to reduce collisions, ASC Zones can lead to distracted driving, as the driver has to monitor their speed, which means looking away from the road to their speedometer for brief periods of time.

This project seeks to create a software application for a smartphone, that detects when a vehicle the phone is in enters an ASC zone, starts tracking the vehicle's speed, and gives the driver an audible alert if their average speed is at risk of breaking the speed limit, thereby reducing dependence on the driver to check their speedometer.

0.2 Acknowledgements

I would like to acknowledge the invaluable contributions of Eur Ing Brian Tompsett and Lydia Bryan-Smith (both, at the time of writing, of Hull University) with regards to LaTeX typesetting; without their guidance I would be stuck with regards to some of its peculiarities.

[1] SPECS (Jenoptik Traffic Solutions UK, n.d.)

Contents

Chapter 1

Introduction

1.1 Context

Average speed check zones are an alternative to traditional fixed point speed cameras. Fixed point cameras take two photos a given time-delta apart and measure the car's distance using road markings, to calculate the speed: $speed = distance \div time$. This is good for enforcing speed in that one position, but provides no enforcement for the road before or after the camera. In contrast, ASC zones effectively use the same methodology but across a longer distance (such as half a mile or 1.5 miles between cameras, and a series of cameras across tens of miles); hence effectively enforcing the speed limit across much larger areas of road.

Currently, there are various products for monitoring a driver's speed and for indicating speed cameras; the main smartphone applications in this space are Google Maps (Google LLC, n.d.b) and Waze (Waze Mobile, 2021); while Apple Maps (Apple Inc., 2021) will inform the user of fixed speed cameras[1], it does not inform of ASC zones.

However, Google Maps does not register ASCs as actual "zones", but instead as a fixed speed camera at the start of the zone. Waze displays progression through an ASC, but does not calculate average speed. TomTom GO Navigation (TomTom International BV., 2021) does track average speed in an ASC zone, but operates on a paid subscription model, so is not available to everyone. Hence, there is space in the market for a free solution to monitoring speed in ASC zones.

1.2 The Problem

While conscientious drivers should be aware of their speed anyway, it is likely that many check their speed more often and with more discretion whilst

[1]In the UK, these were originally "Gatso" cameras, later followed by Truvelo and Truvelo d-cam (Truvelo Ltd., 2020)

within an ASC zone. This means they may be focused on their speedometer when something important is passed, such as a direction sign, a signal from another road user, or an overhead gantry message - such as the "Red X" on Smart Motorways[2], or a speed limit change.

1.3 A Solution

By creating a smartphone application that warns a user if they are about to exceed the speed limit, the load on the driver can be reduced, allowing them to have more awareness of the road. By making this application free to use (whether free or supported by advertisements), this increases availability, potentially increasing the impact of this project.

1.4 Report Structure

The rest of this report is structured into the following chapters:

- Background: Describing and laying out technologies and concepts that may be used in the development of the application or are useful for understanding other technologies.
- Aims and Objectives: Specification in detail about what features the application should have and how this will be tested or measured.
- Design: A description of the initial design, visual and technical, for the application, and design(s) for surveys to gain feedback on the UI of the application.
- Technical Development: Discussion around the development of the application. Note that the actual development log will be listed in Appendix A.
- Evaluation: Discussion and Evaluation of how closely the application meets the Aims and Objectives in the aforementioned chapter.

After which there will be Appendices and the Bibliography.

[2]Smart Motorways (RAC, 2022)

Chapter 2

Background

The development of this project requires understanding and examination of various topics across various fields; including Kinematics, Programming, Law, and UX. Hence this chapter discusses and describes relevant topics.

2.1 Speed Cameras

2.1.1 Fixed-Point Cameras

In the United Kingdom, traditional fixed-point speed cameras were of a "Gatso" type, later replaced by "Truvelo", and later Truvelo D-Cam. They are installed either by the Local Authority, the local Police Force, or the relevant highways agency[1]; with all three often forming "Road Safety Partnerships" for given areas, that can then receive grants from the central government (King et al., 2011) to use for, among other things, the installation of speed cameras. The decision of where to install a speed camera is made on a few factors; with ageas (2019) claiming that at the proposed location, greater than 20% of drivers must exceed the speed limit, and that there must be a history of serious accidents.

At first glance, fixed-point cameras appear to work on a rather simple principle. As described in the Context, the simplest view of how to obtain the speed of a vehicle is $speed = \frac{distance}{time}$. Hence the camera can take two measurements a known time apart, work out the distance the vehicle travels between those measurements, and calculate the speed. Fixed-point cameras use K and Ku-band Radar signals to determine the speed of the vehicle; K-band meaning that the frequency used is between 18 and 27 GHz (IEEE, 2020); and Radar, an acronym for Radio Detection and Ranging (Rad, 1943), referring to the usage of a transmitter, receiver, and processing of K-Band or adjacent frequencies (in the Radio or Microwave ranges)

[1] *National Highways* in England, *Transport Scotland* in Scotland, *Traffic Wales* in Wales, and *DfI Roads* in Northern Ireland.

to determine properties of an object. The time of flight is the total time between transmitting a signal and receiving the reflection; and this can be used to determine the distance. The speed of light is known[2], and so the distance can be calculated using a rearrangement of the previous formula: $distance = speed \cdot time$. Hence, overall, the speed of a vehicle travelling towards/away from the camera can be calculated by:

$$c' = \frac{c}{1.003} \tag{2.1}$$

$$first\ measurement = c' \cdot time\ of\ flight \tag{2.2}$$

$$second\ measurement = c' \cdot time\ of\ flight \tag{2.3}$$

$$speed = \frac{|second\ measurement\ -\ first\ measurement|}{time\ between\ measurements} \tag{2.4}$$

However, this requires storage and memory of two time of flight measurements, and two distance measurements. Instead, an inherent property of waves can be used to determine the speed. The Doppler effect, first described by Doppler (1842) but better described for Radar applications by Wolff (n.d.), is the property of waves that, as an emitter moves away or towards a stationary receiver (or vice versa), there is an apparent change in the frequency. The frequency is higher if the motion is towards the stationary point, as the receiver will receive a greater number of waves per second; and lower frequency if motion is away from the stationary point as it receives less waves per second. Due to this, the "doppler shift" - the difference between real and effective frequency, Δf - can be calculated as follows:

$$\Delta v = -(v_{receiver} - v_{source}) \tag{2.5}$$

$$\Delta f = \frac{\Delta v}{c'} \cdot f_{emitted} \tag{2.6}$$

In the case of the speed camera, we can control $f_{emitted}$, and we must take into account the fact that doppler shift will occur twice. Hence, the velocity v of a car can be calculated as such:

$$v = \frac{\Delta f}{f_{emitted}} \cdot \frac{c'}{2} \tag{2.7}$$

If the vehicle's velocity calculated by equation 2.7 exceeds the road's speed limit, the camera takes two photos in quick succession. In the original Gatso cameras these photos were on photographic film, which would be collected and processed by the local police force on a regular basis. Retrofitted

[2]299 792 458 metres per second in a vacuum, known as c, and slower in air: which can be calculated with $speed = c \div n$ (Hecht, 2002); where n is the refractive index of the medium. de Podesta (2002) gives 1.0003 as the approximate refractive index of light in air; approximate as it can be affected by factors such as moisture content.

Gatso and the newer Truvelo & DCam systems upload their photographs and Radar Measurements to a system with the local Police Force; these systems are believed to be connected with the Police National Computer (PNC) (ACRO, n.d.). These photos contain the registration plate of the vehicle[3], and markings on the road. These markings are graduated and are used to determine the distance the vehicle travels between the photos, and hence a speed can be determined; this *secondary measurement* is used by prosecution teams to confirm an offence was made, but is not absolutely needed for a prosecution.

2.1.2 Average Speed Cameras

It may be wise to think, since average speed cameras were introduced after fixed-point cameras, that they would be significantly more advanced. In some ways they are, but in other ways they are much simpler.

NPCC (n.d.) tells us that ANPR, Automatic Number Plate Recognition, refers to the reading of vehicle number-plates using OCR[4], storing information about those number-plates (such as the location and time of the scan), and querying a central system (such as the PNC) for details about the vehicle and its registered keeper. Number plates in the UK and generally in the EU are retroreflective; meaning that light is reflected back in direction it came from. Hence, most ANPR systems, including those used by Average Speed Cameras, take photos and use a flash in the infrared range to take a clearer image of a number plate.

Average Speed Cameras are deployed as systems consisting of multiple ANPR-equipped cameras, at known intervals, along a road or roads. These cameras can be permanently installed along roads, or temporarily installed at the site of roadworks. Installations come in various forms; they are installed on their own poles, on existing street furniture (EG lampposts), and on overhead gantries. The system has knowledge of the road distance (as opposed to direct distance, colloquially "as the crow flies") between cameras, and uses this knowledge combined with timestamped photographs of vehicles passing the cameras (with ANPR being used to match number-plates to timestamps) to calculate an average speed. This average takes into account factors such as changing lanes, and using junctions in an attempt to defeat the system.

A given system must have at least two cameras (so there are two measurements to calculate speed from), but there is no legal or algorithmic upper bound on the number of cameras in, or the length of, a single system (Car-

[3]Gatso cameras must take a photo of the rear of the vehicle due to the use of a bright flash, so will always obtain the registration plate; but do not photograph the driver and so the identity of the offender can be disputed. Truvelo & DCam may take photos of the front, thereby identifying the offender, but some vehicles lack front plates.

[4]Optical Character Recognition (Eikvil, 1993).

buyer, 2021). Any upper bound is likely to be budgetary, technological, or jurisdictional; that is, the authority responsible for the system cannot afford the number of cameras, the computers to store and process data from the cameras, or legally cannot deploy cameras past a county or city boundary.

Put simply, average speed cameras go back to using the simple equation of $speed = \frac{distance}{time}$. By applying this over a larger distance of road, rather than a small section, it encourages drivers to stay at or below the speed limit, and to drive in a smoother manner.

2.1.3 Summary

With regards to the application this project produces, from the previous knowledge, the following factors should be noted:

- Fixed-point cameras are not relevant to average speed cameras in terms of applied technology, and therefore aren't relevant to the application. At most, there may be scope to provide notice of fixed cameras within the app.
- There is no public knowledge of pairing between any two cameras in a system; they may be paired sequentially or randomly.
- Hence, if a dataset has specific locations of cameras within a system, it is reasonable that average speed should be calculated between sequential cameras - IE between the current position and the previous camera, then reset at the next camera.
- If the dataset does not have specific locations, and only the start and end, then the average speed should be calculated between the start and end.

2.2 Predicting Speed

The change in speed over time, acceleration, is

$$a = \frac{v - u}{t} \tag{2.8}$$

This allows for finding change of speed in the past; but the application must effectively forecast whether a user's vehicle is going to exceed the speed limit. Rakha et al. (2001) provide and validate a model to predict maximum acceleration for trucks (LGVs). However it does not suggest whether their model works for smaller vehicles, and the model requires input parameters of tractive effort calculated from transmission efficiency, current speed, engine power, etcetera; it's a very comprehensive model, but an average driver may not want to go and find these details for their vehicle just to track

average speed. Hence this model is unsuitable for the application; however the methodology may be useful to analyse.

Instead of a complex model, the application can just use past acceleration to predict future speed. This is naive model, but by using appropriate development techniques such as separation of concerns, this model can be changed for a more accurate one in the future. The basis, then, is a simple rearrangement of 2.8:

$$v = u + a \cdot t \tag{2.9}$$

The main tuning parameter of this model for forecasting is t, time; whether this is user configurable (a setting saying "Notify me t seconds before exceeding the limit") or "hardcoded" is a consideration to be made. As well, this needs to be used with some hysteresis; it will be exceedingly annoying to the user if the application constantly alternates the alerting state.

2.3 Programming and Development

There are various options for platforms on which this application could be developed. This section discusses options, and ultimately decides as to which platform should be used.

2.3.1 Cross-platform frameworks

One contemporary paradigm is the use of cross-platform frameworks; where a developer writes their code using provided libraries, and the framework handles the "mapping" between calls into the libraries and the various host features. These frameworks can range from intrinsic to the language - in the case of Java and the openJDK (Oracle Corporation, 2022), where a JVM is almost completely necessary for running the program and so is provided by the Java developers - to external frameworks developed by entities separate to those who develop the language.

Flutter Flutter (Google LLC, 2021b) is a cross-platform framework provided by Google (but is open source, so can be modified by the general public) that allows developers to build "multi-platform applications from a single codebase". It uses and itself is written in Dart, Google's "client optimized language for fast apps on any platform" (Dart project authors, 2022). Bar the singular codebase, it provides explicit widgets for specific platform actions, such as permissions dialogs.

Being provided by Google and using a language from Google means there is a high level of integration between langauge features and the framework, whilst still maintaining separate development groups.

UNO Platform UNO Platform is a framework for "Pixel-Perfect Multi-Platform Applications with C# and WinUI" (nventive Inc., 2021). It allows a developer to write once and compile applications for most major platforms, whilst also allowing for platform-specific code.

The use of C# and WinUI is advantageous, as the University of Hull teaches C# throughout undergraduate Computer Science courses, so prior knowledge can be reused; and WinUI knowledge can be reused in future applications.

MAUI .NET MAUI is a cross-platform UI framework that allows .NET developers to write UIs that work on multiple .NET targets (.NET Foundation Contributors, 2021). All .NET targets on .NET 6 share the same "base class library", meaning the underlying hardware and platform is abstracted away, but UI definition is still platform specific. MAUI allows a developer to use XAML to define UIs that work across Android, iOS, macOS, and Windows.

Again, the use of .NET (and therefore most likely C#) is advantageous for building on prior knowledge. However, being a framework still in "preview", whilst meaning there's still improvements to be made and inherent longevity, means that MAUI may be more prone to bugs and errors than a more mature framework.

Summary Whilst these frameworks all ease cross-platform development, and help developers use a single codebase instead of multiple, for this project expediency and ease of development means that cross-platform is less important than a working application in the first place. For example, all of these frameworks enable the application to run on a desktop or laptop computer; but the chances of a user bringing even a laptop into their vehicle is very low, and the objective of this project is a smartphone application. So the potential added complexity of using a multi-platform framework to enable use on a platform that isn't relevant is not conducive to the project.

Hence, this project will not use a multi-platform framework, and will instead use platform-native development on a mobile platform.

2.3.2 Platform Native

Statcounter Limited (2021) gives Android and iOS an approximate combined market share of 99.27% as of March 2022. However, even in the remaining 0.73% of the market, there are various options; Tizen[5], KaiOS, and SailfishOS to name a few, as well as mobile versions of desktop systems, such as pureOS.

[5]Backed by the Linux Foundation, but primarily developed by Samsung.

While all could be good platforms for this project, Android and iOS' combined dominance and availability make them the two platforms to decide between, if this application is to provide the largest benefit to the public. Hence it is prudent to contrast and compare these platforms, and discount the others.

Android Android is a platform originally developed by the Android Open Source Project, but later adopted and marketed by Google. The platform is based around the JVM, originally Dalvik (which JIT'd[6] all applications) but later ART (in which applications are compiled to native code during installation), and hence originally development was done in Java.

However, since 2017 Kotlin has had "first-class support" from the Android Team (Shafirov, 2017), and since 2019 has been the preferred language for android development (Lardinois, 2019). Kotlin is a language from JetBrains™ with modern functional and object-oriented features; originally on the JVM, but since Kotlin 1.3, a beta feature to compile directly to native executables has been available. When originally announced, Krill (2011) described Kotlin as has having "features so desperately wanted by the developers".

iOS iOS is the platform used by Apple on their iPhones and the majority of their iPads (until the introduction of iPadOS). Apple provides an iOS development kit, iOS SDK, and an IDE, XCode, as well as documentation and other tooling.

Historically iOS applications were written in Objective-C, an object-oriented language originally developed in the 1980s. However since reaching version 1.0 in 2014, Swift (Apple Inc., 2014) has transitioned into being the main language for iOS development. Swift is also an object-oriented language.

Comparison Both Android and iOS have mature and full-featured SDKs, tools, and ecosystems. Both have historical and current languages, with iOS' languages integrating more tightly with the SDK than Android's.

However, the advantage Android holds over iOS is one of cost. To develop for iOS native, a developer is required to have a MacOSX device to run XCode and the rest of the toolkit; even when using cross-platform frameworks, this requirement remains. It is also advisable to have a physical iOS device to test on. At the time of writing the cheapest MacOSX system from Apple's UK store is £699 for a base-model Mac mini; and that price doesn't include a screen or any peripherals. This is a significant outlay for a smaller developer, not even counting the cost of an iOS device. Renting time

[6] JIT: Just-in-Time, a methodology in which code is compiled to an intermediate form, then during runtime interpreted into machine instructions; See Aycock (2003).

on a Mac is possible, with Moboware, Inc (2022)'s "Macincloud" solution starting at $4 an hour, and Scaleway SAS (2022) offering €0.10 per hour pre-tax. However, this requires the use of remote development, which can be complicated to set up; so the discount in money is offset by the spend in time.

In comparison, Android Studio[7] is OS agnostic; it will run on Windows, Mac, Linux, and even some ChromeBooks, as long as they have a 64-bit processor, enough RAM (8GB on most platforms), and enough storage. It is also possible to develop an Android app without Android Studio; the build system, Gradle (Gradle Inc., 2022), is usable from the command line, so any text editor could be used for development.

Furthermore, publishing an application is another advantage for Android. To publish an iOS app, a developer must enrol in the Apple Developer Program and pay an annual fee of $99 for individuals or $299 for companies (Apple Inc., n.d.); this is regardless of whether your application is free or not. Other than this, there is no supported way of publishing applications.

On Android, a developer can publish their application to the Google Play Store (Google LLC, 2021c), to alternative marketplaces, or just offer application files (specifically APKs) for download.

- To publish on Google Play, a developer must pay a one-off fee of $25; bar this, any costs are as proportions of advert revenue and of transactions users make when buying an application or making in-app purchases; so a completely free app is free to publish.

- Alternative markets include F-Droid (F-Droid Limited and Contributors, 2022), which is completely free to publish to, but doesn't allow for paid applications; it does allow for developers to have donation links, and for some in-app purchases.

- Finally, although disabled by default, Android allows for installing applications from downloaded files, also known as sideloading (Butler, 2022). A developer could develop and compile their application, then distribute the resulting apk file through any method they like - whether a download link on a website, or with some form of repository.

2.3.3 Summary

Hence, this project will develop an application for Android native; as it provides the most broad and easily accessible platform to develop and publish on. Furthermore, this project will use Kotlin as the language for development, as this is the contemporary choice; with Lardinois (2019) quoting Google as saying "If you're starting a new project, you should write it in Kotlin".

[7]The primary IDE for Android development, (Google LLC, 2022)

Chapter 3

Aims and Objectives

The overall aim, as stated in the Abstract, is to create an application to reduce load on drivers in average speed check zones. However, this is rather opaque as a goal, and it is hence necessary to break this aim into smaller, manageable aims and objectives.

3.1 Primary Objectives

The overall primary objectives are:

1. To develop an Android Application that can track a user's average speed.
2. This application will be able to use the collected data to predict if a user's average speed is going to go over a set limit, and will audibly and visually warn the user.
3. Tracking can be started and stopped manually by the user.
4. The application can detect when the user enters an Average Speed Check zone, based on location.
5. Tracking can be started and stopped automatically, when a user enters or exits an ASC zone.
6. All manual interaction can be supplanted with voice commands[1].

These primary objectives are intended to implement key functionality, with any other functionality being a "nicety" to implement later.

[1]For example, through integration with Google Assistant (Google LLC, n.d.a).

3.2 Secondary Objectives

If possible, the application could also:

7. Allow the user to change the audio used for the audible warning.

8. Allow the user contribute to the dataset used for automatic detection of ASC zones.

Changing the audio allows the user a choice of how they wish to be alerted; whether they want a subtle alert or a very distinct, jarring alert. Allowing the user to contribute to the dataset is a case for the public good, but depending on the dataset may require substantial development work; hence it is a secondary objective.

3.3 Tertiary objectives

In some territories, the use of physical devices to detect speed cameras is illegal, and applications doing the same are either explicitly or potentially illegal. One such case is France; any device or system used to "detect the presence" of devices for regulation of road traffic is illegal under the Code de la route (Légifrance, 2012). Hence, an objective for the application would be:

9. To detect if the device is in a territory with such a regulation, and

10. Automatically prevent its own usage in those territories, providing a message to the user about why.

Furthermore, different classes of vehicles have different speed limits. In the UK, the National Speed limit for LGVs over 7.5 metric tonnes is 60 MPH, regardless of whether the road is single or dual carriageway (GOV.UK, n.d.). Hence:

11. The application allows the user to set what class of vehicle they're driving.

12. The application uses a speedlimit adjusted to account for this.

3.4 Summary

In terms of evaluation of the project, if the primary objectives are met then the project can be judged to have succeeded; the secondary and tertiary objectives are an opportunity to extend functionality.

Chapter 4

Design

Once an initial design is proposed, any changes to that design should be noted and tested to ensure there are no regressions. In this section, initial designs for the UI, the user flows, and application structures are proposed.

4.1 User Flows

The basic flow of application is laid out in figure 4.1. The rest of the flows are effectively automation around this basic flow.

The main automation is effectively a separate loop that invokes the original flow, by checking for location of the user and comparing to a dataset of average speed zones; and similarly the original flow must now return to the invocation loop when it determines it has exited a zone. This is displayed in figure 4.2.

Figure 4.1: Basic flow for the application.

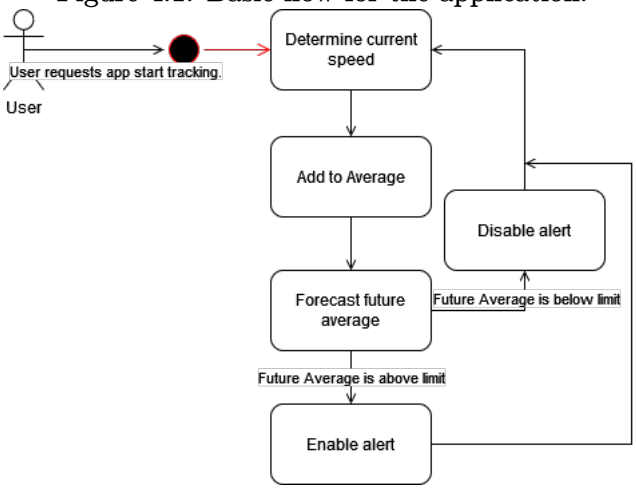
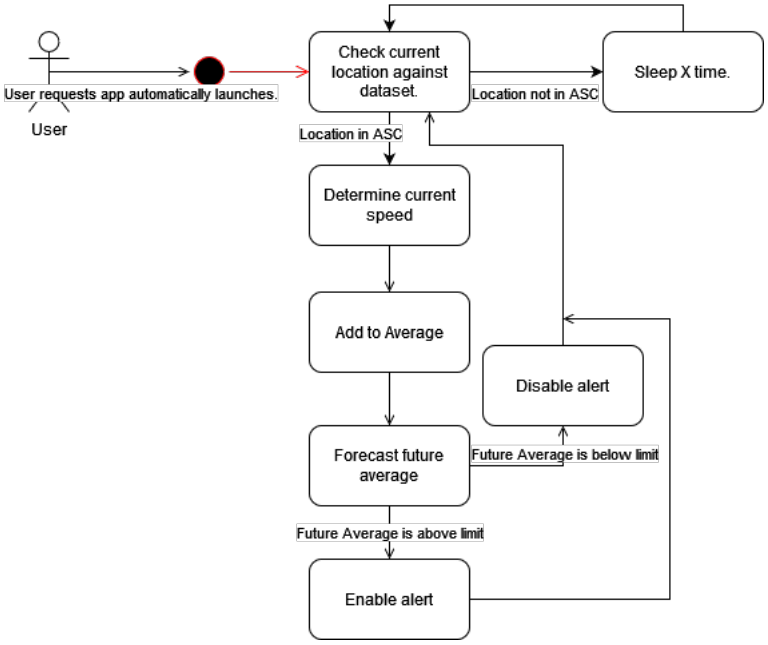


Figure 4.2: Flow with automation.



Chapter 5

Technical Development

While the actual development of code can be found in appendix A, it is worthwhile discussing, separately to that, justification and reasoning behind certain decisions and implementations, as well as more general discussion about development on Android.

5.1 Android Jetpack

Android applications previously used the Android support library to allow for some form backward compatibility; Instead of the developer explicitly handling older versions, the support library allows developers to either have an implementation of "newer features on earlier versions", or "gracefully fall back to equivalent functionality" (Android Open Source Project, 2021b). However since Android 9.0, the support library has been replaced by Android Jetpack libraries. Jetpack not only provides the backwards compatibility of the support libraries, but also a large amount of classes and helper functions that "help developers follow best practices, reduce boilerplate code ... so that developers can focus on the code they care about" (Google LLC, 2021a). Case in point with this project; the application makes use of *AppCompatActivity*, which means the main activity will have a consistent appearance across previous versions of android; and the Jetpack preferences library is used as discussed below.

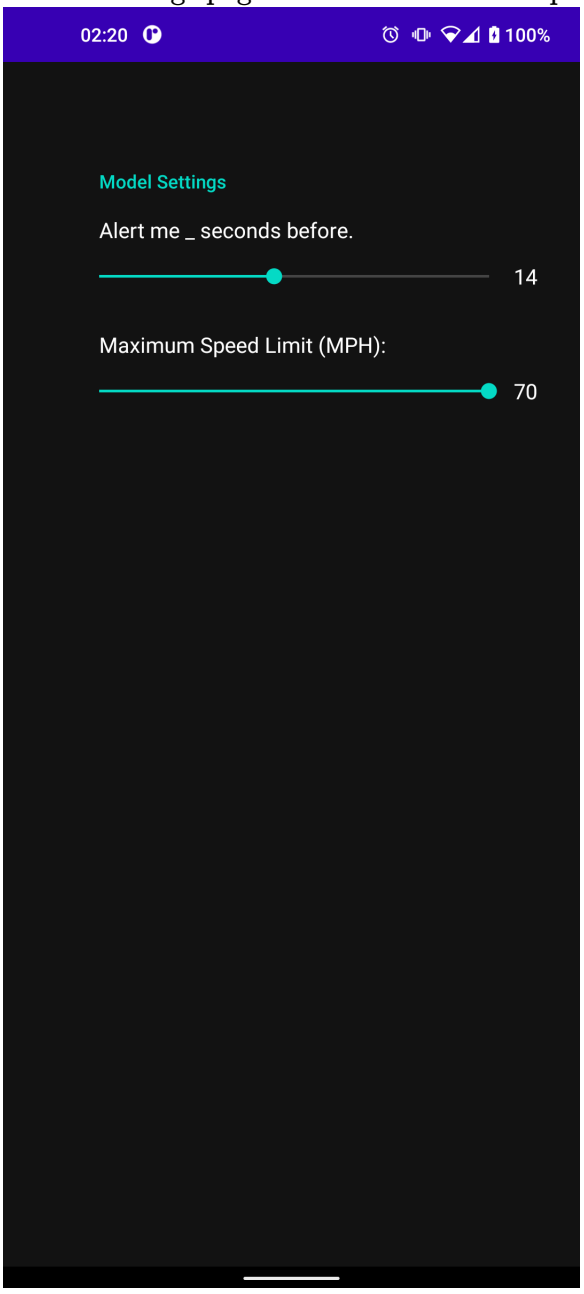
5.2 Settings Page

Certain parameters of the application are user configurable; for example, except for in Automatic mode, the speed limit can be changed by the user, and the forecast time for the model can be configured by the user. However, the speed limit should be changeable from the main interface of the application, while the forecast time does not need to be. Hence a "settings page" should be created.

Android Open Source Project (2021a) provides a guide on how to implement a settings screen, using a *PreferenceScreen* fragment. By using this part of the Android Jetpack library, a developer doesn't have to implement their own Settings system. However, to get to this settings page, the user must interact with something; one idea is for a settings button on the top bar of the application, which on Android is referred to as the "ActionBar". Android Open Source Project (2019) provides information on how to configure an ActionBar that works across past and future android versions, and allows for action buttons. However, despite implementing this in the project, a settings button was never seen on the action bar throughout development. A future task for this project would be revisit and attempt to implement this; it is more likely to be a bug with the project's code than with the underlying Android libraries.

For testing purposes, a button to reach the settings page was explicitly added. It was removed during design of the rest of the UI, then re-added at the end. The settings page itself can be found in figure 5.1; currently containing just two settings, but could easily be extended for further features.

Figure 5.1: Settings page at end of main development.



5.3 Android Resources

Android resources are "additional files and static content that your code uses" (Android Open Source Project, 2022a), including images, sounds, layouts, and (static) values. In this section the use and features of some resource forms is discussed.

Resources can be "qualified" with a number of factors, such as device orientation, type, "night mode", language, etc; allowing for automatic reconfiguration when the device changes.

5.3.1 Drawables

Images in Android are considered to be "Drawables", and they can be either raw images in Bitmap or Vector form, or more Complex forms. For example, a file can define a StateList composed of other drawables, and code can change between them, usually based on the state of the container.

This project's only explicit use of a drawable is as a background SVG[1] used to display the speed limit, imitating the background of a UK Speed Limit sign. However, this could become part of a StateList, with other countries backgrounds, or signs with specific meaning (E.G. the UK's National Speed Limit sign), to better improve a user's recognition of the speed limit.

The use of an SVG instead of a bitmap allows for better scaling for different UI sizes; bitmap (also known as raster) graphics are prone to artifacts on upscaling (as information has to be interpolated).

5.3.2 Layouts and View Binding.

When defining an activity's or fragment's initial state, a developer should provide a layout file that defines the initial state in XML. It is theoretically possible to build a layout in code, and it is sometimes necessary to make changes in code; when *inflating* its layout, an Activity needs to add a fragment to take the place of a *FragmentContainerView*. However the actual layout should be in the layout resource file, and the code should manipulate the data displayed within the layout.

Manipulating views within a layout has historically been a case of calling *findViewById()* to get the view to manipulate. However this is not type safe and has few compile-time guarantees. Since Android Studio 3.6 (and the version of Jetpack released alongside it), this can be replaced by View Binding (Android Open Source Project, 2022f); upon creation of an activity or fragment, the view inflates its layout and stores the root element as a member field. Then, instead of calling findViewById(), a developer can access binding.viewId.property, which provides nullability and type-safety.

[1] Scalable Vector Graphics (W3C SVG Working Group, 2010)

This project has a few layouts in use; two activities and two fragments, one each for the main page and the settings page. It also uses view binding in the main activity and the "DashboardFragment"; it is not necessary to do so in the settings page and fragment, as access to the components is handled preferences part of the Jetpack Library.

5.3.3 Strings

Android allows for easy localisation through the use of string resources. When the developer uses some text, the code refers to the string resource instead; based on the device's locale, this is substituted for a translated string if available.

This extraction of string resources also allows for a "single source of truth"; rather than having to find all instances of a string in code and replace them, a developer wishing to change text just has one location to alter.

Every string in the project is a string resource, but it provides no translations for them.

5.3.4 Integers

Similar to the single source of truth for strings, Android resources can provide a single source of truth for Integer values. In the project this is used to set default values and limit values for the settings page.

5.4 LocationManager: Getting location on Android.

The basis of this project involves location data. Attaining objectives four, five, nine, and ten, require the use of location data; and getting speed for objectives one and two, while possible to do through other methods, is best done through the location system as explained later in this section. The first part of this is ensuring the application has the appropriate permissions; in the case of location, the application is required to request *ACCESS_FINE_LOCATION* and *ACCESS_COARSE_LOCATION* (Android Open Source Project, 2022e). This is done in the application's manifest file, AndroidManifest.xml.

Getting location on Android involves the use of the LocationManager class (Android Open Source Project, 2022c). LocationManager does not provide a public constructor, so code attempting to use LocationManager must get an instance in some other way. In the project, this is done through a call to *context.getSystemService()* and passing in the constant *Context.LOCATION_SERVICE*; this then returns a Java *Object*, so the calling code must cast the returned Object into a LocationManager.

This LocationManager instance then has two paths for a developer to get location data from it. The seemingly obvious way is to call *getCurrentLocation*, which "asynchronously returns a single current location fix". However, this quickly becomes complex: this method must be called with an executor as one of the parameters, to provide which thread the callback should occur on. This in itself is not terrible to work with, but the complexity comes with calling this in a loop to constantly update the average speed; this loop should be executed on a background thread as well, so now there's a background thread managing repeated calls to another background thread, and managing the transfer of data back to the main UI thread can be difficult.

The alternative is through the use of *requestLocationUpdates*. This method allows code to specify minimum distance and time between updates, and register a Listener to be executed when an update occurs. This is much easier, as the update loop and state of whether to get updates is handled by the system's LocationService and its own thread(s). When the user requests to stop tracking, this is as simple as calling *removeUpdates*; this de-registers the Listener passed as a parameter (if it was already registered), and no further updates are provided.

5.4.1 In the Project

In the Project, this is implemented as the ViewModel itself being a LocationListener, and having two functions, *startTracking* and *stopTracking*, that call the relevant LocationManager functions. The *onLocationChanged* callback function checks if the Location has a speed, and if so adds it to a mutable list, then calculates the average speed from the list.

5.5 MediaPlayer: Playing sounds on Android.

As part of objective two, the application should audibly warn the user. On Android, the default way to play an audible sound is to use an instance of MediaPlayer (Android Open Source Project, 2022d). Unfortunately, MediaPlayer is a very obtuse part of the Android Runtime. It works on the principle of a state machine, and this state machine is well documented and understandable; but there is no way for code interacting with it to query the state, bar a binary "isPlaying" and "isLooping" variables. The only way to handle this is for the interacting code to manually keep track of what state it "should" be in, handle callbacks, and effectively reset and start over if an error is encountered[2].

In this project, the class written to handle this has an internal member, *isPrepared*, that keeps track of if the MediaPlayer instance is ready to play

[2]The MediaPlayer namespace has some named errors, but most errors encountered during this project were either "attempting to pause whilst paused", or just gave back numbers with no clear meaning.

the alert sound, from the beginning. The function to start playing checks this value, and does not call play if it isn't true; if it is, then it starts playing and sets the value as false. The function to stop playing checks if this value is false and if the *isPlaying* value is true, and if so pauses the player and seeks the track back to 0 seconds; a full call to the MediaPlayer's *stop()* method would require going through the "prepare" flow again. Seeking is also an asynchronous operation, so during creation of the MediaPlayer, the handler registers itself for "OnSeekComplete", and it is this callback function that sets *isPrepared* back to true. Through this, calls to the handler's start and stop methods are idempotent.

This is the simplest case for a handler for MediaPlayer. If this application was doing more complex things, such as playing multiple files, a more complex handler would have to be written. Regardless, it is interesting that there is no direct access to state for MediaPlayer; it suggests the internal functionality of MediaPlayer is too complex to represent the state to the user.

5.6 Testing and Test-driven development.

Beck (2002) writes that test-driven development is when "we drive development with automated tests"; and follow the two "simple rules": "Write code only if an automated test has failed", and "eliminate duplication". A perhaps less obtuse interpretation of this, is that test-driven development is the process of writing tests first, then writing code to make those tests pass.

This project started with the intention of following the TDD methodology. However due to time constraints and mismanagement, all of the code was written before implementing tests; hence there is very little to talk about in terms of test result progress throughout the project. Still, there is discussion to be had about testing in general.

5.6.1 Testing on Android

Android Open Source Project (2022b) says "the most important distinction for app developers is where tests run"; and hence the two overall classifications for android tests: Local tests which run on the development machine, and Instrumented tests which run on an Android device, emulated or real.

Local tests are more suited for the traditional "unit tests", as when running them the overall android runtime doesn't exist, so attempting to construct Android classes (for example, LocationManager and MediaPlayer) can end up failing unless they're mocked.

In comparison, Instrumented tests are ideal for "integration tests", as they are actually run on an Android Device; meaning the android runtime exists, and test code can use android classes and contexts. In the context

of the project, some instrumented tests were written in *AverageSpeedModelTest.kt*; one to test that the AverageSpeedModel is instantiated to the expected state, and one to test the basic tracking functionality. However, in part due to lack of documentation around the use of LocationManager test providers, this second test fails: the AverageSpeed is reported as 0, when it should be 17. This is likely due to a delay between calls to *setTestProviderLocation* and when the LocationManager actually calls the onLocationChanged callback; however, documentation on this could not be found. The solution would be manually adding a delay in the test code, but Android Instrumented tests cannot be ran in parallel on the same device, so a delay would slow down the test speed.

Appendix A

Development Log

Note that, as the LaTeX for this report is itself hosted in the repository, this git log contains commits not relevant to the application; where possible they have been removed, but some may remain.

Bibliography

(1943) Radio detection and ranging. *Nature*), 152(3857), 391–392, URL https://doi.org/10.1038/152391b0.

ACRO (n.d.) Police national computer services. Available online: https://www.acro.police.uk/PNC-services [Accessed 27/01/2022].

ageas (2019) How dangerous does a road have to be to get a speed camera? Available online: https://www.ageas.co.uk/solved/road-safety/how-dangerous-does-a-road-have-to-be-to-get-a-speed-camera/ [Accessed 25/01/2022].

Android Open Source Project (2019) Add the app bar. Available online: https://developer.android.com/training/appbar [Accessed 25/04/2022].

Android Open Source Project (2021a) Settings - Android Jetpack. Available online: https://developer.android.com/guide/topics/ui/settings [Accessed 09/03/2022].

Android Open Source Project (2021b) Support library. Available online: https://developer.android.com/topic/libraries/support-library [Accessed 26/04/2022].

Android Open Source Project (2022a) App resources overview. Available online: https://developer.android.com/guide/topics/resources/providing-resources [Accessed 25/04/2022].

Android Open Source Project (2022b) Fundamentals of testing android apps. Available online: https://developer.android.com/training/testing/fundamentals [Accessed 29/04/2022].

Android Open Source Project (2022c) LocationManager. Available online: https://developer.android.com/reference/android/location/LocationManager [Accessed 28/04/2022].

Android Open Source Project (2022d) MediaPlayer. Available online: https://developer.android.com/reference/android/media/MediaPlayer [Accessed 28/04/2022].

Android Open Source Project (2022e) Permissions on android. Available online: https://developer.android.com/guide/topics/permissions/overview [Accessed 26/04/2022].

Android Open Source Project (2022f) View binding. Available online: https://developer.android.com/topic/libraries/view-binding [Accessed 25/04/2022].

Apple Inc. (2014) Swift has reached 1.0. Available online: https://developer.apple.com/swift/blog/?id=14 [Accessed 25/04/2022].

Apple Inc. (2021) Apple maps: You'll like where this is going. Available online: https://www.apple.com/uk/maps/ [Accessed 25/11/2021].

Apple Inc. (n.d.) Enrollment - support - apple developer program. Available online: https://developer.apple.com/support/enrollment/ [Accessed 18/10/2021].

Aycock, J. (2003) A brief history of just-in-time. *ACM Comput. Surv.*), 35(2), 97–113, URL https://doi.org/10.1145/857076.857077.

BBC News (2016) Average-speed camera coverage in uk 'doubles' in three years. Available online: https://www.bbc.co.uk/news/uk-36399408 [Accessed 26/10/2021].

Beck, K. (2002) *Test-Driven Development By Example*). Pearson Education, Inc.

Butler, S. (2022) What is sideloading, and what are the risks? Available online: https://www.howtogeek.com/773639/what-is-sideloading-and-should-you-do-it/ [Accessed 26/04/2022.

Carbuyer (2021) Average speed cameras: how do they work? Available online: https://www.carbuyer.co.uk/tips-and-advice/160228/average-speed-cameras-how-do-they-work [Accessed 22/02/2022].

Dart project authors (2022) Dart programming language. Available online: https://dart.dev/ [Accessed 25/04/2022].

de Podesta, M. (2002) *Understanding the properties of matter*). Second edition edition, Taylor & Francis.

Doppler, C. (1842) Über das farbige licht der doppelsterne und einiger anderer gestirne des himmels. Originally published in *Abhandlungen der k. böhm. Gesellschaft der Wissenschaften (V. Folge, Bd. 2, S. 465-482)*; the 1903 reprint is available online at https://archive.org/details/ueberdasfarbigel00doppuoft/mode/2up [Accessed 27/01/2022].

Eikvil, L. (1993) Optical character recognition. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.4980&rep=rep1&type=pdf [Accessed 01/02/2022].

F-Droid Limited and Contributors (2022) F-droid. Available online: https://f-droid.org/en/ [Accessed 26/04/2022].

Google LLC (2021a) Android jetpack. Available online: https://github.com/androidx/androidx [Accessed 29/10/2021].

Google LLC (2021b) Flutter. Available online: https://flutter.dev/ [Accessed 13/10/2021].

Google LLC (2021c) How google play works. Available online: https://play.google.com/about/howplayworks/ [Accessed 26/04/2022].

Google LLC (2022) Android studio. Available online: https://developer.android.com/studio [Accessed 26/04/2022].

Google LLC (n.d.a) Google Assistant, your own personal Google. Available online: https://assistant.google.com/ [Accessed 4/11/2021].

Google LLC (n.d.b) Google Maps: Explore and navigate your world. Available online: https://www.google.co.uk/maps/about/ [Accessed 25/11/2021].

GOV.UK (n.d.) Speed limits - GOV.UK. Available online: https://www.gov.uk/speed-limits [Accessed 4/11/2021].

Gradle Inc. (2022) Gradle build tool. Available online: https://gradle.org/ [Accessed 28/04/2022].

Hecht, E. (2002) *Optics*). Fourth edition international edition, Addison Wesley.

IEEE (2020) Ieee standard letter designations for radar-frequency bands. *IEEE Std 521-2019 (Revision of IEEE Std 521-2002)*), 1–15.

Jenoptik Traffic Solutions UK (n.d.) Specs - jenoptik. Available online: https://www.jenoptik.co.uk/product/specs/ [Accessed 27/10/2021].

King, B., Surtees-Goodall, S. & Jeanes, M. (2011) The road safety partnership grant programme. Available online: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/4004/report.pdf [Accessed 25/01/2022].

Krill, P. (2011) Jetbrains readies jvm-based language. Available online: https://web.archive.org/web/20190907161741/https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html [Accessed 17/10/2021].

Lardinois, F. (2019) Kotlin is now google's preferred language for android app development. Available online: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/ [Accessed 17/10/2021].

Légifrance (2012) Code de la route, article r413-15. Available online: https://www.legifrance.gouv.fr/codes/article_lc/LEGIARTI000025111528/2021-10-14 [Accessed 14/10/2021].

Moboware, Inc (2022) Macincloud. Available online: https://checkout.macincloud.com/ [Accessed 27/04/2022].

.NET Foundation Contributors (2021) .net multi-platform app ui (.net maui). Available online: https://github.com/dotnet/maui [Accessed 14/10/2021].

NPCC (n.d.) Automatic number plate recognition (anpr). Available online: https://www.npcc.police.uk/FreedomofInformation/ANPR.aspx [Accessed 01/02/2022].

nventive Inc. (2021) Uno platform. Available online: https://platform.uno/ [Accessed 14/10/2021].

Oracle Corporation (2022) Openjdk. Available online: https://openjdk.java.net/ [Accessed 25/04/2022].

Owen, R., Ursachi, G. & Allsop, R. (2016) The effectiveness of average speed cameras in great britain. *Technical report*), Royal Automobile Club Foundation, URL https://www.racfoundation.org/wp-content/uploads/2017/11/Average_speed_camera_effectiveness_Owen_Ursachi_Allsop_September_2016.pdf.

RAC (2022) Smart motorways - what are they and how do you use them? — Video guide. Available online: https://www.rac.co.uk/drive/advice/driving-advice/smart-motorways/ [Accessed 25/01/2022].

Rakha, H., Lucic, I., Demarchie, S. H., Setti, J. R. & Aerde, M. V. (2001) Vehicle dynamics model for predicting maximum truck acceleration levels. Available online: https://doi.org/10.1061/(ASCE)0733-947X(2001)127:5(418) [Accessed 01/03/2022].

Scaleway SAS (2022) Simple cloud pricing. Available online: https://www.scaleway.com/en/pricing/#apple-silicon [Accessed 27/04/2022].

Shafirov, M. (2017) Kotlin on android. now official. Available online: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/ [Accessed 17/10/2021].

Statcounter Limited (2021) Mobile operating system market share worldwide. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed 17/10/2021, 25/04/2022].

TomTom International BV. (2021) Turn your phone into a tomtom device. Available online: https://www.tomtom.com/en_gb/navigation/mobile-apps/go-navigation-app/ [Accessed 25/11/2021].

Truvelo Ltd. (2020) TRUVELO "D-CAM". Available online: https://www.truvelouk.com/products/d-cam-speed-red-light-camera/ [Accessed 25/11/2021].

W3C SVG Working Group (2010) Scalable vector graphics (svg). Available online: https://www.w3.org/Graphics/SVG/ [Accessed 25/04/2022].

Waze Mobile (2021) Waze. Available online: https://www.waze.com/ [Accessed 25/11/2021].

Wolff, C. (n.d.) Doppler- effect. Available online: https://www.radartutorial.eu/11.coherent/co06.en.html [Accessed 27/01/2022].