BTech 451B

ASIST - Asset History In Real Time

Development of a Mobile Computing Application

Anthony Peek

ID 2628256
UPI apeee229

26th October 2015

Acknowledgements

Kodie Wixon, Team Manager, Software Development
Opus International Consultants Ltd.

Roquito Lim, Senior Financial Analyst
Opus International Consultants Ltd.

Andrew Bruce, Business Group Manager
Opus International Consultants Ltd.

Duan Zhao, Information Analyst / Asset Manager
Opus International Consultants Ltd.

Dr. Sulo Shanmuganathan, Chief Technical Officer
Opus International Consultants Ltd.

Dr. Sathiamoorthy Manoharan, Academic Supervisor
The University of Auckland

Executive Summary

Opus, an infrastructure consultancy firm have asked for ASIST, a mobile application theorised in a recent in-house innovation competition, to be developed. ASIST's, or asset history in real time's goal is to convert the significantly large amounts of data collected for their structures such as buildings and roads from convoluted and expansive, to intuitive and minimal. This has lead to a mobile augmented reality solution idea by Andrew Bruce.

Augmented reality in this case utilises the mobile device's location with components such as the GPS, compass and accelerometer to derive the absolute positioning and angle of the device. When paired with the device's camera, information related to this positioning can be visualised in real time against the real world using a data layer. Information collected about assets such as condition data, past repairs, and future plans can then be displayed on this data layer in real time while working in the field. This is a far more practical solution that can be understood quickly and without prior knowledge of the domain, compared to the current data tables used which require processing and transformation to gain understanding from.

The project is to transform this idea into a functional application. The first 12 weeks, or phase one of this project was assigned to scoping and design. The key decision required in this time was the choice of platform to use. First, an introduction to mobile programming and augmented reality is presented (PART I). Next, research and reasoning with respect to the implementation of ASIST is presented (PART II), including research of current comparable applications, and platform and hardware limitations.

After making the decision to pursue native Android development, PART III discusses the development process in detail, introducing the different components such as GPS, accelerometer, and compass required by the application, and the integration of these to create the augmented data layer which displays the road surface. This process presented many challenges, resulting in many changes being made, and limitations being introduced throughout development.

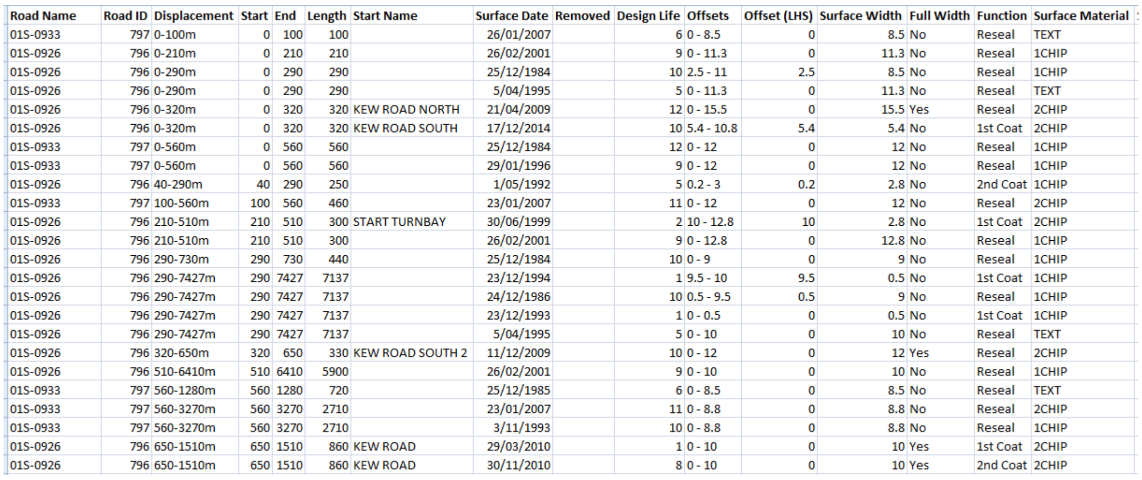
Finally, an evaluation of the application is made, and areas for future work discussed (PART IV). Due to the challenges faced throughout development, the scope of the project was reduced to only the core component of ASIST - the visualisation of roads. Although the additional features were not completed, this core functionality has been completed to a satisfactory level which solves the given problem of road condition visualisation.

Contents

I. INTRODUCTION

1.1 Problem Description

ASIST, or asset history in real time, is one of the product ideas to emerge from the Opus 2014 Big Ideas Innovation Competition. Designed by Andrew Bruce and further documented by Duan Zhao, ASIST aims to simplify the task of translating asset data such as the condition of roads and buildings into a comprehendible form. A vast amount of data is gathered and stored to describe these assets, and currently, this data is simply given to the user in tabular form, as below:



Road Name	Road ID	Displacement	Start	End	Length	Start Name	Surface Date	Removed	Design Life	Offsets	Offset (LHS)	Surface Width	Full Width	Function	Surface Material
01S-0933	797	0-100m	0	100	100		26/01/2007		6	0 - 8.5	0	8.5	No	Reseal	TEXT
01S-0926	796	0-210m	0	210	210		26/02/2001		9	0 - 11.3	0	11.3	No	Reseal	1CHIP
01S-0926	796	0-290m	0	290	290		25/12/1984		10	2.5 - 11	2.5	8.5	No	Reseal	1CHIP
01S-0926	796	0-290m	0	290	290		5/04/1995		5	0 - 11.3	0	11.3	No	Reseal	TEXT
01S-0926	796	0-320m	0	320	320	KEW ROAD NORTH	21/04/2009		12	0 - 15.5	0	15.5	Yes	Reseal	2CHIP
01S-0926	796	0-320m	0	320	320	KEW ROAD SOUTH	17/12/2014		10	5.4 - 10.8	5.4	5.4	No	1st Coat	2CHIP
01S-0933	797	0-560m	0	560	560		25/12/1984		12	0 - 12	0	12	No	Reseal	1CHIP
01S-0933	797	0-560m	0	560	560		29/01/1996		9	0 - 12	0	12	No	Reseal	1CHIP
01S-0926	796	40-290m	40	290	250		1/05/1992		5	0.2 - 3	0.2	2.8	No	2nd Coat	1CHIP
01S-0933	797	100-560m	100	560	460		23/01/2007		11	0 - 12	0	12	No	Reseal	2CHIP
01S-0926	796	210-510m	210	510	300	START TURNBAY	30/06/1999		2	10 - 12.8	10	2.8	No	1st Coat	2CHIP
01S-0926	796	210-510m	210	510	300		26/02/2001		9	0 - 12.8	0	12.8	No	Reseal	1CHIP
01S-0926	796	290-730m	290	730	440		25/12/1984		10	0 - 9	0	9	No	Reseal	1CHIP
01S-0926	796	290-7427m	290	7427	7137		23/12/1994		1	9.5 - 10	9.5	0.5	No	1st Coat	1CHIP
01S-0926	796	290-7427m	290	7427	7137		24/12/1986		10	0.5 - 9.5	0.5	9	No	Reseal	1CHIP
01S-0926	796	290-7427m	290	7427	7137		23/12/1993		1	0 - 0.5	0	0.5	No	1st Coat	1CHIP
01S-0926	796	290-7427m	290	7427	7137		5/04/1995		5	0 - 10	0	10	No	Reseal	TEXT
01S-0926	796	320-650m	320	650	330	KEW ROAD SOUTH 2	11/12/2009		10	0 - 12	0	12	Yes	Reseal	2CHIP
01S-0926	796	510-6410m	510	6410	5900		26/02/2001		9	0 - 10	0	10	No	Reseal	1CHIP
01S-0933	797	560-1280m	560	1280	720		25/12/1985		6	0 - 8.5	0	8.5	No	Reseal	TEXT
01S-0933	797	560-3270m	560	3270	2710		23/01/2007		11	0 - 8.8	0	8.8	No	Reseal	2CHIP
01S-0933	797	560-3270m	560	3270	2710		3/11/1993		10	0 - 8.8	0	8.8	No	Reseal	1CHIP
01S-0926	796	650-1510m	650	1510	860	KEW ROAD	29/03/2010		1	0 - 10	0	10	Yes	1st Coat	2CHIP
01S-0926	796	650-1510m	650	1510	860	KEW ROAD	30/11/2010		8	0 - 10	0	10	Yes	2nd Coat	2CHIP

Table 1. Current Road Assessment and Maintenance Management (RAMM) data

Although descriptive, this representation of data requires a significant understanding of the domain and is overall difficult and time consuming to analyse. The proposed solution to this, ASIST, aims to use augmented reality to visualise this data in real time as shown in figure 1:

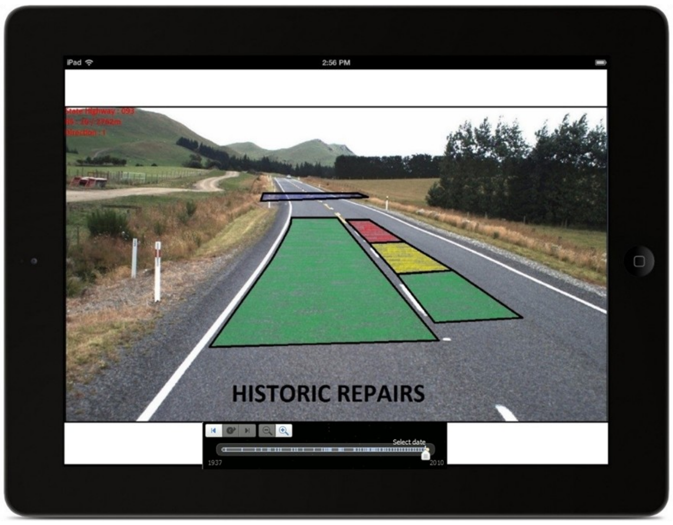


Fig 1. Road Surface Condition

1.2 Introduction to Mobile Application Development

In recent years the world has seen a shift in personal computer use. Once bounded to the family home or work office, computers have been reshaped, resized, and re-imagined and are now with us 24/7, either sitting in one's pocket, attached to a wrist, or even as part of eyewear products. Due to this, and the fact that approximately 93% of the population have an active mobile device, mobile development has undoubtedly become one of the biggest forms of software development for the consumer market [1].

Beginning in the era of "The Brick", or the Motorolla DynaTAC 8000X, Nokia introduced the arcade video game Snake to their early monochrome phones in the 1980s [2]. Other games followed and the phone developed into much more than a communications device. Unlike back then when it were solely the device manufacturers who were capable of application development, nowadays due to the very low barriers of entry into mobile development, anyone and everyone wanting to try and take advantage of the huge demand for applications can get involved in mobile application development. This has lead to, as of July 2014, over three million applications being available to download from the leading application stores including Google Play (1.3 million) and the Apple App Store (1.2 million) [3]. App stores like these are the main point of sale for mobile applications and allow for standalone and multinational developers alike to sell their products to the world, where the app store will then take a certain percentage of the sale price if it were a paid for application. Many different categories of applications exist for smart phones and tablets. In Apple's app store, 20 different categories are listed, with over a fifth of applications being games, followed by business, education, and lifestyle, each with approximately 10% of the market share [4].

In mobile development, as with any form of software development, many design principles must be considered and enforced throughout implementation to produce a successful application. Most importantly, and what most of the following factors will be classed under is the user experience of the application. If the system isn't responsive, intuitive, and correct, the application will not be successful. A good idea may make an app visible in the marketplace, but in terms of successfulness once the user has an application installed, it does not matter what the program does, as long as it delivers on what was promised in a way that satisfies the user. With that said, the first step to any software development project should be to research and investigate the market for what technology already exists, and whether the current products are already too established in the market to enter, or if there is a gap in the market that your idea can take advantage of.

Once the idea is proven, countless other factors have to be considered. If the application is a contracted project, then user and data requirements are of up most importance and should be the driving factors behind the entire operation. Collaboration between developers and clients is therefore paramount and should be taken as seriously as the development itself. Numerous other factors make up mobile application development, most of which fall under the following areas of development techniques and technologies:

A. Platforms and Languages

With the rise of mobile development in recent years, there are now countless tools and technologies at the disposal of mobile application developers. One of the first things a developer must decide in terms of technology is what mobile platform or operating system(s) he wants the application to be compatible with. This will influence what programming language(s) will be used. If the developer only needs to target a single operating system, then he can complete the project using platform specific native programming. However if multiple platforms exist, then native programming across multiple systems can be used, which requires knowledge of multiple languages and the capability of translating code between the various languages. To negate the need for this application translation, a cross platform application such as an HTML5 web app can be created [5].

B. Programming Environment

Once a target platform and language has been chosen, then the programming environment can be chosen. Anything from a simple text editor such as Notepad++ with language specific formatting to a fully inclusive integrated development kit (IDE) can be used, each with their own advantages and disadvantages. If a developer is confident using the command line and doesn't need syntax auto correcting, then text editors are a light weight tool that allows for a program to be created without the need of an abundance of extra files and programs. On the other hand, IDE's such as Visual Studio and Eclipse allow for the application to be created, tested, and published all from within the application, with the downsides of extra files being generated by the IDE and the developer becoming dependent on using such tools and not being able to then go and work on that project easily in a text editor.

C. Third Party Libraries

Third party libraries - functionality built by other developers, is another tool available to developers, and one that should most definitely not be ignored. Unless development is on the cutting edge of technology or is in an area where no development has been done before, chances are someone has already created something that will be of use and will reduce development time, and therefore increase productivity. The only downside of such third party library use however is the potential learning time required to understand and take advantage of the library. If documentation is provided and the coding style used is intuitive, then uptake of the library should be straightforward. However the opposite is possible, and a lack of, or poorly constructed documentation and code can cause adoption of said libraries to be troublesome and time consuming.

D. Structure

As is the case with tools and technology, there are also numerous frameworks, methodologies and techniques that can be applied to mobile application development. In any development project, even one being completed by a single individual, it is important to maintain a well structured program. A well structured program achieves two things—efficiency via using non repeating code, and readability via intuitively well defined sections. By correctly creating classes and methods to complete recurring functions, development time should be reduced by not needing to repeat code, and future developments, either internal or external to this project may be able to use these functions as opposed to them not being able to if the functions were written in line and specified for the current program directly. Additionally, a well structured program is easy to understand by both the original author, as well as potential future users of the code base.

E. Maintainability

Continuing on from having a well structured program, other efforts can be made to ensure maintainability of a program, and other reasons exist for keeping a program in a well maintained state. Firstly, the code itself should be understandable by using intuitive function and variable names. However in the presence of more elaborate programs, informative comments should be used to explain the flow of an application and its components. Another need for maintainability is driven by future proofing and version control, this is especially important in mobile applications. Many mobile applications receive updates on a regular basis and hence it is imperative that the program is kept in a manner which allows for future changes to be made and previous versions to be kept safely. Git provides an easy to use and free version control system, as well as plug-ins that can be used in commonly used IDEs such as Eclipse. Also to handle regular updates and the issues that can arise by some users updating the application and others not, it is important to create both forward and backward compatibility between versions when needed. For forward compatibility, the more difficult of the two, assumptions about future releases have to be made and current components created to be as unrestrictive as possible on future releases. Backwards compatibility then requires newer released software to accept input generated from older standards. Again, if the development is maintained using correct version controlling, then it should be easy for developers to understand what will and won't be compatible with older systems.

F. Efficiency

At a lower level, applications are made competitive based on how efficient their algorithms are. Efficiency is measured in many ways such as run-time, memory use, and storage use. For most algorithms such as sorting and searching, there is in general no method that will perform best in all measurements, so a trade-off has to be made. Fortunately however, when it comes to the vast majority of mobile applications, they do not require striving for such low level efficiencies as both the data involved and algorithms run on that data are generally relatively small and straightforward, respectively. With that said, big data has become very important in the IT industry in recent years and is spreading over various platforms, including mobile [6]. Mobile devices are fortunately now equipped with in most cases far more memory, storage and CPU speed than is required to run the majority of mobile applications, such as the recently released Samsung S6 with 3GB of RAM, 1.5GHz quad-core processor, and up to 128GB of storage, so should be capable to run more intensive algorithms that weren't necessarily developed with efficiency in mind [7].

G. Development Process

At the highest level of development—choosing the process in which to complete the project, many options exist. Common methodologies include iterative and incremental development, waterfall, spiral development, rapid application development, and the growing in popularity, agile methodology [8]. Discussed will be two of the main methodologies; waterfall and agile. Waterfall is the most traditional approach and has a linear flow of events from designing and planning through to coding, then testing and delivery. This approach makes for easy measurement of work completion and removes the need of much client communication once the initial meetings planning the program are complete. Agile on the other hand divides time into short 'sprints' of usually weeks in which a deliverable is required and reviewed by the customer. This means that the customer can have a larger input on the project. Additionally, if completion dates are of concern for the application, then agile is able to have more basic versions of the software available earlier than a waterfall approach which sequentially develops functions to a high level of quality [9].

1.3 Differences Between Mobile & Embedded Application Development

But what makes mobile development different to regular embedded systems development? Although similar overall, many additional requirements for mobile application development exist, including the following as described by Wasserman [10]:

A. Additional Inter-Application Communication

Web services aside, the overall interaction between embedded applications is generally minimal in comparison to that of mobile applications. With typically a large number of applications coming from a variety of distributors, data is often shared and translated between applications on mobile devices to provide users with additional services. Leveraging and maintaining this capability can be vital to creating a useful and well performing application depending on the domain of use.

B. Sensor Handling

As will be elaborated throughout this report for ASIST in particular, hardware sensors of mobile devices drastically change what it means to interact with the system. Smart mobile devices have access to an array of hardware sensors, and in Android these are divided into the three broad categories of: motion, environmental, and position sensors. Motion sensors measure the rotational and acceleration forces along three axes and includes gravity and rotational vector sensors, accelerometers, and gyroscopes. Environmental sensors include barometers, photometers, and thermometers which measure air pressure, illumination, and temperature, respectively. Finally, position sensors measure the position of a device via orientation sensors and magnetometers. Additional hardware sensors which are less common in embedded systems that mobile devices make use of are cameras, microphones, and touch screens. Using any number of these sensors provides the application with an incredibly large amount of information which can be leveraged in a variety of ways. It is becoming increasingly uncommon for mobile applications not to make use of any of these sensors, meaning that it is becoming increasingly important for developers to use this information to make effective applications. Embedded environments in comparison don’t far from mouse, keyboard and any domain specific inputs, with variables such as location only being able to be estimated via the corresponding IP address unless additional equipment is used.

C. Families of Hardware and Software Platforms

Compared to embedded systems which often execute custom-built programs created for the properties of the device, mobile devices are often expected to support applications developed for a range of devices, and support applications developed for different operating system versions. Therefore it is important for a developer to be aware of forward and backward compatibility across API versions, and to understand portability limitations [11].

D. Security

In the most “closed” cases, embedded devices work in isolation and are therefore difficult to attack. Mobile devices on the other hand are comparatively “open”, allowing for installation of potentially harmful applications capable of extracting local data.

E. User Interfaces

Embedded application development allows for complete control of the user interface. Although still in control of the user interface in mobile development, it is generally accepted that developers adhere to externally developed user interface guidelines, often included in the SDKs used for development.

F. Difficulty of Testing

Embedded applications are easily tested by emulators or traditional methods. Due to some of the differences mentioned above, particularly sections *B.* and *C.*, difficulties are added for testing mobile applications. When using sensor information, desktop emulators are unable to be used due to not having access to those sensors. Likewise, emulators cannot always be used to test across all platforms.

G. Power Consumption

Not an issue with desktop machines, but the power consumption of mobile applications can vary and as a result, reduce the device's battery life. Efficient use of sensors and other power depleting services is required when developing to increase the longevity of the device and therefore application.

1.4 Introduction to Augmented Reality

Continually growing in popularity to solve problems over a variety of domains, augmented reality (AR) refers to the combination of digital and tangible information. Related to virtual reality (VR), where an artificial space is created for the user to interact with via primarily vision and movement, AR replaces the artificial space with their real perspective of the world, populating this perspective with corresponding digital information such as three dimensional shapes and textual data [12]. Both of these fall under the broader category of mixed reality (MR), which is defined more loosely as the general integration between the real and virtual worlds and can be modelled by the following continuum [13]:

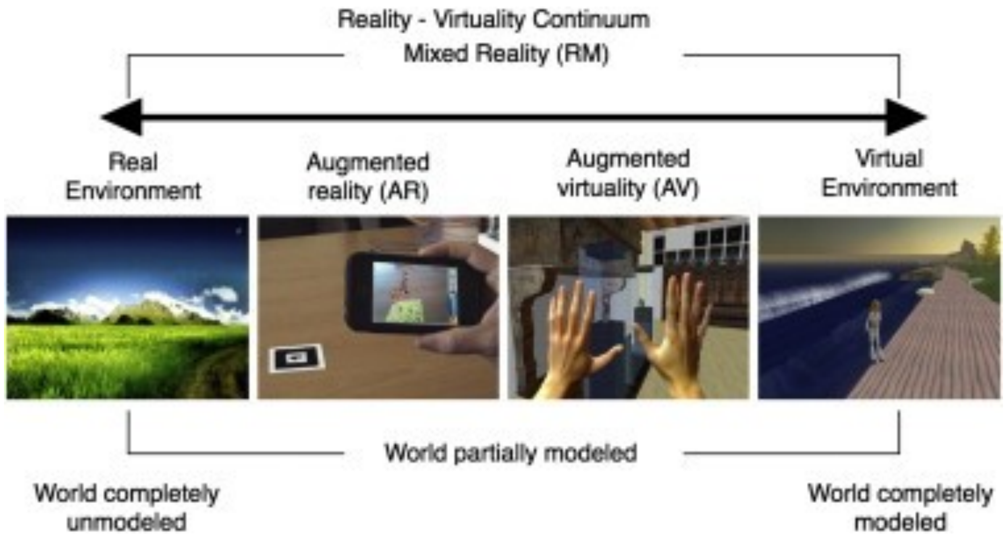


Fig 2. Virtuality Continuum [14]

With ever increasing computing power, image recognition techniques, and locational awareness, AR is being realised as a viable solution for an ever expanding array of problems. Two examples include:

Home Decorating - Given images of a living space and a catalogue of furniture items, users can effortlessly trial a large range of furniture arrangements in their home:

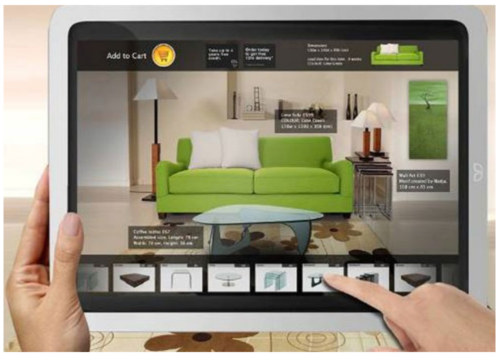


Fig 3. Home Decorating AR [15]

Clothes Shopping - Similar to home decorating, user's can trial clothes that interact with their movements without needing to wear them:

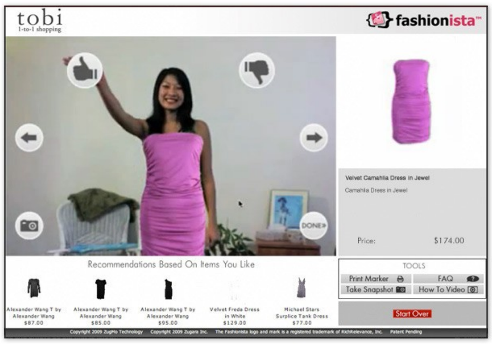


Fig 4. Clothes Shopping AR [16]

Closer aligned to the project at hand, location based augmented reality products are just as popular as these primarily image recognition applications above, especially when considering deployment onto mobile devices. These will be showcased in the forthcoming implementation research and reasoning section.

1.5 AR Requirements & Foundations

A large amount of research and work has been conducted to develop both the visualisation of the data layer, as well as algorithms to link real and virtual spaces [17]. However before unrestricted access to smart devices capable of all of the required hardware functions became available, early work was concerned with the fundamentals of AR. Such work which still applies today includes the Tracking Requirements for Augmented Reality by Ronald Azuma [18]. In a completely virtual environment, approximations of the user's orientation and position are adequate to create a sufficient visualisation since "small errors are not easily discernible because the user's visual sense tends to override the conflicting signals from his vestibular and proprioceptive systems."

In AR however, small mismatches between the real world and the digital objects populating that space due to a mistake in the calibration of tracking the user are easily noticeable i.e. when a virtual object doesn't align with the real world correctly. Millimetres in distance and fractions of a degree in rotational error of the user tracking was shown to cause large errors when dealing with objects two metres away, something that will need to be taken into account when developing and testing the ASIST application.

Once accuracy is assumed sufficient, representation of these virtual three dimensional objects is required. Typically in this style of application, objects to be rendered and their positional and dimensional information will be stored in a database or at the simplest level, an array like structure which is to be accessed when the device recognises that the object should be in it's frame of view. The next step is correctly representing these shapes. The standard approach to displaying three dimensional objects is through the use of model view and projection matrices. A model matrix is used to translate, rotate, and scale the model. The view matrix then translates and rotates everything in the world to put the camera in the origin. Finally the projection matrix takes those points in the three dimensional space and projects them onto a two dimensional screen as illustrated:

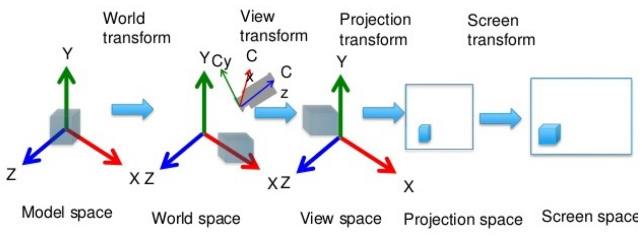


Fig 5. Model view projection transformation [19]

Kutulakos and Vallino [20] explain the geometrical foundations of this conversion from object to image. The projection of the object requires knowing the combined effect of the object-to-world, world-to-camera, and camera-to-image transformation, described by equation (1):

$$\begin{bmatrix} u \\ v \\ h \end{bmatrix} = \mathbf{P}_{3\times4}\mathbf{C}_{4\times4}\mathbf{O}_{4\times4}\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \qquad (1)$$

Where $[x\ y\ z\ w]^T$ is a point on the virtual object, $[u\ v\ h]^T$ is it's projection, O_{4x4} and C_{4x4} are the matrices corresponding to the object-to-world and world-to-camera transformations, and P_{3x4} is the matrix modelling the object's projection onto the image plane.

1.6 Mobile AR

Because the interaction between realities occurs in real time, AR opens the door towards a paradigm shift with how everyday computing takes place. This is then further exaggerated when applied to mobile computing which is becoming increasingly common, and in need of alternative interaction methods [21]. Due to the ever increasing hardware capabilities of mobile devices, such as camera, orientation, and location technologies, mobile platforms are becoming increasingly context-aware [22]. This allows for various pieces of information unobtainable by a desktop application to be leveraged and applied to the given problem, such as real time local event information.

1.7 User Evaluation of Mobile AR

On top of the heavily documented areas of AR requirements and technology, an important issue to discuss, especially when developing for a large number of clients and potential users is the user satisfaction of the system. Olsson et al. [22] focus on assessing what is required by an AR application by conducting user research regarding expectations and acceptance by potential users. A collection of AR scenarios ranging from practical uses such as location navigation while travelling, to leisure focused activities such as virtual street art creation where the user can graffiti the real world via the augmented layer are trialled in the analysis. From user surveys it was found that participants generally perceive AR solutions concerned with practical cases such as navigation and furniture arrangement to be more useful than those of social and personal benefit such as street art or a virtual mirror for testing makeovers. The effects of the participants' technology orientation, as well as other factors such as age and willingness to share information on the internet were also assessed against their attitude towards AR. Age differences provide inconsistent feedback with no significant correlations found with respect to AR acceptance. Gender however was indeed found to have an impact, with males being more positive towards AR services than females overall. In addition to this, and as expected, those that were highly technologically orientated were found to have a more positive acceptance of AR products. Due to ASIST being a practical use case for AR, and no known problems with respect to the other features discussed at Opus being apparent, the outlook for user acceptance of the proposed application is promising.

1.8 Conceptual ASIST

A. The Solution

Certain characteristics of data can only be seen when data is represented graphically. Hence a more intuitive approach to interpreting this data is through visualisation [24]. This has lead to the idea of an augmented reality visualisation of the data. Assuming that intended results are like that displayed below, all gathered data will be intuitively recognisable, requiring little specialised knowledge of the domain, and increasing both productivity and user tolerance and acceptance of the system [25].

B. Application Functionality

ASIST is intended to contain a handful of related features which all revolve around the idea of augmenting information of the asset in real time. This will be known as the 'live view' function throughout this document. With respect to roading, the condition, such as skid resistance and age of seal will be visualised with simple colour coding to represent the status of the road, as displayed in figure 6.

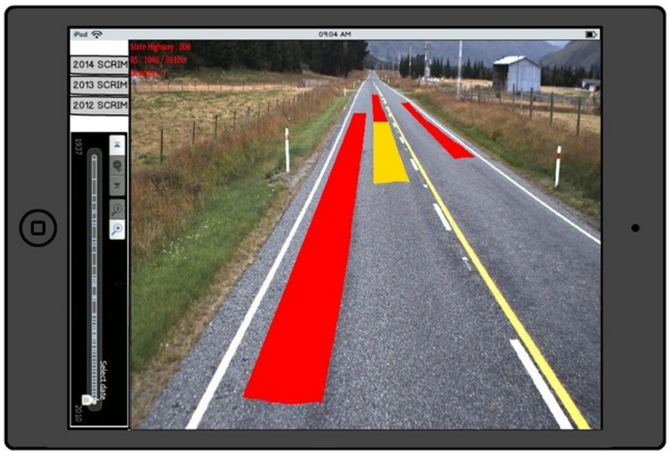


Fig 6. Road Surface Skid Resistance

Using a colour scheme that is recognisable by the user, such as green for good condition and red for bad, or the use of an intuitive legend that allows for more detailed data to be represented maximises the information gained from the application, while minimising complexity and user resistance of adoption. This data can then be used in other domains such as determining if there were any roading factors involved during a crash investigation. It is key that the data is understandable in this scenario due to outside users such as the police force needing to interpret the road condition correctly and easily.

Another augmented reality feature intended to be included in ASIST is the process of inventory validation. Assuming high location accuracy of the device being used, the application should be able to be used in the field to validate if the stored data of an object such as a culvert/drain pipe is correct in relation to the real world. Figure 7 illustrates such testing.

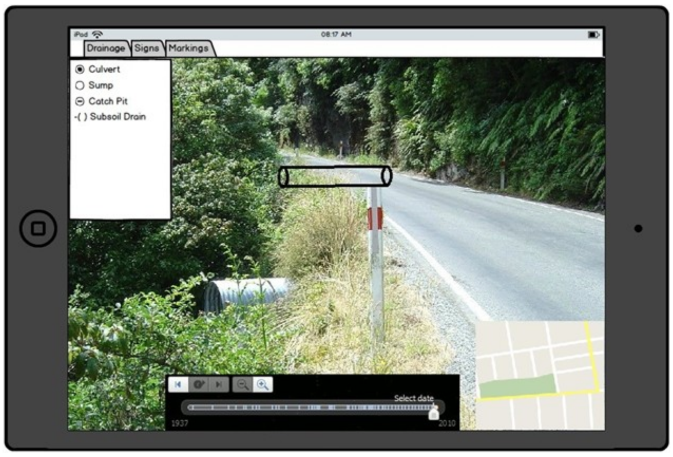


Fig 7. Inventory Validation—Misplaced Culvert

The last main augmented reality feature pertains to structures such as buildings and bridges. ASIST should be able to compare a current construction site or structure with either previous historic images and information as shown in figure 8, or in the case of ongoing construction, 3D rendering of the proposed finished product. Again this simplifies and speeds up the process of translating data into usable knowledge. This feature requires the same location based information as the roading component, but utilises a more elaborate technique for displaying data, so is planned to be implemented only after the roading component is complete.

As seen in the images, all of the functions are planned to have a historic time slider to allow users to quickly compare and contrast the condition of assets over different periods of time, allowing for trends in degradation of assets to be analysed and hence decisions on future improvements to be made while on site.

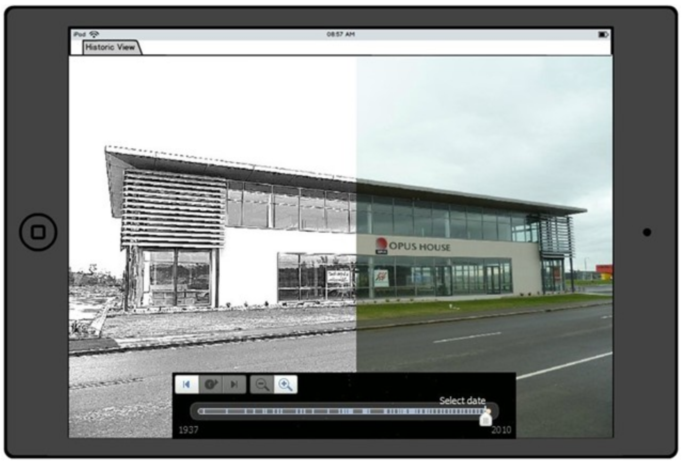


Fig 8. Asset Image Comparison

C. Use Cases

Now that the application and it's overall functionality is understood and reasoned, specific use cases will be discussed and their relation to the back-end asset manager explained. The asset manager is a more traditionally laid out interface that gives users the ability to access and modify details of an asset via text and other inputs. It works alongside the aforementioned live view function and stores the data required to visualise the information in that format.

1. Management Portal—KPI Performance and Report Generation

Throughout the process of a construction project, the site manager will want to assess and document the progress of the work being done. The user will have ongoing milestones and goals which are to be achieved, and requirements which must be compared to key performance indicators (KPI). This view will provide a real time dashboard of the current project and will allow the user to generate reports regarding KPI, maintenance costs, custom reports and more. These can then be accessed from both the mobile application and desktop applications that are connected to the Opus network.

2. Asset Information Portal

Accessible from either the associated live view function, or directly through the asset manager, this view will provide a listing of nearby assets while in the field, or a list of assets at a location specified by user input. The user can then select which asset he/she would like to analyse. Information specific to that asset, similar to that observed in the earlier data table, will be displayed.

This includes;

Location data	- New Zealand State Highway Treatment Length, Offset, GPS coordinates etc.
Functionality	- Use of the asset.
Dimensions	- Length, width, height etc.
Age	- Time since installation.

3. Condition Data

Extending from the asset information portal, the user can then view and update stored condition data of a given asset. This view combines features from the previous two use cases, leveraging the performance reports written in the management portal and displaying asset information. Other related condition information of the asset is also included, such as current and previous photos and condition descriptions as seen in the live view function being accessible.

4. Maintenance Data

When a request to assess asset maintenance information is made, this portal will give access to; previous work undertaken, such as initial construction and recent repairs, current maintenance work being completed on the asset, and reports for future construction endeavours on the asset.

5. Asset Photo Library

When a user wishes to update the current photos on hand for an asset, the photo library is used. Either linked to an external camera function or preview-able from within ASIST, the user can take, annotate, and save an image. This will be automatically associated to the asset in question, derived from location and perspective data of the device.

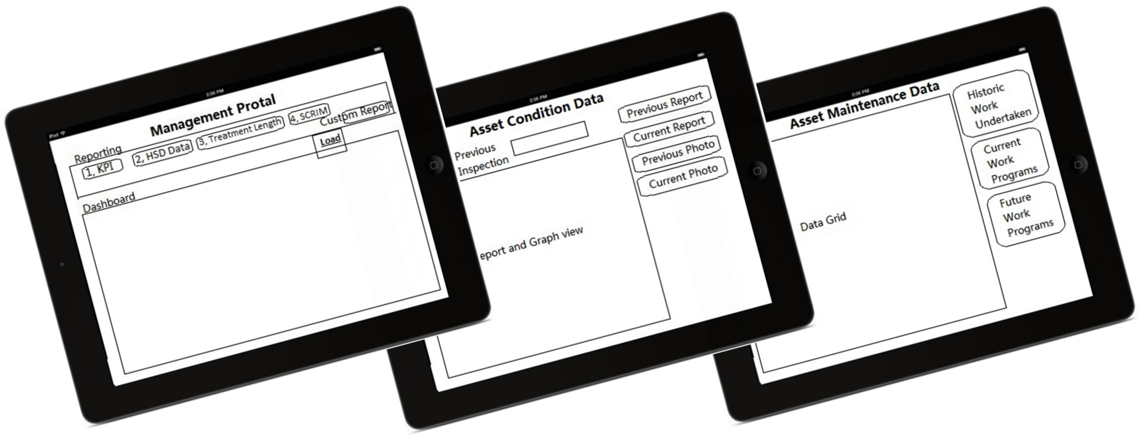


Fig 9. Asset Manager—Management Portal, Asset Condition Data, Asset Maintenance Data

1.9 Student Role & People Involved

The above information outlining the vision and capabilities of the application has been derived from initial concept documents developed by Andrew Bruce and Duan Zhao. The student's role is to make it reality. Chief technical officer Sulo Shanmuganathan has passed this project to senior financial analyst Roquito Lim who has in turn given myself and software development team manager Kodie Wixon the project to complete. Kodie acts as project supervisor, giving tasks to complete and ensuring project requirements and deadlines are met.

During the scoping and design phase of the first three months, the task of assessing the potential development paths of ASIST and making the decision, with Kodie's support on how to proceed with development (II. Implementation Research and Reasoning). The following three months saw the student play the leading role in the project, undertaking the majority of development and making most development decisions (III. Development Process).

The intended outcome by submission time was to at a minimum have completed a working prototype with the core live view functionality. Although this has been achieved, design decisions have been continually changed due to difficulties faced to ensure a deliverable, leaving room for future work to be completed.

II. IMPLEMENTATION RESEARCH & REASONING

Initial meetings with, and documentation from Opus had suggested for the development of ASIST to be undertaken via either a web based application, or a Windows based native application. Executives pushed for a web-based implementation due to the cross platform operability that it would achieve, allowing all current company devices, independent of operating system, to be compatible with a single implementation of the ASIST application. However, from the development side:

> "The preferred software platform for the development of ASIST at this point in time is a Windows based platform as this is the current operating system within Opus so integration with existing infrastructure and support should be less complicated." [26]

It was also mentioned however that the platform that ASIST would operate on is not critical, and that the main factors when choosing a platform are the availability of key components within the hardware.

Before researching specific platforms and their compatibility with an augmented reality application, comparable, already completed augmented reality solutions were researched in an attempt to adapt or use them as a reference while developing ASIST, and to discover what tools and technologies were used in the development process. With augmented reality applications being available on the Apple Store since 2009, it came as no surprise to discover many applications that fulfilled a similar purpose as the plans of ASIST [27].

2.1 AR Overview

Augmented reality solutions can generally be divided into two categories; either location based, or image recognition based. Location based services use the device's GPS to determine the user's location, and the device's accelerometer to determine the angle the mobile device is being held on and hence what it is looking at. Image recognition services on the other hand use purely the device's camera with image processing calculations to determine what it is looking at. Simple implementations for domain specific applications can see a set of certain patterns be hard coded and then easily recognised when aligned to the camera correctly. QR code readers are a common example of this type of image recognition, with facial recognition being a more elaborate and unconstrained domain.

The second component of augmented reality is a digital data layer which is superimposed over the top of the user's perspective. In either augmented reality case, once the target of the application has been recognised, the augmented data layer is then populated with predefined information over the top of the camera's preview display.

During the early stages of this project, the decision to implement ASIST as a location based augmented reality application was made as to not restrict the type of asset in which ASIST is to assess. Following are the three main levels of current location based augmented reality applications researched and how they can be applied to ASIST.

2.2 Current AR Solutions

A. Augmented Reality Fundamentals

Each of the three levels utilize the device functions of vision, location, and movement, alongside a data layer to convey information to the user. Even if the data displayed is not in a form as ASIST requires, the fundamental functionality behind the application can be used as a guide to help direct initial development. Applications under this category include Junaio and Wikitude Places which mark points of interest such as landmarks or restaurants on the data layer to help users navigate [28]. Since this type of application contains the majority of functionality that ASIST requires, the technology used should be able to be applied to ASIST as well. Both Junaio and Wikitude Places are built using a web-based implementation. Junaio uses AREL—Augmented Reality Experience Language, which is a combination of HTML, CSS, JavaScript and PHP. Wikitude Places uses the Wikitude SDK, which again uses HTML5, CSS, and JavaScript.

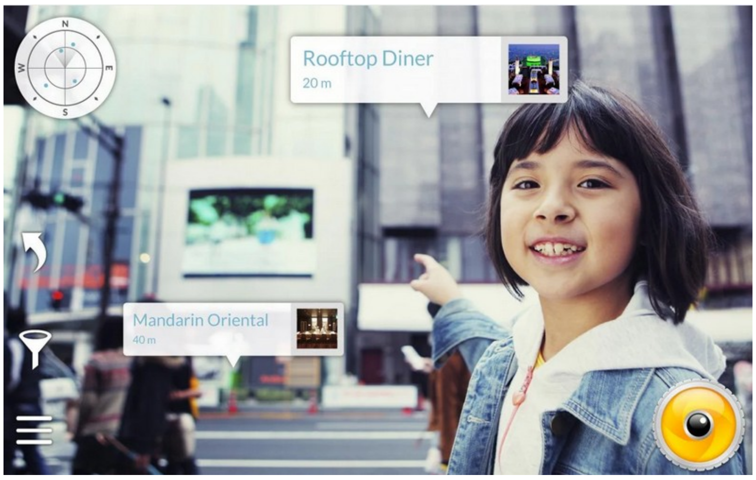


Fig 10. Wikitude Places Augmented Reality Application

B. 3D Augmented Reality Rendering:

Continuing on from the augmented reality fundamentals, applications that render both realistic and abstract 3D descriptions of assets are also widely available. Satellite AR and ESET Augmented Reality are examples of this and extend from a basic augmented reality application by applying interactive object specific graphics [29] [30]. Understanding how to use the data layers effectively as is done in these applications will be very important in contributing to the usefulness of ASIST. These techniques will be looked at further into the development lifecycle when the time comes to finalising data representation.

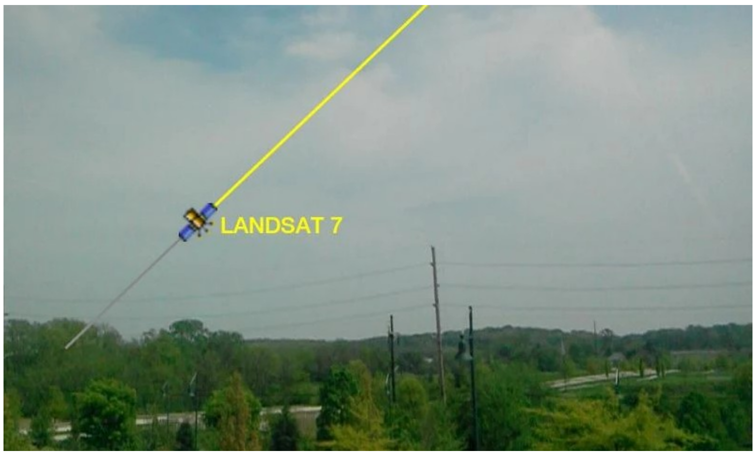


Fig 11. Satellite Augmented Reality Application

C. Geographic Information System (GIS)

As part of the research process, a three hour seminar by local firm Augview discussing their own augmented reality products was attended. Their main product is an augmented reality powered geographic information system, and is by far the most comparable application to ASIST discovered when researching the current market. A GIS is a "system for capturing, storing, checking, and displaying data related to positions on Earth's surface" and hence relates closely to what ASIST is trying to achieve [31]. Winner of various 2014 NZ spatial excellence awards, Augview "allows users to visualize underground objects that they wouldn't usually see" [32]. After talking to Augview business development manager Melanie Langlotz, she said that the application was developed natively for both Android and iOS devices.

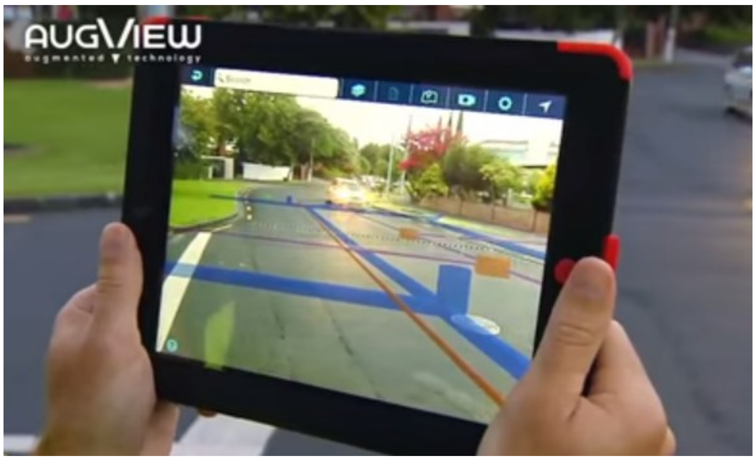


Fig 12. Augview GIS Augmented Reality Application

Due to the fact that the current applications researched had been developed using various techniques, the conclusion from these findings was to continue research. Knowing that each of the aforementioned platforms are capable of housing an augmented reality application, the decision on which platform(s) to pursue will depend on;

1) Findings with respect to each of the different platforms

2) Hardware limitations of the devices that are compatible with the different platforms

3) Development capability of the student and others involved with development with respect to the different platforms

Hence the following sections will discuss these areas in the order in which they were completed.

2.3 Web Application Research

Due to a web-based application being the least limiting in terms of platform from the solutions suggested, much of the initial pre-development time was spent researching and testing the capabilities of this approach with respect to ASIST. Other than cross platform functionality, one of the appealing factors of creating a mobile web application is that alongside a UI framework, the only core tools required are simply HTML5, JavaScript, and CSS. This allows anyone with experience in web design, one of the more common forms of software development, the ability to create a mobile application.

Due to ASIST's requirements of having access to the device's camera, accelerometer, and location, the first task was ensuring that these functions were accessible in JavaScript. This quickly lead to the adoption of Apache's Cordova:

> "Apache Cordova is a set of device APIs that allow a mobile app developer to access native device function such as the camera or accelerometer from JavaScript. Combined with a UI framework such as jQuery Mobile or Dojo Mobile or Sencha Touch, this allows a smart phone app to be developed with just HTML, CSS, and JavaScript." [33]

Once it was apparent that a web based implementation had access to the essential features required by ASIST, various tools and methods such as IDEs, SDKs and APIs for augmented reality web development were researched and tested.

A. Integrated Development Environment (IDE)

Due to being relatively new to web development, the first tool looked into was the integrated development environment. An IDE combines all the tools required to develop for a chosen platform to help automate or simplify common tasks such as compiling and testing applications while eliminating the need to use the command line or other external applications. Therefore finding the right IDE would fast track adoption of the HTML5/JavaScript platform and reduce the number of teething issues faced when starting development with this new platform.

After researching and testing different environments, the most appropriate choice to continue with was Intel's Cross Development Kit (XDK) [34]. As a free download from a reputable name, along with sufficient development guides and samples, it was an easy decision to make and using the XDK sufficiently reduced the time taken to adopt the platform.

Once familiar with the system, the sample augmented reality application from Intel was tested:

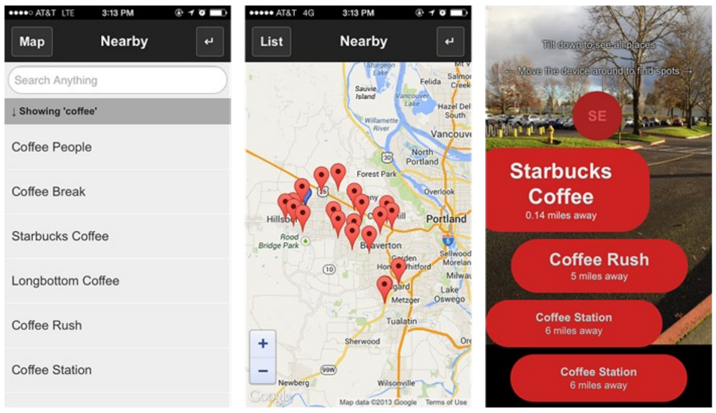


Fig 13. Location Based Augmented Reality Sample Application

Much like the augmented reality fundamentals applications discussed previously, this application appeared to be a great starting block to build ASIST from, with only the visual data layers needing modification to display data correctly for ASIST. Unfortunately, as many other users had mentioned, the application failed to perform correctly, and even after hours of modifications and additions, access to the camera was unavailable and hence the decision was made to move on to further research in the form of third party software development kits:

Software Development Kits (SDKs)

As with most domains of software development, a code base of some description has already been written to simplify development, and despite augmented reality's reasonably recent arrival, a hand full of SDKs and APIs are available for use. Three viable SDK options were analysed; Wikitude, Layar, and Metaio.

1. Wikitude SDK

Carrying on from the aforementioned Wikitude augmented reality application, Wikitude also offers their SDK to developers to create their own applications. Wikitude's all-in-one AR solution includes image recognition & tracking, 3D model rendering, video overlay and location based AR - everything that ASIST needs, and more.

An example of an ASIST-like application created using the Wikitude SDK is the Hermes Virtual Tour:

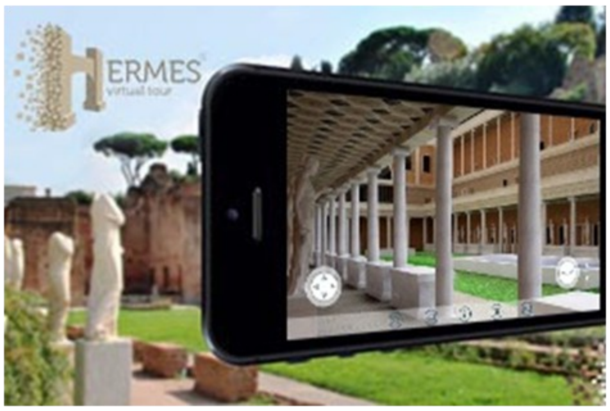


Fig 14. Hermes Virtual Tour Augmented Reality

The Hermes Virtual Tour recognizes via location and image recognition missing monuments, and renders them on the augmented data layer.

The SDK is available in a variety of options, ranging from a free, watermarked trial, to an all inclusive ~$6600 per year license. If it is decided to complete the development of ASIST with a web based implementation, then the free version of Wikitude will be tested, and funding discussed with Opus if it appears to be a viable solution.

2. Layar SDK

As with the Wikitude SDK, the Layar SDK also implements both vision based and location based augmented reality. As well as JavaScript Cordova libraries, Layar also offers plug-ins for both native Android and iOS development, offering flexibility of platform if Layar is chosen to aid development. Unfortunately the Layar SDK comes with only a 30 day free trial before a purchase is required, so the decision was made against investigating this tool further.

3. Metaio SDK

Metaio is the last web based add-on researched [35]. It shares a lot of similarities with Wikitude, including a similar price model of a free unlimited trial followed by various pricing for different features. Metaio then extends past the requirements of ASIST with a whole host of features including Unity support and facial tracking.

In conclusion, if a web-app were pursued, any of the above SDKs would be applicable to the ASIST project. However, all of the SDKs researched go well beyond what is required by ASIST at this stage, such as image recognition and wearable technology integration. Provided that the student is confident in using the fundamental Cordova device plug-ins, an SDK may not be needed. If the decision to use an SDK is made however, that decision would be based on ease of use, documentation, and price. From the findings thus far, Wikitude is the front runner due to granting access to an unlimited free (watermarked) license, alongside very recent documentation discussing the steps required to integrate and use the SDK with Intel's XDK and other IDEs.

2.4 Hardware Requirements

Once research on the web platform had been completed, time was spent researching hardware that satisfies the needs of ASIST. The following has been adapted from the mid year project plan completed to present to Opus.

A. GPS

When choosing a mobile device to use with the ASIST application, the most important feature to consider is the accuracy of the device's GPS. If the GPS is inaccurate, then ASIST's usability will be severely limited. There are a few different GPS technologies that can be used, each with varying degrees of accuracy.

The standard GPS in older mobile devices uses triangulation (or multilateration) between nearby cell-phone towers, input from in-range Wi-Fi, and tracking from your last known position - without the need of an aerial or view of the sky [36].

'Real' i.e. satellite GPS systems can then be combined with this type of GPS to be known as Assisted GPS/A-GPS/AGPS to enhance results further. It appears most modern mobile devices have some form of assisted GPS technology, at least. The 'real' GPS system used is run by the U.S. and provides services worldwide with 32 satellites [37].

To aid this service, other satellite systems can be used in conjunction to provide more accurate location positioning. These systems include:

- BeiDou and Compass - China
- Galileo - Europe
- IRNSS - India
- QZSS - Japan
- WAAS - North America
- GLONASS - Russia

B. GLONASS

GLONASS is Russia's equivalent to the above American satellite GPS system, is implemented worldwide with 24 satellites, and is integrated in a number of common phones. By supporting both US and Russian systems in a receiver, the number of available global satellites increases. Therefore, more geographic locations are able to receive four or more signals from satellites, which in turn means more successful position calculations and also better accuracy of the calculated positions in challenging environments.

Field tests in downtown San Francisco found that the positioning accuracy increased by as much as 50% when adding GLONASS. The tests were executed with the help of Qualcomm using two Sony Ericsson Live with Walkman™ smart phones, and 600 measurements were recorded and analyzed per device [38]:

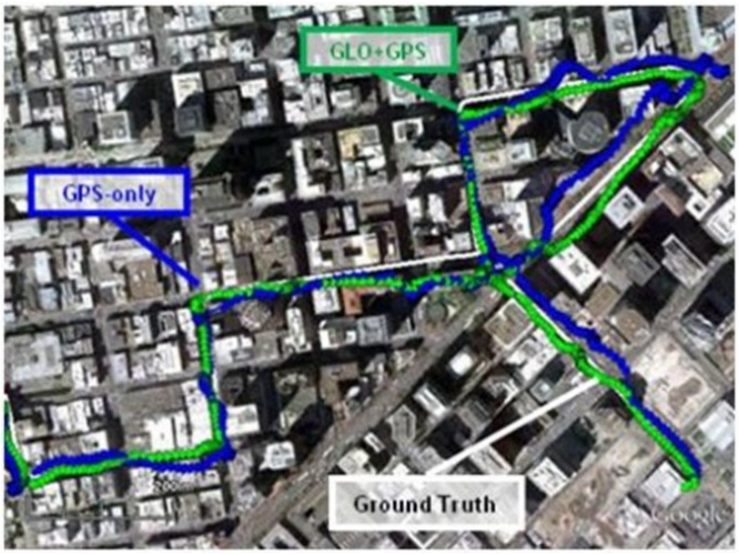


Fig 15. GPS and GLONASS Accuracy Comparison

Configuration	Number of satellites used*	CEP 68% (m)**
GPS only	5, 5	30, 4
GPS + GLONASS	10, 9	17, 7

* The number of satellites on average that were used to calculate the position. More satellites usually mean a more reliable and accurate location.
**CEP 68% means that 68% of the 600 measurements are within this distance, in meters, from the reference location. Hence lower values mean better accuracy.

Table 2. GPS and GLONASS Accuracy Comparison

Personal testing with a Sony Xperia ZR smart phone which uses GLONASS achieved accuracy of three metres when running a Chartcross Ltd GPS test.

From these findings, it will be in Opus's best interests to use such a device in conjunction with ASIST. Fortunately, Qualcomm Snapdragon processors and chipsets, which are commonly found in the vast majority of Android consumer mobile devices post October 2012 appear to have GLONASS and BeiDou built in via iZat [39].

The Qualcomm website shows 21 Android, 1 Windows, and 3 Amazon Fire devices using this chip, so the advice from this information would be to develop for Android tablets. Other chipsets offering GLONASS are available, but again, the majority of these are powered by Android.

A breakdown on Apple tablets:

- Wi-Fi only iPads do not have a GPS [40].
- 3G/4G cellular models do, but exactly what technology they use differs.
- GLONASS support has been integrated since the 3rd generation iPad (March 2012 release) [41].

Due to some remote locations not having network access to calculate position information from, Wi-Fi only tablets without a 'real' GPS should not be considered.

C. Minimum Hardware Requirements

Required hardware specifications as given in original Opus document:

- Screen size - minimum of 7 inches
- Camera - minimum of 2 megapixels
- Storage - minimum of 16GB

PriceSpy lists 76 tablets with GLONASS support that satisfy the above attributes.

D. Requirements Expanded

Screen Size - The larger the screen size, the better ASIST will be able to convey information, meaning that tablets with a screen size of 10 inches and above should also be considered. For roads and un-detailed assets, 7 inches is sufficient, but may become cluttered with more complicated assets.

Camera - At this stage, since the augmented reality is planned to only use positioning and accelerometer data, image recognition isn't needed, thus camera quality isn't a concern. As long as the terrain is distinguishable on screen, the camera will be sufficient. However, if accuracy needs to be increased due to leveraging image recognition, then a higher quality camera may be required.

Storage - The application size will be minimal, with cached map data requiring the majority of the storage. Depending on how much space the operating system uses, 16GB should be sufficient for ASIST and any other Opus and work related software.

Memory and Processing - As with storage, given that the tablet is adequate in the above categories and has a recent chipset that includes GLONASS capability, then it is expected that the CPU and memory will also be sufficient.

Network - Strong preference is given to devices with 3G/4G capability for live data communication, rather than relying on an additional device to be used as hotspot data source.

E. Conclusions

The original Opus report suggested a Windows device as to integrate with the Opus network easier. However at this stage there is no Windows product that supports all project requirements. Additionally, the Opus network integration is most likely manageable on any operating system and so this shouldn't be a reason to choose Windows over the other options.

This has left the decision of tablet choice to be produced from either: Samsung (Android), Sony (Android), Google (Android), and Apple (iOS). Price wise, the Android solutions are generally more affordable.

If building for a single operating system while wanting to be compatible with as many products and brands as possible, Android development is the best choice from a hardware perspective. Next will be a review of Android's capabilities regarding housing and developing this type of application compared to HTML5 and JavaScript. Web based examples of working augmented reality applications have been researched, but the difficulty of developing these in comparison to Android development has yet to be evaluated. If neither Android or HTML5 appear to be the correct solution, then Apple's Swift will be assessed as a possible development language.

F. Specific Tablet Recommendation

Tablets that fulfil all requirements start at ~$500. The lower end of the Samsung Galaxy Tab range is what to expect for this price. Screen sizes in this price bracket range from 7 to 10 inches. Storage of 16GB, but often with expandable storage up to 64GB. A low resolution camera (2-5 MP), and CPUs between 1-2GHz. Above $500, sees the four brands mentioned above represented by their various tablet models with various specifications. For testing purposes, and most likely deployment, the lower priced models are adequate as the main features that ASIST relies on such as the GPS are very similar or identical across tablets over all price ranges.

The most affordable, suitable model is the Samsung Galaxy Tab 4 8" 16GB 4G, $448:

- 1.2GHz Quad Core Processor
- 1.5GB RAM
- 16GB Hard Drive
- 3MP Rear Camera and 1.3MP Front Camera
- Android KitKat
- SD Card slot for up to 64GB
- 8 Inch Screen
- 4G Cellular Network Connection

G. Alternative Hardware Solution

Another option that would not restrict development language and possibly achieve more accurate GPS results is the use of an external GPS.

If accuracy with the above technology is found to be insufficient, then external GPS products that communicate with mobile devices are also an option that should be considered due to their ability to increase accuracy and being able to be positioned away from other utensils capable of degrading the device's internal GPS such as magnetic cases. Products are generally in the $100-$300 range.

Examples include:

- Garmin GLO—combines GPS and GLONASS and connects to device via Bluetooth.
- GNS 2000.
- SkyPro XGPS160.

At first ASIST will be tested with an un-aided tablet, and only if the accuracy of results is insufficient, then should an external device like this be trialled.

Due to this research giving preference towards an Android platform, the next step was to research and test the augmented reality capabilities of native Android development:

2.5 Native Android Development

Again, as with mobile web development, one of the first steps in the development/testing process is to find, if possible, a satisfactory IDE. In previous years, the tool of choice has been Eclipse with an Android plug-in. However since December 2014, Google's Android Studio has superseded Eclipse and offers a dedicated and elaborate development environment for mobile Android applications [42].

Initial research was again spent looking for available SDKs to simplify the development process. Augmented reality kits such as Qualcomm's Vuforia were found, but with Android Studio only being recently released, documentation to utilise the SDKs were insufficient, so research quickly turned into hands on development and testing with the standard Android APIs.

First an understanding of the overall framework structure and interaction between different files involved in an Android application were required. There are four main application components that contribute to an Android application, each of which serves as a point for the application to be accessed. They are; Activities, Services, Content Providers, and Broadcast Receivers [43]:

A. Activities

Activities are generally the main component used when developing applications. A single activity represents one screen of the application and any interactions available on that screen. They are independent of one another and are then linked to form a multi-screen application. ASIST will primarily use activities; one for the main live view function, and one for each asset management task.

B. Services

The opposite to an activity is a service. Services are user interface-less processes which run in the background and do not effect the user's current interaction with the application he/she is using, whether it be the application that this service belongs to, or another application. ASIST may use services to periodically update cached asset and structure data when road works or construction has been conducted.

C. Content Providers

A content provider is the application's file system, which allows access from other authorised applications. In ASIST's case, using a content provider will allow other Opus software to communicate with ASIST's data source.

D. Broadcast Receivers

Broadcast receivers take action based on events from; the system, the current application, or other applications. An example of ASIST utilizing this functionality may include initiating a service such as inventory validation once the GPS of the system has achieved an accuracy level above a certain threshold.

Once familiar with the composition of a native Android application, the remaining development time was spent developing a prototype for one of the core requirements of ASIST, the camera. When first deciding on implementation, the choice between two camera APIs had to be made. With the recent release of Android 5.0 Lollipop, a new camera API, camera2, has been released, deprecating the original camera API, meaning that no future capabilities will be added. Unfortunately due to the main tablet being used in the Opus work force not having an operating system update scheduled in the foreseeable future, the decision was made to use the older API. This is only a minor issue however and is not going to be a problem for at least a few years, which then the code can be adapted for adoption by the newer platforms. Additionally by using the older API, many advantages exist, including the availability of far more documentation and examples.

The camera application was made from two java classes; one extending Activity, and the other extending the SurfaceView class, along with a selection of xml files to initialise the interface:

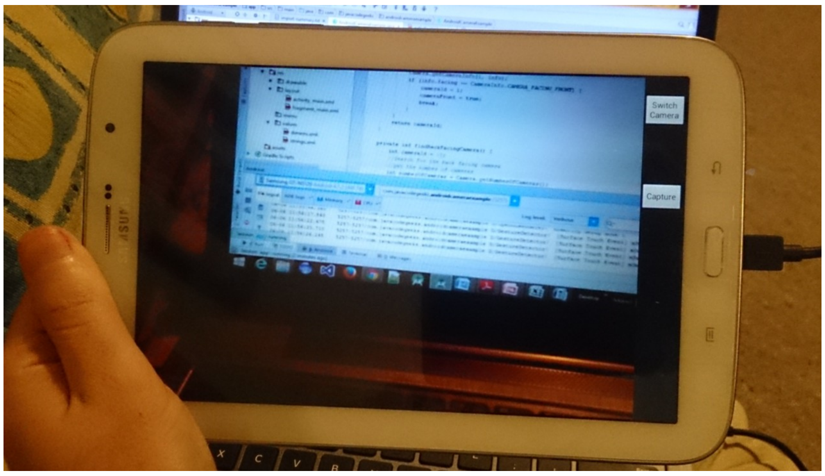


Fig 16. Self Developed Android Camera Application

This solution was achieved easier than the web implementation, as was expected due to greater experience with Java development, as is discussed next.

2.6 Developer Capabilities

Throughout university, a variety of programming languages have been learned. Of these, the majority of programming work completed to date, and hence strongest and most understood language is Java. Due to native Android development being primarily Java based, this skill set lends itself to this development path. A considerable amount of time has also been spent investing into HMTL5 and JavaScript over the past six months as well to give the multiplatform approach a fair trial. This includes website development study as well as opting to complete a recent mobile application assignment with web technologies, relating directly to the ASIST project due to it's use of location and spatial data. Knowledge of SQL and database principles also exist if required to manipulate the data involved in this project.

2.7 Recommendations Moving Forward

Due to the information learned over the previous sections, decision to pursue development of a native Android application has been made. The advantage of cross platform compatibility that a web based solution would bring is enticing, but is ultimately outweighed by the increased development efficiencies that will be observed moving forward with the project due to the student's personal experience and capabilities. In addition to this, the vast majority of tablets already deployed throughout Opus are Samsung Android devices, so a cross platform solution isn't strongly required. Discussions with Kodie and other members of the development team support this decision, with another member of the team also being confident with Java development.

The first task of development will be testing the accuracy of a purely location based implementation using the GPS, compass and accelerometer. To save time this will be conducted with self generated GPS coordinates, rather than using real data provided by Opus due to a transformation of the data into GPS coordinates being required before being usable.

When accuracy is determined to be acceptable, development of data representation will begin. Current ideas as illustrated in images throughout this document suggest some basic representations to use for road surfaces. These will be built on and expanded for a wider range of assets, and to allow for more detail to be disseminated to the user. These descriptions will have to be responsive and be determined by the user's perspective, most likely requiring OpenGL rendering, or another alternative, which will be investigated when required.

After these live view components are complete, expansion and iteration of the application can be conducted. This will include working through the use cases in order of priority decided by Opus, and ongoing performance evaluations with involved parties to ensure correctness and usability of the application. It is likely that not all functions outlined in the initial documentation will be achievable in the given time frame, so decisions and trade-offs between quantity and quality of functions will need to be made throughout development.

2.8. Risk Analysis

Due to the majority of this project being undiscovered territory for those involved, there are numerous areas for this project to fall over. Here we take note of some of these, and suggest solutions or alternative methods if forced to change our intended approach.

Low accuracy

One of the first hurdles to overcome in development is the accuracy of the GPS and other positioning hardware of the device. To simplify development and allow for maximal scaling of the application to different assets, efforts will first be put towards a purely location based solution. There is a real chance however that this will not be sufficient. Two alternatives have been theorised:

The first is the addition of manual calibration. Given a road for example, where an offset in accuracy of a few metres can be the difference between lanes, the user can manually input the centre line of the road by drawing on the touch screen. ASIST will then shift it's projection of the asset to align with this input.

The second alternative relates to a change in data representation. If the device is unable to project the description over the asset accurately, then data should be displayed in a way that doesn't rely on absolute accuracy. This may include a rendering of the asset that is partially independent of the location information. Given that the user is in a specific location and the asset to be analysed is selected, then that object can be rendered on the screen while not specifically looking at the asset. This is a less optimal solution, however given that the user knows what asset they've selected, this approach will still convey the same information.

Asset Data

As illustrated previously, the current structure of asset data isn't directly usable as we require GPS coordinates to plot information. It is expected that data will be received in a usable form at a later date for testing. However this isn't confirmed, and the task of this data conversion may also be adopted, requiring additional time.

Platform Incompatibility

Despite research thus far suggesting a Java based Android implementation, there are still unknowns to be discovered throughout development. During the process of development issues creating the requested functionality are likely to arise. The absolute worst case will be discovering that Android is no longer a viable development path, although this is unlikely. This may then require translating the project into another language. Fortunately with access to Java skilled mentors, this is unlikely to occur.

2.9 Source Control & Android API Version Comparison & Selection

After arriving at the conclusion that native Android development is indeed the most appropriate path to follow with respect to both the application requirements, and the experience available of those involved, the next and main endeavour of this project to undertake is the development of ASIST. Before planning of development can begin, the first step is to organise online source control to provide both insurance of completed work in the event of incorrect development decisions, as well as to keep those requesting information regarding project progress up to date by granting access to this repository. As suggested by Opus due to the availability of free private repositories, Bitbucket was chosen to be used for this task. Once created on the website, a Bitbucket plug-in for Android Studio was used to commit, push, and pull changes to this repository seamlessly throughout development.

Another predevelopment task was assessing which Android operating system versions were capable of the features required to house this application, and appropriate for Opus with regard to their current fleet of tablets distributed throughout the company. The following table outlines the key developer features added to the most recent Android operating system versions with ASIST specific features in bold, along with the Android market share break down of what operating systems are being used throughout all Android devices (development features made available on earlier versions are available on future systems) [44]:

Version	Key Developer Features Added	Release Date	Android Market Share
Android 6	Custom Chrome Tabs for better in app browser support App Permissions management update	2015 October 5	
Android 5.1	No key developer features added	2015 March 9	5.1%
Android 5.0	Several new API Tracking battery consumption app	2014 October 17	15.9%
Android 4.4	Public API for SMS management. Improved memory usage **Security enhancements** NFC Host Card Emulation Printing Framework **Storage Access Framework** **Hardware Sensor Batching** **Full-screen immersive mode** GLES2.0 SurfaceFlinger Chromium WebView Audio tunneling to DSP Audio monitoring Wi-Fi certified Miracast New Bluetooth profile IR Blasters API Wi-Fi Tunneled Direct Link Setup (TDLS) support Tools for analyzing memory use **Screen Recording**	2013 October 31	39.2%

Table 3. Android API Version Changes

Version	Key Developer Features Added	Release Date	Android Market Share
Android 4.3	**OpenGL for Embedded Systems 3.0 graphics support** Logging and analyzing enhancements Wi-Fi scanning API Improved DRM (digital rights management) API VP8 encoding	2013 July 24	4.5%
Android 4.2	Secure USB debugging Vsync timing Triple buffering reduced touch latency CPU input boost Native RTL support External display support - Display Manager Nested fragments Renderscript Compute - run tasks on the GPU Renderscript ScriptGroups, built-in intrinsics	2012 November 13	15.2%
Android 4.1	App stack navigation for deep navigation Camera sound uodates NFC supports large payloads over bluetooth WIFI/WIFI-Direct service discovery Large, detailed, multi-action notifications Input manager allows you to query input devices	2012 July 9	12.1%
Android 4.0	Low-level streaming multimedia Grid Layout Spell checking service Address Space Layout Randomization VPN client API Remote Device camera enable/disable ZSL exposure, continuous focus, and image zoom Flags to help control system UI	2011 October 18	3.7%

Table 4. Android API Version Changes

Security enhancements

Once working with real data, security around storing, modifying, and transferring the vast and valuable information that Opus and ASIST works with is of upmost importance, so the availability to include additional security measures makes Android 4.4 appealing to use. Of the security enhancements added in Android 4.4, the cryptographic algorithm improvements are the most applicable to ASIST. Support for the Elliptic Curve Digital Signature Algorithm has been added, allowing for increased security of digital signing. This can be applied to the signing of a data connection between an onsite tablet and a central data source to ensure authenticity of the mobile device requesting information.

Storage Access Framework

The new storage access framework simplifies the process of integrating multiple storage services into an application. Initial testing with ASIST will only use a small set of self generated local data. However when this needs to be combined with multiple local and/or cloud based file systems for different asset types such as roads and buildings, this should make the process of using the data by both the developer and end user a more straight forward process. The datasets to be loaded and viewed while in the field will be easily selectable by the user.

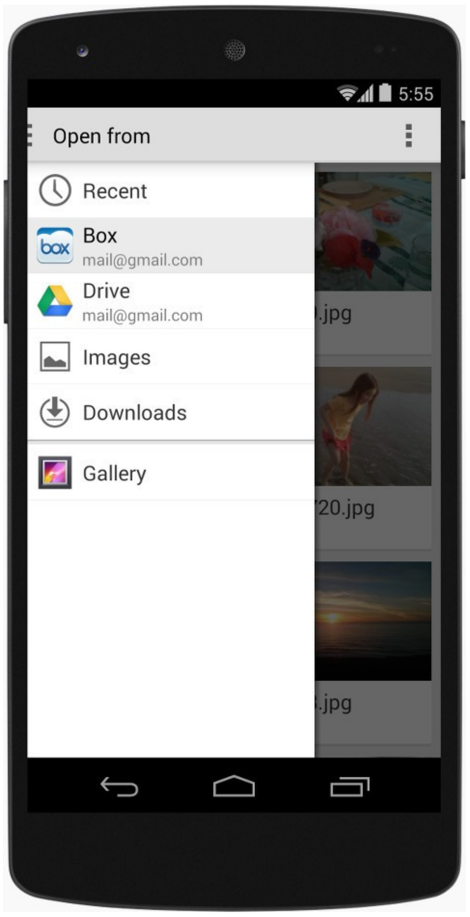


Fig 17. Storage Access Framework

Hardware Sensor Batching

The core functionality of ASIST relies on multiple ongoing sensors (location, accelerometer, compass, gyroscope), all of which consume a significant amount of power. Hardware sensor batching is an optimisation which dramatically reduces the power consumption of such processes. Sensor events are delivered efficiently in batches to allow the device's application processor to remain in a low power idle state until batches are sent. As is done already to gain access to this data, batched events can be requested from a sensor using a standard event listener, and the frequency of batches can also be controlled. Sensor events can also be tracked while the device is asleep, which will negate the satellite reconnection times that can occur when reopening ASIST after turning the device's screen off.

Full-Screen Immersive Mode

Like many camera based applications, having a full-screen in ASIST will be advantageous to allow for displaying as much information as possible. This mode removes the status and navigation bars, and allows areas where these bars would usually be to also contain touch events like that of the rest of the screen, without the status bars reappearing. Transitions between full-screen and the regular view can then be made by downward touch gestures from the top of the screen which can be used to reveal options for ASIST.

Screen Recording

Primarily for documenting progress and showcasing the application in action, the screen recording capability added in Android 4.4 allows for videos to be recorded via USB to Android Studio directly. Recording controls are accessed directly from Android Studio which overcomes the need of installing a secondary application which often requires rooting of the device for screen recording to work.

OpenGL for Embedded Systems 3.0 graphics support

Android 4.3 adds native support for OpenGL ES 3.0 which includes texture compression which will benefit ASIST if textures are used for the drawing of condition data. Additional benefits include advanced texture rendering and shading which again may benefit ASIST dependent on the comprehensiveness of the graphics used.

Tablets Available at Opus

Various tablets and devices are used throughout Opus, but the most common of these is the Samsung Galaxy Note 8.0 GT-N5120, upgradable to Android version 4.4.2.

Fig 18. Samsung Galaxy Note used at Opus

Due to all of these features as a whole, but primarily due to the tablets available at Opus and the ability to record the screen directly, Android 4.4 has been chosen as a minimum operating system requirement for ASIST. The reason to make such a decision at the beginning of development is to guide development towards using the most appropriate features available at the time, while minimising the use of deprecated classes and maximising forward compatibility. With this said, the majority of development features available across Android versions 4.0 and higher overlap, and so this decision can be changed in the future. A newer version such as Android 5.0 was dismissed due to the currently low market share usage of that operating system, and the current uncertainty around whether the tablets used at Opus will become eligible for such an operating system upgrade.

III. DEVELOMENT PROCESS

Once source control was prepared and a target operating system version selected, development could begin. Taken from the mid year presentation, the following high level schedule was attempted to be followed, with source control contributing to 'Application Setup':

Task	Iteration (Week)	Time (Hours)
Application Setup	1	20
Camera Preview	2	10
GPS Tracking	2	10
Accelerometer and Compass Tracking	3	10
Data Layer	3-4	20
Generate Data and Test	4	10
Transform Real Data to Test	5	20
Re-evaluate Performance	6	5
Compile Documentation of Process	6	10
Basic Documentation and Stakeholder Meeting	6	5
Testing by Invercargill Team	6	1
Report on Outcome	7	10

Table 5. Development Schedule

3.1 Application Setup

In addition to source control, application setup involved implementing the overall structure of the application. This included the creation of default activities for the live view, asset management, and settings windows, with a main menu and buttons linking them together.

```
public class MainMenu extends ActionBarActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main_menu);
        }

        public void liveView(View v) {
                Button button = (Button) v;
                startActivity(new Intent(getApplicationContext(), LiveView.class));
        }
…
}
```

Fig 19. Main Menu Code Snippet

The remaining time was spent experimenting with the layout options available - both the full-screen capability of Android 4.4 as previously discussed (left), and different button placements dependent on the orientation of the device (right, bottom):

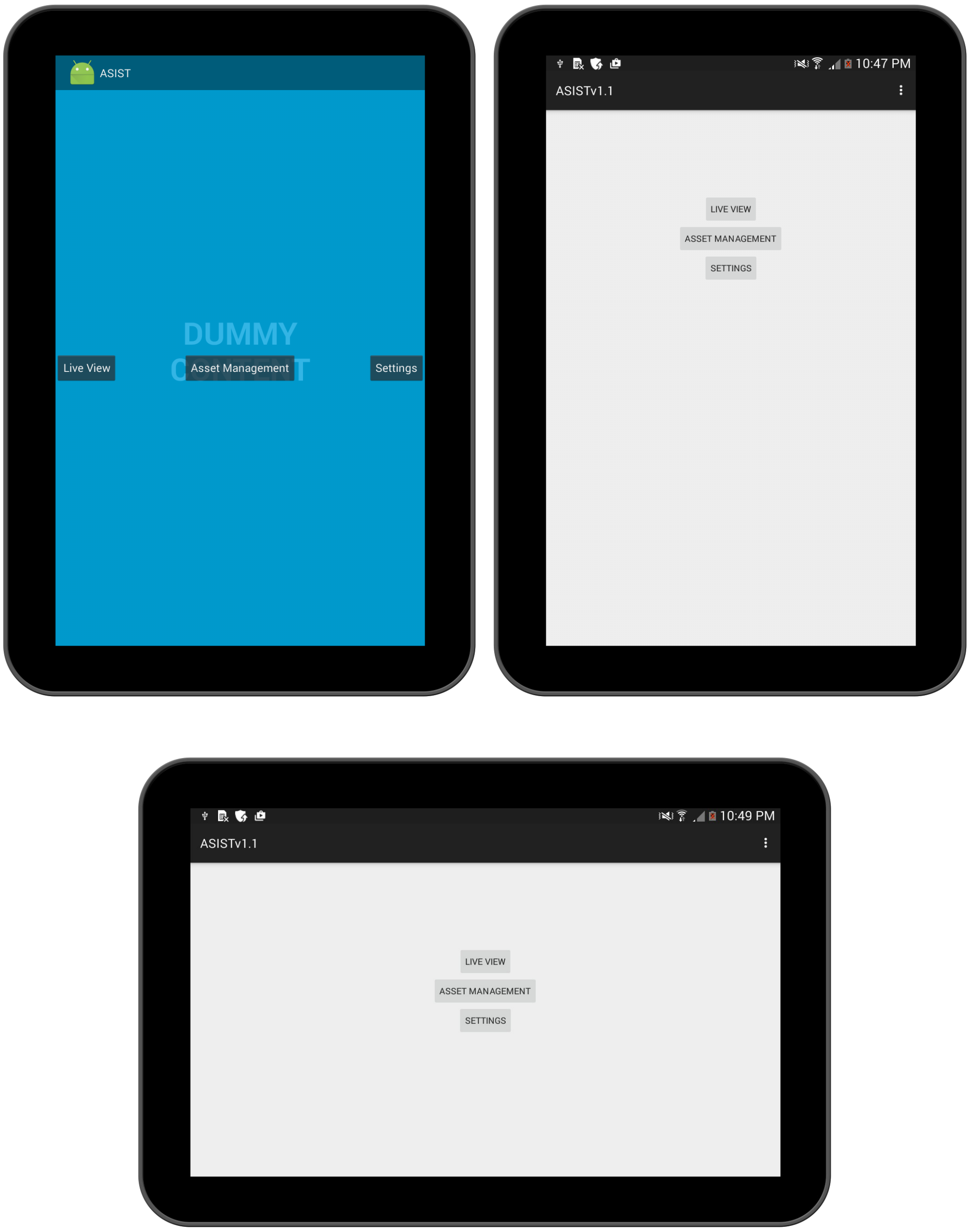


Fig 20. Main Menu Layouts

3.2 Camera Preview

The next three points (camera preview, GPS tracking, accelerometer and compass tracking) are the foundational building blocks in which any location based augmented reality application is built, and thus required preparing before development could continue. One of the only negatives that come from developing for Android 4.4 rather than Android 5.0 is related to the camera. To be compatible with earlier versions, the Android camera class was used. Unfortunately this has been deprecated and replaced by the camera2 interface from Android 5.0 onwards, meaning that development of the camera class has been discontinued, and usage of the class will slowly be phased out. Camera2 has been introduced to add additional features and simplify the process of setting up the older camera as will soon be discussed. However due to it's maturity, the positive of using the now deprecated camera class comes from the vast number of online resources available to aid development compared to that of the newer camera2 class.

Firstly, as with any hardware feature, permissions and uses-feature tags must be supplied to the manifest file to gain access to the hardware:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Fig 21. Hardware Permissions

A basic camera application can be broken down into two main components; the camera, and the 'surface' in which that camera's contents will be displayed:

```
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
        private SurfaceHolder mHolder;
        private Camera mCamera;
        …
}
```

Fig 22. Camera Preview Class Definition

Because ASIST at this stage only needs to provide a preview of the camera perspective, rather than taking pictures and recording video, development can be simplified by reducing the number of camera related methods required.

```
public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;
        mHolder = getHolder();
        mHolder.addCallback(this);
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
}
```

Fig 23. Camera Preview Constructor Method

This is then accompanied by creation of the aforementioned surface which involves linking the camera object to a surface for display of this preview, and implementing the following required methods of this surface interface:

```
public void surfaceCreated(SurfaceHolder holder) {
        try {
                // create the surface and start camera preview
                if (mCamera == null) {
                        mCamera.setPreviewDisplay(holder);
                        mCamera.startPreview();
                }
        } catch (IOException e) {
                Log.d(VIEW_LOG_TAG, "Error setting camera preview: " + e.getMessage());
        }
}


public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
        refreshCamera(mCamera);
}


public void surfaceDestroyed(SurfaceHolder holder) {
        mCamera.stopPreview();
        mCamera.release();
}
```

Fig 24. surfaceCreated, surfaceChanged, surfaceDestroyed methods

These were the initial methods used to create a working camera. However as can been seen in the surface changed method, no information from the input parameters is being used to change the surface. The next step was to add the capability of camera rotation based on the screen's orientation, which is done in the surface changed method. This was the first main road block in development, requiring different rotation strategies to be trialled before finding one which worked correctly:

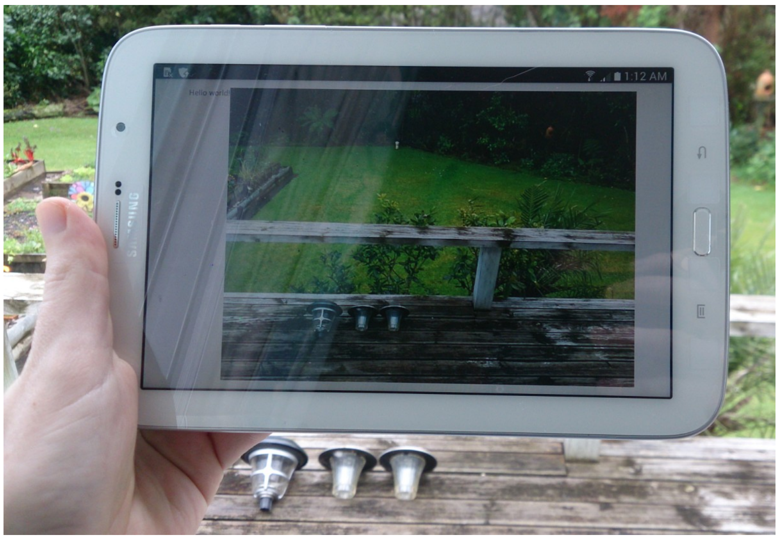


Fig 25. Camera working correctly while landscape

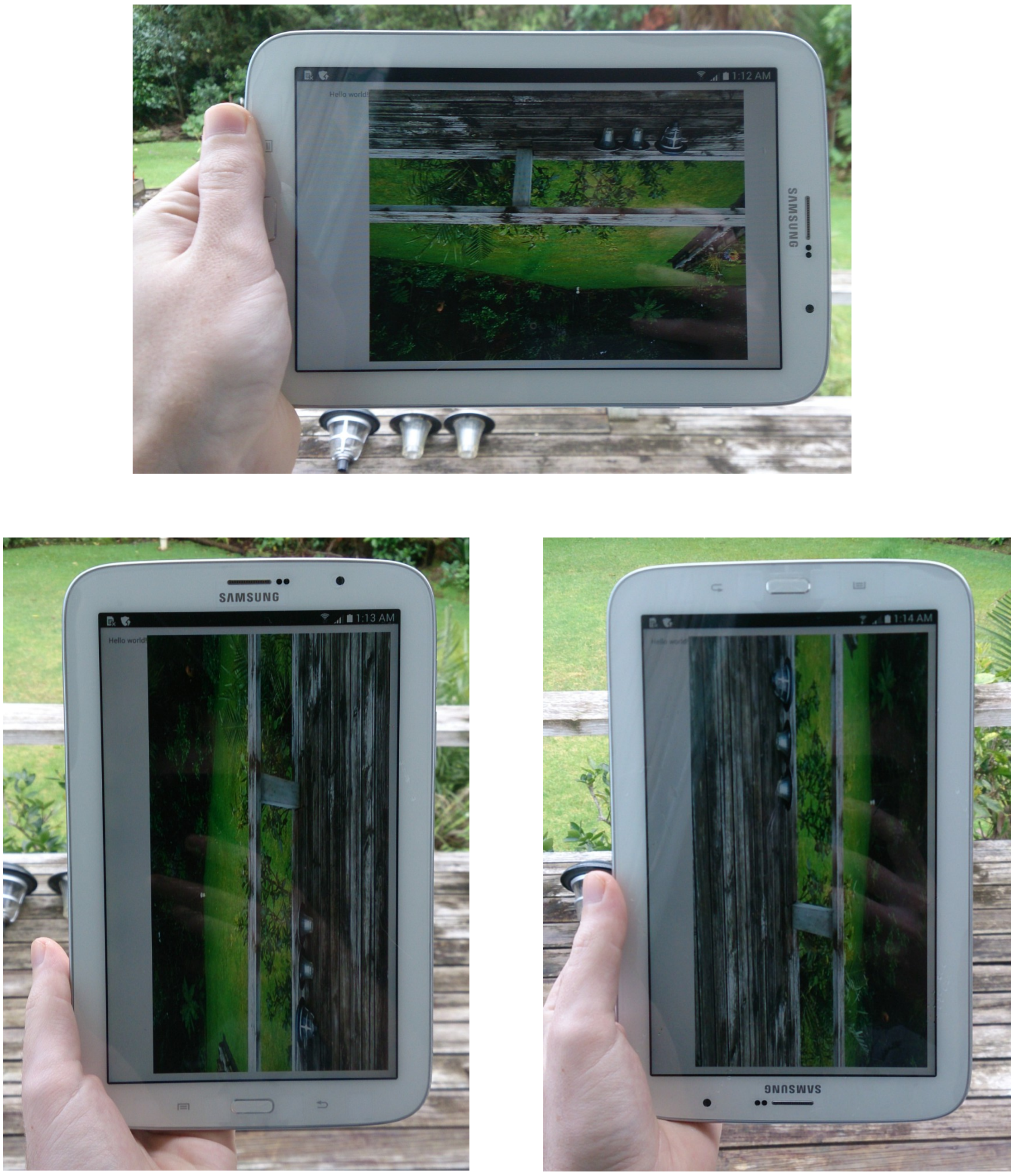


Fig 26. Camera working incorrectly when rotated

```
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        mCamera.stopPreview();
        LiveView.setCameraDisplayOrientation(LiveView.class ,1, mCamera);
        Camera.Parameters parameters = mCamera.getParameters();
        Display display = ((WindowManager)getContext().getSystemService
                (Context.WINDOW_SERVICE)).getDefaultDisplay();

        if(display.getRotation() == Surface.ROTATION_0){
                parameters.setPreviewSize(h, w);
                mCamera.setDisplayOrientation(90);
        }
        if(display.getRotation() == Surface.ROTATION_90){
                parameters.setPreviewSize(w, h);
        }
        if(display.getRotation() == Surface.ROTATION_180){
                parameters.setPreviewSize(h, w);
        }
        if(display.getRotation() == Surface.ROTATION_270){
                parameters.setPreviewSize(w, h);
                mCamera.setDisplayOrientation(180);
        }
        mCamera.setParameters(parameters);
        refreshCamera(mCamera);
        mCamera.startPreview();
}
```

Fig 27. surfaceChanged method to rotate camera display correctly

The only problem with the above technique was while upside down in landscape mode, the image continued to rotate incorrectly (upside down). However only a single landscape and portrait mode were required, so this problem was ignored and left to be fixed at a later date if the time was available to allow for more important tasks to be completed. Incorrect margins around the camera view were then remedied before moving on to the next component of the application.

3.3 GPS Tracking

To display information relative to the user's location, tracking of the device's GPS is required. There are two main ways of achieving this, either through Android's built in location API, or through Google Play's services API, each with their own positives and negatives. It is recommended by Android to use the Google Play services location API over the alternative, but with little explanation for this suggestion. If pursuing the Google Play services path, an extended setup of the service must be undergone to leverage the API, compared to simply importing the location package if using Android's built in methods. The main difference with respect to ASIST between these two methods however is the fact that Google Play services requires a network connection to make location requests, while the built in package does not. As previously mentioned, since this application is to be used out in the field, there is the requirement for ASIST to be able to work without a network connection while in sheltered, distant areas without a reliable, or any available wireless network. For primarily this reason, the decision to use Android's location package was chosen.

The minimum location information required for this application to function is the user's latitude and longitude, with other details such as altitude and movement speed being required depending on implementation. The first step to gaining this information is to initialise the location manager:

```
private LocationManager locationManager = null;
...
//Location:
locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.NO_REQUIREMENT);
locationManager.requestLocationUpdates(locationManager.getBestProvider(criteria,
        true), 0, 0, this);
```

Fig 28. Location manager initialisation

The criteria settings specify fine accuracy and no power requirement rather than coarse accuracy and a higher power requirement to ensure that the information received from the GPS is as accurate as possible. Location updates are then requested with this criteria, with the two zeroes referring to a minimum time passed and distance moved of zero to trigger near instantaneous updates of location to be used by the application. This coupled with the required 'onLocationChanged()' method allows the latitude and longitude to be accessed and used as follows. Due to updates occurring near instantaneously, 'lastLocation' is equivalent to the user's current location, therefore negating any observable latency between the device's location readings and real time coordinates.

```
@Override
public void onLocationChanged(Location location) {
        lastLocation = location;
}

lastLocation.getLatitude();
lastLocation.getLongitude();
```

Fig 29. onLocationChanged() method and accessing current latitude and longitude

Of the device sensors used, location tracking is the most straight forward. However as will be discussed, problems occur when integrating this information with additional sensors and across multiple files in an application.

3.4 Accelerometer & Compass Tracking

The final sensors required for a location based augmented reality application to work involve tracking what angle the device is being held on. This information is found using the accelerometer, compass, and gyroscope from the sensor manager:

```
sensors = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
Sensor gyroSensor = sensors.getDefaultSensor(Sensor.TYPE_ORIENTATION);
Sensor accelSensor = sensors.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor compassSensor = sensors.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Fig 30. Sensor initialisation

To begin recording this information, the following initialisation was used:

```
        private void startSensors() {
                isAccelAvailable = sensors.registerListener(this, accelSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
                isCompassAvailable = sensors.registerListener(this, compassSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
                isGyroAvailable = sensors.registerListener(this, gyroSensor,
                        SensorManager.SENSOR_DELAY_NORMAL);
        }
```

Fig 31. startSensors() method

Like the location listener, sensor methods regarding ongoing information retrieval are required to be implemented for these sensors:

```
        @Override
        public void onSensorChanged(SensorEvent sensorEvent) {
                switch(sensorEvent.sensor.getType()){
                        case Sensor.TYPE_GYROSCOPE:
                        break;
                        case Sensor.TYPE_ACCELEROMETER:
                        lastAccelerometer = sensorEvent.values.clone();
                        break;
                        case Sensor.TYPE_MAGNETIC_FIELD:
                        lastCompass = sensorEvent.values.clone();
                        break;
                }
                this.invalidate();
        }
```

Fig 32. onSensorChanged() method

As can be seen above, gyroscope data has been discarded due to the fact that the tablet being tested on doesn't support this functionality, but primarily due to the fact that the combination of the accelerometer and compass provides enough data to make the gyroscope redundant for this application. As with location tracking, gaining access to this sensor information is well documented and relatively straight forward, with issues mainly arising due to incomplete interface overloading or incorrect project setup with respect to granting the application access to use these sensors. The real challenge came from leveraging all of this information simultaneously to create the on screen data layer.

3.5 Data Layer

Planned to take 20 hours to complete, development of the data layer drastically reduced the pace of development that was up until this point, ahead of schedule. After unsuccessful attempts at applying projection and model view matrices to transform the real world three dimensional space into a two dimensional coordinate system on the tablet's screen, various tutorials were followed to achieve augmentation. Before this began however, a new layer for displaying information over the camera view was required to be created. Until this point, all functionality of the above sensors had been contained within a single file. However it is this data layer that requires all of this sensor and location information, while the lower layer camera doesn't transmit or consume any of this information. For this reason, the above sensor information was moved to this new data layer:

```
public class OverlayView extends View implements SensorEventListener, LocationListener {
        private LocationManager locationManager = null;
        private SensorManager sensors = null;
        …
}
```

Fig 33. OverlayView data layer definition

The first attempt after this refactoring involved creating a bearing to other fixed locations only taking into account left-to-right rotational information. Coordinates of the north and south poles, and west and east 'poles' were recorded, and bearings created as follows:

```
protected static double bearing(double lat1, double lon1, double lat2, double lon2) {
        double longDiff = Math.toRadians(lon2 - lon1);
        double la1 = Math.toRadians(lat1);
        double la2 = Math.toRadians(lat2);
        double y = Math.sin(longDiff) * Math.cos(la2);
        double x = Math.cos(la1) * Math.sin(la2) - Math.sin(la1) * Math.cos(la2) * Math.cos
                (longDiff);

        double result = Math.toDegrees(Math.atan2(y, x));
        return (result+360.0d)%360.0d;
}

double angle = bearing(device.latitude, device.longitude, target.latitude, target.longitude);
double xPos, yPos;

if(angle < 0)
        angle = (angle+360)%360;

xPos = Math.sin(Math.toRadians(angle)) * distanceApart;
```

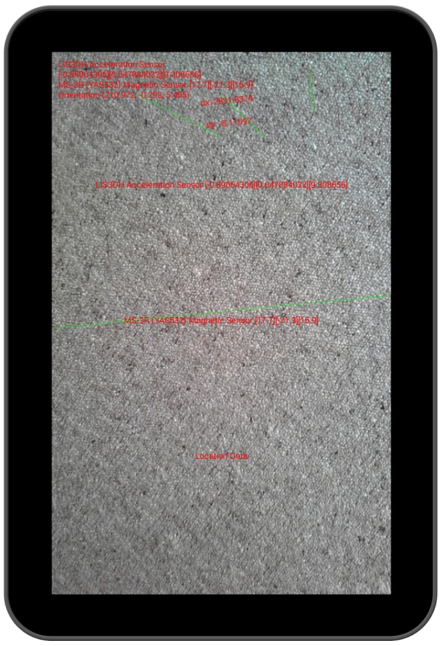
Fig 34. Bearing calculations

The next step was to then take into account the vertical rotation of the tablet as well. Various options on how to achieve this were explored. First the decision to include altitude data was considered. Like latitude and longitude data, this can be extracted easily, and the altitude of target locations can be set in a similar manner. To draw this, a more elaborate technique than used above involving rotation matrix calculations and remapping of the coordinate system were used as follows:

```
        // compute rotation matrix
        float rotation[] = new float[9];
        float identity[] = new float[9];
        if (lastAccelerometer != null && lastCompass != null) {
                boolean gotRotation = SensorManager.getRotationMatrix(rotation,
                        identity, lastAccelerometer, lastCompass);

                SensorManager.getOrientation(rotation, orientation);

                canvas.save();

                canvas.rotate((float) (0.0f - Math.toDegrees(orientation[2])));

                // Translate, but normalize for the FOV of the camera
                float dx = (float) ((canvas.getWidth() / horizontalFOV) * ((Math.toDegrees(orientation
                        [0])) - curBearingToTargetLocation));
                float dy = (float) ((canvas.getHeight() / verticalFOV) * Math.toDegrees(orientation[1]));

                 // wait to translate the dx so the horizon doesn't get pushed off
                canvas.translate(0.0f, 0.0f - dy);

                // make the line big enough to draw regardless of rotation and translation
                canvas.drawLine(0f - canvas.getHeight(), canvas.getHeight() / 2, canvas.getWidth() +
                        canvas.getHeight(), canvas.getHeight() / 2, targetPaint);

                // now translate the dx
                canvas.translate(0.0f - dx, 0.0f);

                // draw the point. It's rotated and translated this to the right spot already
                canvas.drawCircle(canvas.getWidth() / 2, canvas.getHeight() / 2, 8.0f, targetPaint);

                canvas.restore();
        }
```

Fig 35. Horizon line drawing

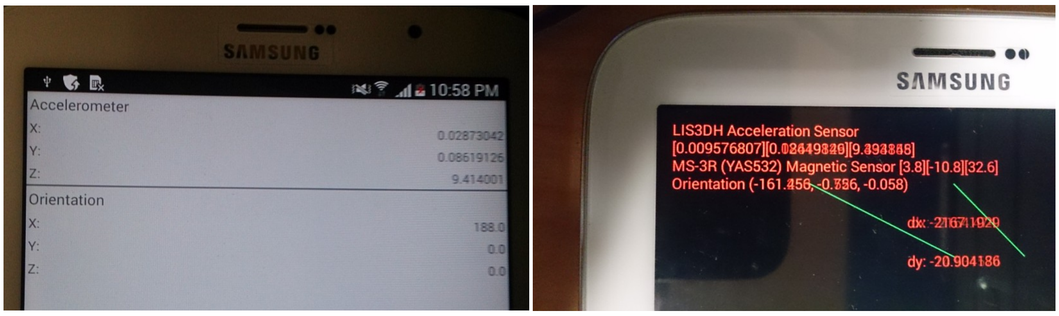
If correct, this should draw the horizon line as well a point on that horizon representing the target location. Unfortunately, this was not the case. Initially, testing of the application saw the horizon and point drawn correctly relative to each other, but incorrectly to the real world. Specifically, this horizon line (green line through the middle of the screen) would be drawn only when the device was tilted towards the ground:



```
LIS3DH Acceleration Sensor
[-0.89064306][0.047884032][9.308656]
MS-3R (YAS532) Magnetic Sensor [17.7][-21.3][16.9]
Orientation (210.972, -0.293, 5.465)
dx: 2831.8374
dy: -8.11097
LIS3DH Acceleration Sensor [-0.89064306][0.047884032][9.308656]
MS-3R (YAS532) Magnetic Sensor [17.7][-21.3][16.9]
Location Data
```

Fig 36. Horizon line drawing

To diagnose this problem, the rotational information was displayed on the overlay of the screen, along with other relevant metrics. Inspecting this, all information being used to construct the overlay appeared correct - rotations and movements along all axes changed as expected when the device was rotated and moved, yet the projection still only appeared while facing the ground. To dig further into this problem, third party applications which display these values were downloaded and tested to ensure correct values were being fed through the data layer of ASIST.


```
Accelerometer
X:        0.02873042
Y:        0.08619126
Z:        9.414001
Orientation
X:        188.0
Y:        0.0
Z:        0.0
```

```
LIS3DH Acceleration Sensor
[0.009576807][0.02649829][9.393868]
MS-3R (YAS532) Magnetic Sensor [3.8][-10.8][32.6]
Orientation (-161.456, -0.356, -0.058)
dx: -216711920
dy: -20.904186
```

Fig 37. Comparing orientation data between third party application (left) and ASIST (right)

This uncovered that all values excluding a single rotational axis were correct, with this incorrect axis being skewed by a certain offset due the dx and dy values from the previous code snippet returning negative infinity (not shown here). Various attempts were made to correct this, but even directly correcting this offset with a fixed value had no impact on this rotational axis. Due to being on the verge of completion, a significant amount of time was spent attempting to remedy this problem, with secondary problems being introduced along the way as a result of different fixes being trialled. Permissions parsing information between classes became an issue during this process, resulting in a large scale refactoring of the LiveView and CameraPreview classes to parse an activity to the constructor of the CameraPreview, rather than a camera object, to grant access to frame of view (FOV) information to correct the above dx and dy calculations. Amongst other methods, the constructor of the CameraPreview was changed as so:

```
public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;
        mHolder = getHolder();
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        mHolder.addCallback(this);
}
```

Fig 38. Old CameraPreview constructor

To:

```
public CameraPreview(Context context, Activity activity){
        super(context);
        mHolder = getHolder();
        mActivity = activity;
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        mHolder.addCallback(this);
}
```

Fig 39. New CameraPreview constructor

Parsing of the camera object as was previously done prevented gaining this FOV information due to only one instance of a camera being able to access the hardware at a time, causing the system to crash whenever a secondary call was made. Although now with a more correct implementation and dx and dy values, the same problem regarding the incorrect rotational axis continued. After further analysis of the methods used, the cause of this problem was still unclear.

To move the project forward, a reassessment of the application's capability requirements was conducted to provide insight on possible alternative methods to solve this problem. Due to all usage of this application being conducted by user's in the field, it can be assumed that the device will be in use while standing at the road side with the tablet being held out in front of the user at a typical constant viewing height. This static parameter can be used to synthesise altitude based on the relative height between the tablet and the road. This meant that altitude information regarding both the road and the device was unneeded and the difference between these two values could be replaced by this fixed height of approximately one metre. Hence development then shifted back towards the earlier solution involving only latitude and longitude.

The next big challenge involving these pseudo altitude values was surrounding how to visualise depth on a two dimensional screen without being able to leverage the rotational matrix as we would like. An alternative way of using the same rotation matrix operations in the 'onSensorChanged()' event was trialled to leverage the required information, resulting in a positive outcome:

```
SensorManager.getRotationMatrix(RTmp, I, gravSensorVals, magSensorVals);
int rotation = Compatibility.getRotation(this);
if (rotation == 1) {
        SensorManager.remapCoordinateSystem(RTmp, SensorManager.AXIS_X, SensorMan-
                ager.AXIS_MINUS_Z, Rot);
} else {
        SensorManager.remapCoordinateSystem(RTmp, SensorManager.AXIS_Y, SensorMan-
                ager.AXIS_MINUS_Z, Rot);
}
SensorManager.getOrientation(Rot, results);
ARView.yaw = (float)(((results[0]*180)/Math.PI)+180);
ARView.pitch = (float)(((results[1]*180/Math.PI))+90);
ARView.roll = (float)(((results[2]*180/Math.PI)));
```

Fig 40. Orientation calculations

Again, the Y position of locations on the screen aren't taking into account any type of altitude data, but they are taking into account the pitch of the device correctly (tilting the tablet towards the sky/ground), resulting in real world points now being displayed in front of the user as expected, rather than only when tilted to face the ground as was the case with the previous implementation.

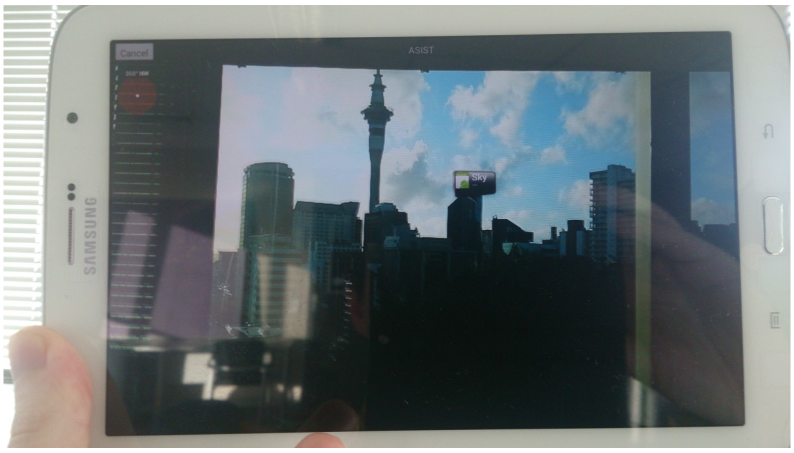


Fig 41. Skytower estimation (using default Android location marker)

In addition to using the estimated one metre difference in height between the user and the ground, other information regarding relative differences between the user and the road can be leveraged to display the road correctly. The main value being the difference in horizontal distance between the user and the road. However before this distance can be used, an understanding of how the road location data is recorded should be explored.

As illustrated in the previously given RAMM data (table 1), location information is given by a road name e.g. "01S-0933", a road ID e.g. "797", and a displacement relative to that road name and ID e.g. "0-100m" which signifies that row of road information correlates to the first 100 metres of that road, while a displacement of "210-510m" refers to the piece of road between the 210 and 510 metre mark, from the beginning of that road. The given road names then correspond to a real English name under the start name attribute, which can be used to find a GPS coordinate - which is required by the application to represent the road's position. Knowing this information then allows for the distance between the user and the start of the road, as well as the end of the road be calculated:

```
        float [] dist = new float[1];
        Location.distanceBetween(currentLocation.getLatitude(), currentLocation.getLongitude(),
                targetLocation.getLatitude(), targetLocation.getLongitude(), dist);
```

Fig 42. Distance between two points calculation

The idea here is that assuming a flat surface, the further the road is from the tablet, the higher on the screen the road should appear. Additionally, the road grows narrower in the distance:

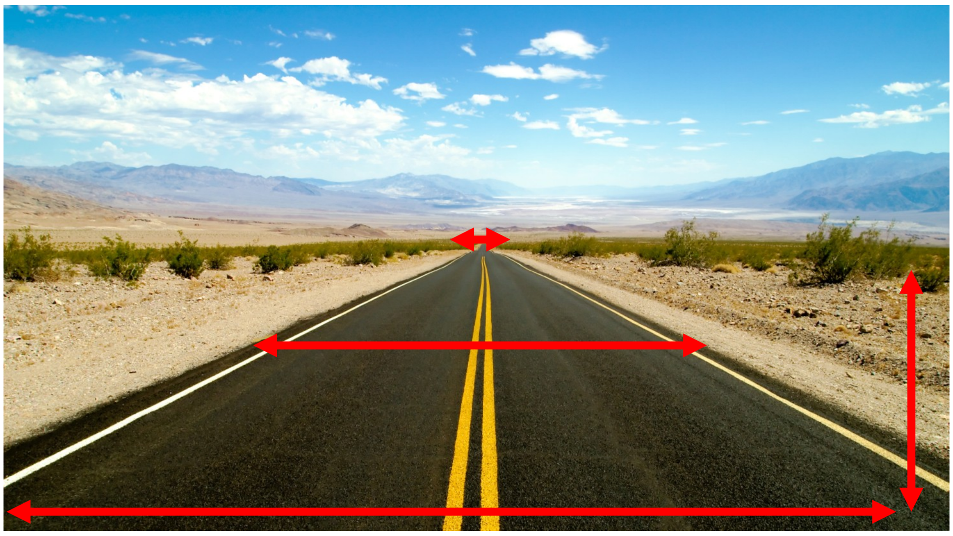


Fig 43. Road dimensions

Having access to all of this information then poses the design question of how to best use the information to represent the data. The first option considered followed the same principle as the previously trialled horizon line visualisation where the line drawn will be bounded by the two coordinate points representing the start and end of a road. Given the user is standing on the road, a line should be drawn underfoot, extending both in front of, and behind the user when the device is tilted in these directions. However the more reliance that is put upon using GPS coordinate points, the more susceptible to accuracy issues the application becomes. Given the application uses a single coordinate to track a road, there are only two places for error; the difference between the true coordinate of that road and the coordinate estimation of that position given by the satellites used, and the same difference regarding tracking of the device itself. When additional coordinates are used to construct a road representation, the number of those errors increases. Furthermore, it is unfortunately not the case that these multiple errors will correct each other, nor highlight the offset required to correct the errors as testing has shown that they are inconsistently different, varying in direction and magnitude depending on satellite position and weather conditions. This results in a coordinate displaying correctly one day, to the east by three metres the next, and to the west by five metres the following day. If all coordinate data were correct, this would be a highly valid solution, granting usability of the application from any viewing angle, however this is not the case.

To solve the issue, simplifications of the proposed method were trialled. Using the knowledge that a road appears higher and narrower as the distance apart increases, as well as assuming a fixed viewing elevation above the road, the visualisation can become increasingly static, while still conveying the required information. Constructions of lines were trialled before choosing to represent a road as a trapezium which fulfils the requirements of becoming narrower and taller in the distance:

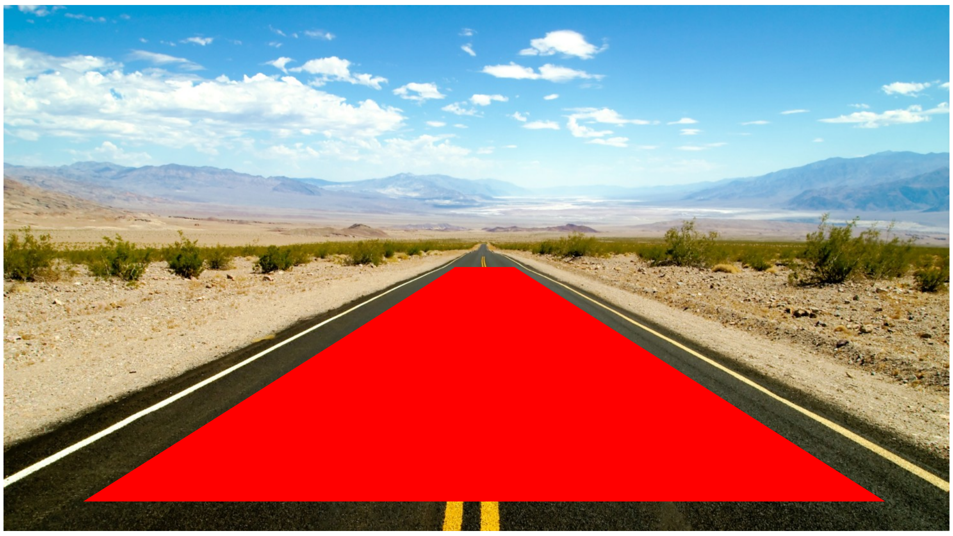


Fig 44. Trapezium road estimation

However this also introduces the largest usability limitation of the application. If this technique is used, then viewing of road condition information can only be completed correctly when looking straight down a road as above, rather than standing on the side of the road and looking at it like so:

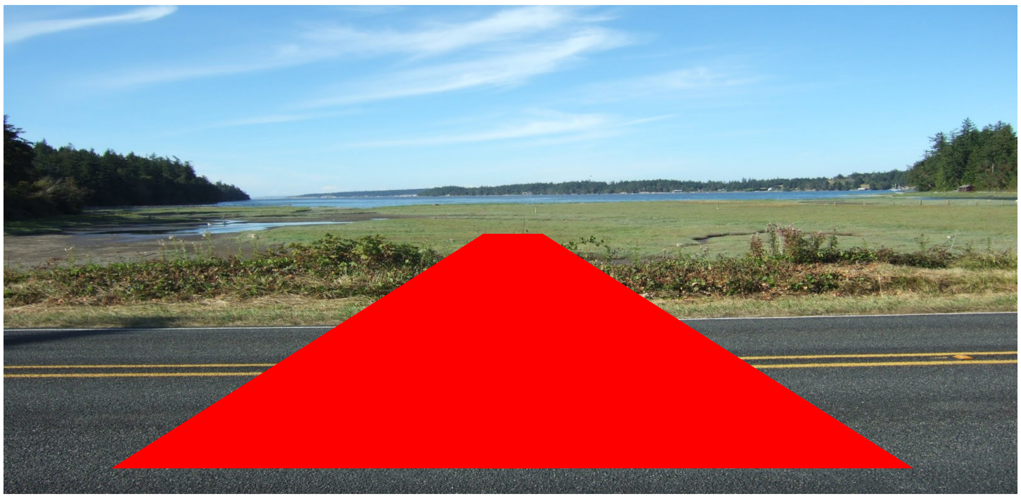


Fig 45. Limitation of trapezium from road side

This restricts the usability of the system, disallowing any other viewing angle from displaying information correctly. However as illustrated in images taken from the operational functions report (fig 1, 6), all concept work of the application displays this limitation, and the overall effectiveness of the application isn't hindered dramatically due to the ease of moving to stand in this required position at the end of the road. Additionally, the difference in angle between looking straight down a road from standing in the centre, compared to doing the same from the side of the road will be minimal, so little that this solution may be viable from this viewing angle as well. Later images will show the validity of this approach when standing on the side of the road, but looking down it to display a trapezium that still corresponds to the road shape.

Using this trapezium method, the final question at this stage is where on the road should the GPS coordinate keep track of? Since the starting position and length of the road is given, that starting position was first trialled as the GPS coordinate to be used for the trapezium. Unfortunately due to the accuracy issues mentioned, if standing in close proximity to this GPS coordinate while using ASIST, the placement of the coordinate relative to the tablet was inconsistent. This resulted in the trapezium only being drawn correctly approximately a quarter of the time, while it was being drawn off screen the other three quarters of the time due to the satellites estimating the given position to be behind or beside the user, rather than in front.

To negate this issue, the GPS coordinate was changed to track the centre point, half way down the road. This alleviated the above problem, but introduced others concerning the positioning of the trapezium on the screen due to the distance from user value changing. After strenuous testing, the following was produced to position road representations on the screen:

```
void drawRoad(PaintUtils dw) {

        currentLocation.setLatitude(ARView.lastLocation.getLatitude());
        currentLocation.setLongitude(ARView.lastLocation.getLongitude());

        for(int i = 0; i <latitudes.length;i++){
                destinedLocation.setLatitude(latitudes[i]);
                destinedLocation.setLongitude(longitudes[i]);

                bearing = currentLocation.bearingTo(destinedLocation);

                if(bearing < 0){
                        bearing  = 360 + bearing;
                }
                bearings[i] = bearing;

        }

        for(int i = 0; i<bearings.length;i++){
                float [] dist = new float[1];
                Location.distanceBetween(currentLocation.getLatitude(), currentLoca-
                        tion.getLongitude(), latitudes[i], longitudes[i], dist);

                if(bearings[i]<0){

                        yPosition = (this.pitch - 90) * this.degreetopixelHeight+400+dist[0];

                        bearings[i] = 360 - bearings[i];
                        angleToShift = (float)bearings[i] - this.yaw;
                        nextXofText[i] = (int)(angleToShift*degreetopixelWidth);
                        yawPrevious = this.yaw;
                        isDrawing = true;
                        roadRenderer(dw, places[i], nextXofText[i], yPosition, true, true, i);
                        coordinateArray[i][0] =  nextXofText[i];
                        coordinateArray[i][1] =   (int)yPosition;

                }else{
                        angleToShift = (float)bearings[i] - this.yaw;

                        yPosition = (this.pitch - 90) * this.degreetopixelHeight+400 +dist[0];

                        nextXofText[i] = (int)((displayMetrics.widthPixels/2)+
                                (angleToShift*degreetopixelWidth));
                        if(Math.abs(coordinateArray[i][0] - nextXofText[i]) > 50){
                                roadRenderer(dw, places[i], (nextXofText[i]), yPosition, true,
                                        true, i);
                                coordinateArray[i][0] =  (int)((displayMetrics.widthPixels/2)+
                                        (angleToShift*degreetopixelWidth));
                                coordinateArray[i][1] =  (int)yPosition;

                                isDrawing = true;
                        }else{
                                roadRenderer(dw, places[i],coordinateArray[i][0],yPosition, true,
                                        true, i);
                                isDrawing = false;
                        }
                }
        }
}
```

Fig 46. Road drawing positioning

Where ‘roadRenderer()’ deals with resizing the trapezium. This sizing first used fixed values to estimate the graphic required to cover the road surface:

```
    void roadRenderer(PaintUtils dw, String txt, float x, float y, boolean test, boolean isRoad, int
            count) {

            float padw = 4, padh = 2;
            float w = dw.getTextWidth(txt) + padw * 2;
            float h;
            if(isRoad){
                    h = dw.getTextAsc() + dw.getTextDesc() + padh * 2+10;
            }else{
                    h = dw.getTextAsc() + dw.getTextDesc() + padh * 2;
            }
            if (test) {

                    if(isRoad){
                            layoutParams[count].setMargins((int)(x - w / 2 - 10), (int)(y - h / 2 - 10),
                            0, 0);

                            //Road Height:
                            layoutParams[count].height = 500;

                            //Road Width:
                            layoutParams[count].width = 120;

                            locationMarkerView[count].setLayoutParams(layoutParams[count]);

                            //Puts the street name in the middle (vertically) of the road graphic:
                            subjectTextViewParams[count].topMargin = 2*(int)roadLengths
                                    [count];
                            subjectTextViewParams[count].leftMargin = 10*(int)roadWidths
                                    [count];

                            locationTextView[count].setLayoutParams(subjectTextViewParams
                                    [count]);
                    }
            }
    }
```

Fig 47. Road drawing sizing

However the testing environment used wasn’t representative of all roads due to the decreasing elevation, and so the final step with respect to controlling the size of the trapezium came from leveraging the information recorded about roads, specifically the road length and road width values. Again, varying values were trialled to find the most accurate and robust solution, resulting in the following values and performance:

```
    //Change height based on length of road:
    layoutParams[count].height = 4*(int)roadLengths[count];

    //Change width of graphic based on width of road:
    layoutParams[count].width = 40*(int)roadWidths[count];
```

Fig 48. Variable road sizing

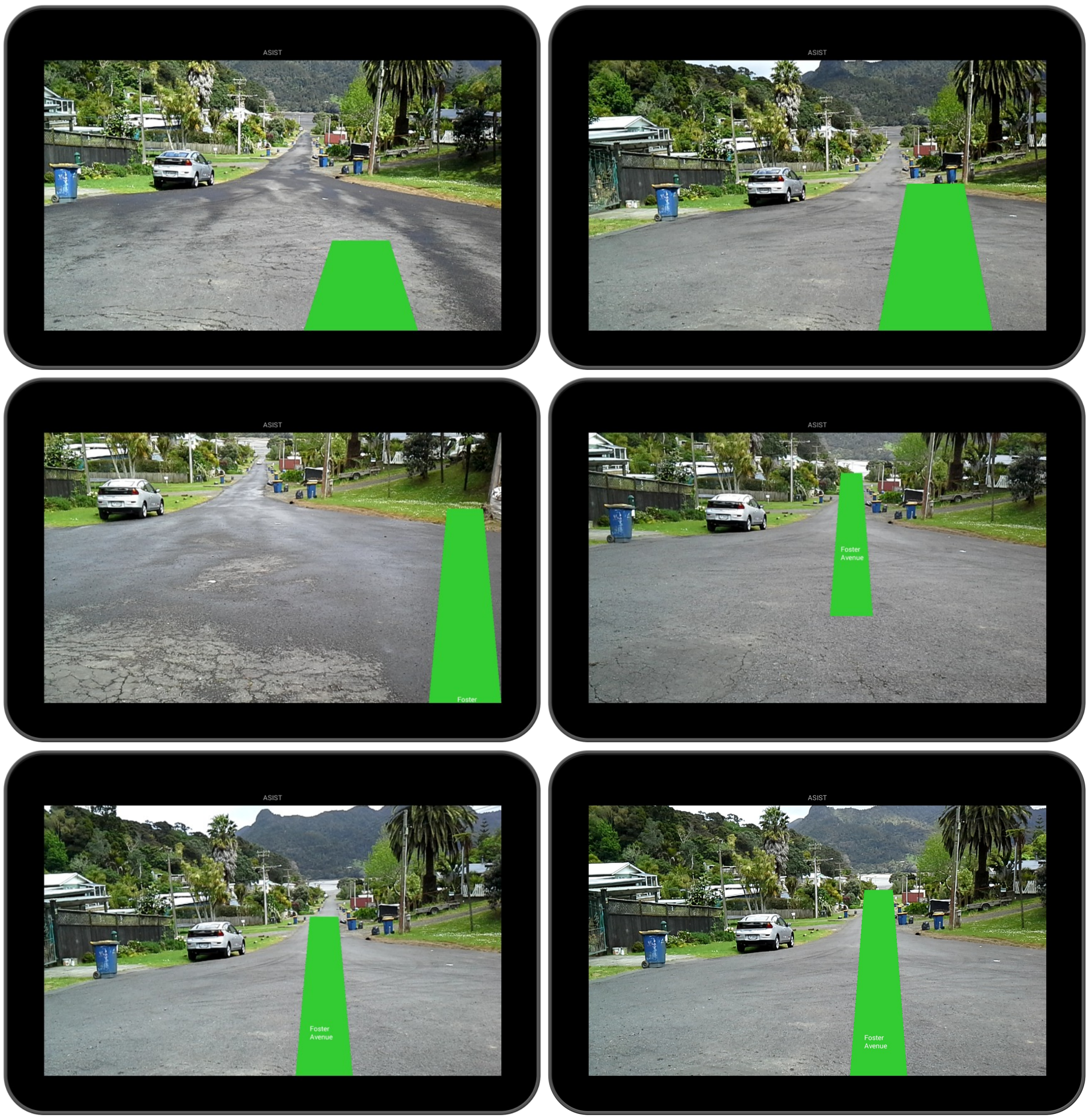


Fig 49. Road drawing trials with varying parameters

The road in the above images widens dramatically when moving towards where the images were taken, hence why a wider base for the trapezium hasn't been used as this road is unrepresentative of a normal, fixed width road.

Due to the unforeseen complications throughout the development of this data layer and the time restrictions of this project, this concludes the development of the road representation, excluding minor details such as displaying the correct condition type which has been simplified by the use of local test data to change the colour of the road:

```
if(conditions[i].equals("Good")){
        locationMarkerView[i].setBackgroundResource(R.drawable.trapgreen5);
}
else if(conditions[i].equals("Bad")){
        locationMarkerView[i].setBackgroundResource(R.drawable.trapred5);
}
```

Fig 50. Road drawing condition

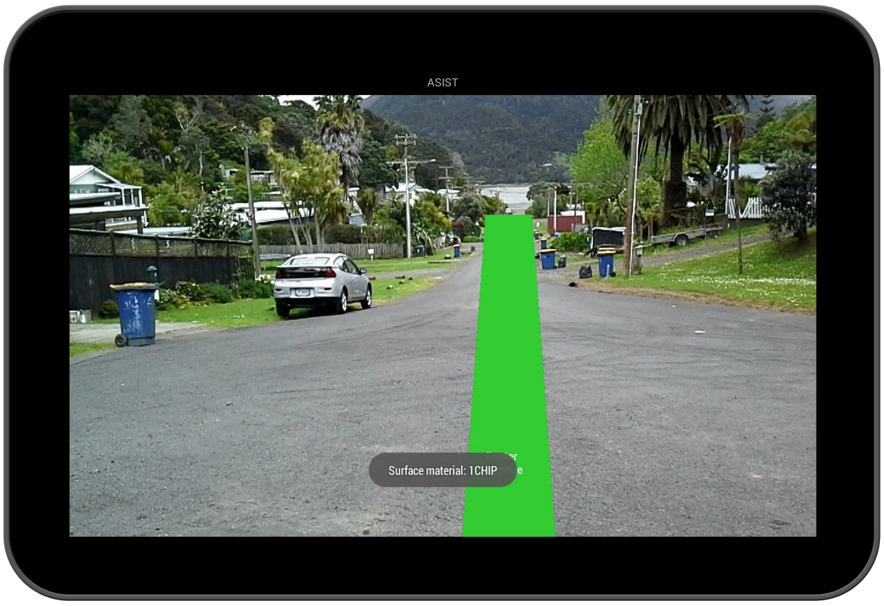
3.6 Additional Features

The goal for ASIST once representation of roads was complete was to incorporate the asset management features discussed previously. However with little time remaining to complete the project, and the increased likelihood of running into time consuming problems when creating new files like this, it was decided that this remaining time would be spent adding extra functionality to the live view feature to allow for future work of this function to be continued with less effort. Additionally, the creation of the asset management function again poses further design questions on the best way to achieve results, further increasing the time required to deliver tangible results. Due to the majority of features requiring a network connection to communicate with central servers, an array of methods could be trialled, including a web portal within the application itself.

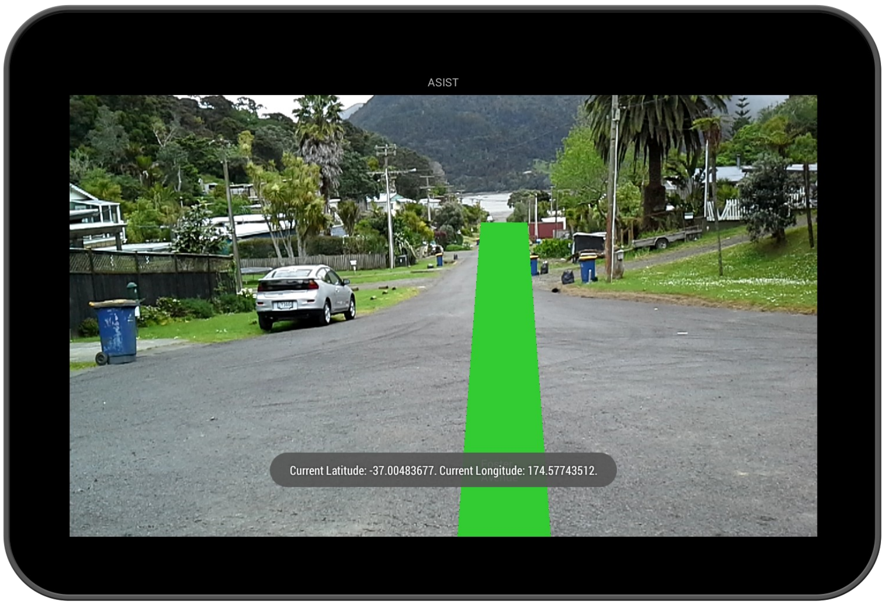
To facilitate future work of the live view feature, user interaction has been enabled by the addition of touch controls. For the time being, the information displayed is placeholder data which can be built on at a later date where more elaborate touch events can be created. Two main touch events have been created. The first displays the surface material of the road when a trapezium is touched. While the second displays coordinate information of the user when the remaining area of the screen is touched:

```
subjectImageView[i].setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {

        if (v.getId() != -1) {

            RelativeLayout.LayoutParams params = (RelativeLayout.LayoutParams)
                locationMarkerView[v.getId()].getLayoutParams();
            Rect rect = new Rect(params.leftMargin, params.topMargin,
            params.leftMargin + params.width, params.topMargin + params.height);
            ArrayList<Integer> matchIDs = new ArrayList<Integer>();
            Rect compRect = new Rect();
            int index = 0;
            for (RelativeLayout.LayoutParams layoutparams : layoutParams) {
                compRect.set(layoutparams.leftMargin, layoutparams.topMargin,
                layoutparams.leftMargin + layoutparams.width, layoutpar-
                    ams.topMargin + layoutparams.height);
                if (compRect.intersect(rect)) {
                    matchIDs.add(index);
                }
                index++;
            }
            Toast.makeText(_context, "Surface material: " + surfaceMaterial[j],
                Toast.LENGTH_SHORT).show();
        }
    }
});
```

Fig 50. Touch event - road touched



Surface material: 1CHIP



Current Latitude: -37.00483677. Current Longitude: 174.57743512.

Fig 51. Touch events

3.7 Summary of Features

As discussed in section 1.8 *Conceptual ASIST*, it was planned for this project to entail not only the rendering of roads in real time, but to include the capability of displaying varying pieces of information such as overall road condition, skid resistance, and age of surface. In addition to this, asset history was to be encoded, allowing users to control the date of information they wish to view via an on screen time slider. Furthermore, non-road assets such as culverts and buildings were to be represented as well, with similar history information options as available for roads. Lastly, the back end asset management features were planned to be included, automatically extracting information from the live view and using it to auto-complete required fields of site inspection reports.

As discussed throughout the development process, challenges were faced which restricted the number of features delivered by this project. This lead to the scope of the project being reduced to delivering only the fundamental features required to represent a road surface. The final project delivers the following capabilities:

- Local databases of information store a road's: GPS coordinates, length, width, and overall condition
- These attributes are then used to: position, size, and colour a road on the augmented data layer, corresponding to that road's real world position
- The road name is also displayed on the road surface
- When the user looks straight down a road from either end, the graphic overlay will be displayed correctly:
-

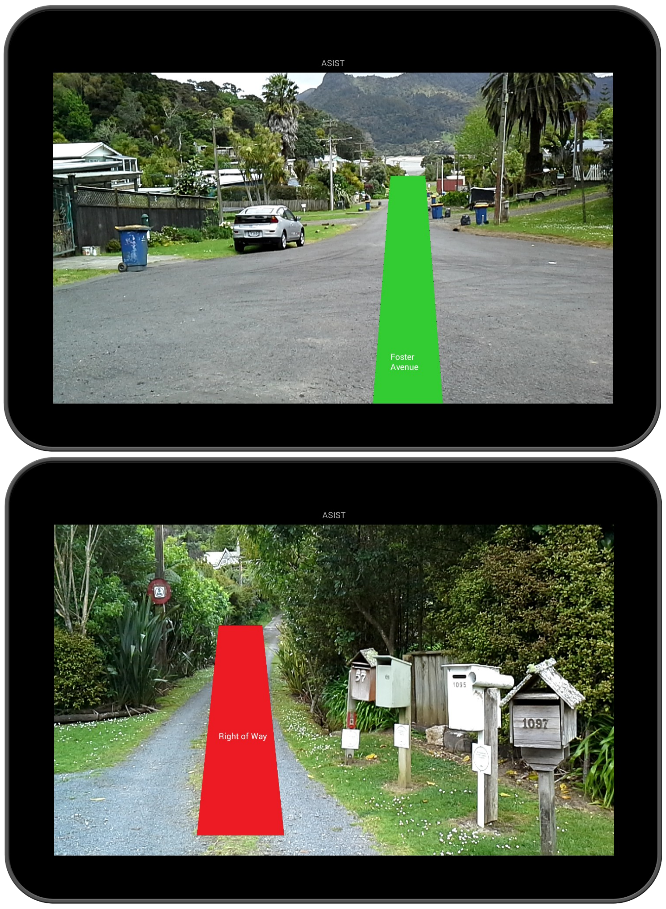


Fig 52. Looking down a long, good condition road (top) & looking up a short, poor condition road (bottom)

- When looking at the road in the same direction, but standing from the side, the accuracy of the graphic overlay will be less than that of above, but still satisfactory to display the required information:

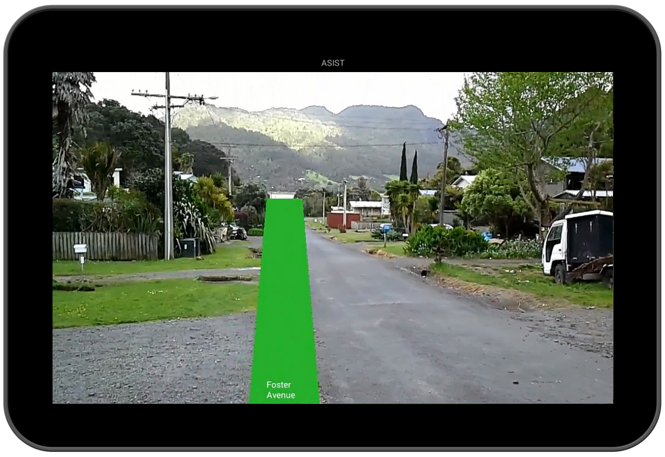
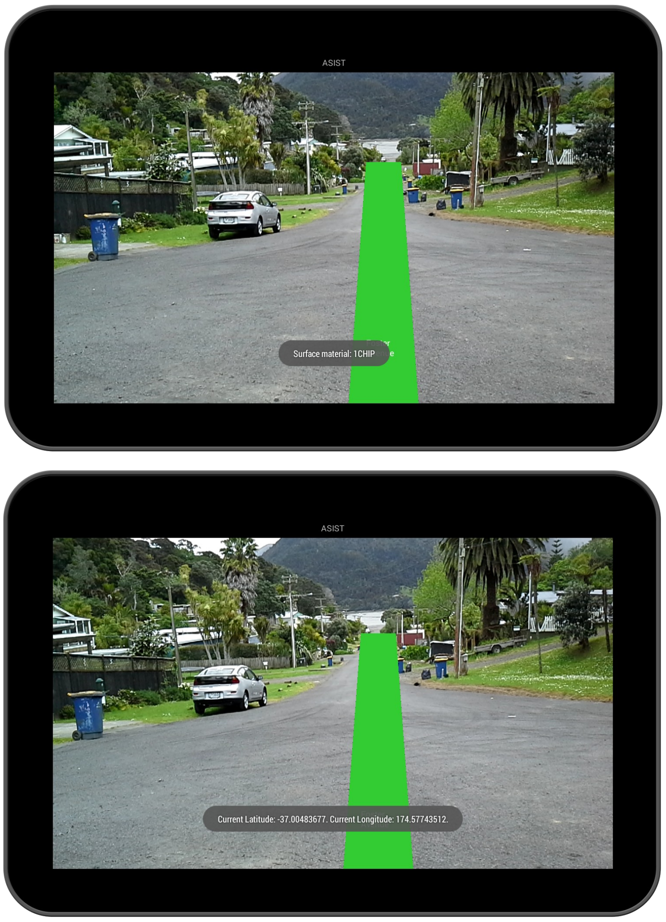


Fig 53. Side of road viewing

- On screen touch events allow users to inspect the road surface material and current GPS coordinates by touching the road and surrounding area, respectively:



Surface material: 1CHIP

Current Latitude: -37.00483677. Current Longitude: 174.57743512.

Fig 54. Touch events for surface material (top) and current location coordinates (bottom)

IV. CONCLUSIONS

4.1 Evaluation

A. Performance

Due to the difficulties faced and limitations of use imposed throughout development, ASIST has moved from the intended Augview-like design towards an application more reminiscent of a POI application, as well as an overall reduction in scope to entail only the core roading functionality. The capabilities of POI applications vary from very simplistic and only taking into account the yaw of the device, to the most elaborate incorporating all three axes of rotation, as well as depth/distance, and application specific parameters. ASIST sits at the more elaborate end of the POI spectrum and grows on the alternatives surveyed by rendering the points of interest in what appears three dimensions over the camera perspective.

The viewing angle associated with regular POI applications is not a factor as it is with ASIST due to the location markers in these applications representing a single point, rather than a piece of road in the real world space. This is one area where ASIST will continue to fall short as long as this implementation is used. However the limitation only affects looking across a road, while the constraint of looking down a road from one end is loosened to include looking down a road from most sides with an acceptable level of alignment error:

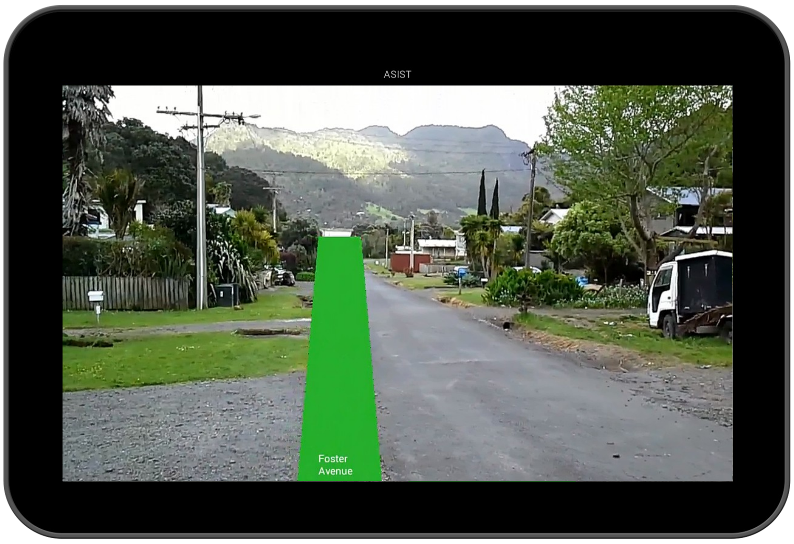


Fig 55. Roadside alignment

The performance of the application when these restrictions are imposed is satisfactory. Correct visualisation of a road in real time has been the major design challenge throughout this project, and although simplified, the solution provided conveys the required information. In most cases the accuracy of the visualisation is adequate, covering the road as expected. However as will be covered in the following usability and future work sections, the information conveyed is simplistic, and the performance of the application can be improved by the inclusion of additional data layers with varying categories of information regarding the use case at hand.

As mentioned in section 3.5, satellite coordinate information can be inconsistent and vary from day to day, as well as be affected by the weather conditions. Although the proposed solution has sought to minimise this variation, it is at times unavoidable. The following shows a worst case scenario in which all factors regarding the data have been held constant, with the only difference being that the two images have been taken 24 hours apart:

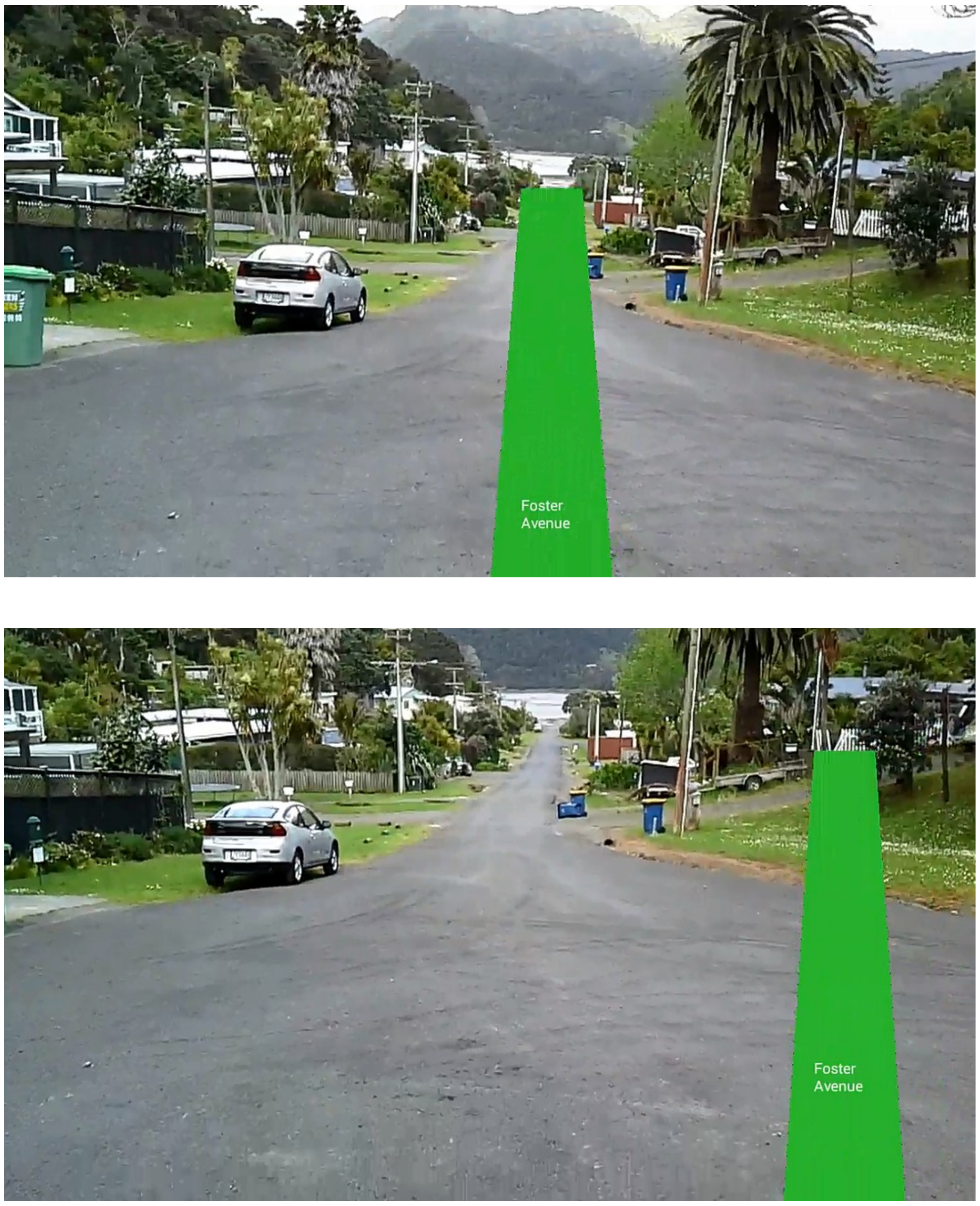


Fig 56. Differing GPS coordinate estimations

B. Usability

Implementation specifics aside, the overall usability of the live view function has been evaluated against Nielsen's 10 usability heuristics where appropriate [45]:

1. Visibility of system status
2. Match between system and real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Error recognition, diagnosis, and recovery
10. Help and documentation

This has been conducted to highlight not only strengths in ASIST's design, but where future improvements can be made:

ASIST has been developed to maintain a minimalistic design while conveying information. The aesthetics and colours used are consistent, intuitive, and their meaning easily understood. Colours such as green for good, and red for poor condition promote recognition rather than recall when it comes to assessing road condition. However only limited information about the road surface is given - the overall condition of the road. Progressive disclosure has been used to display further information when touched in an attempt to remedy this lack of information, but different road textures for example could be added to provide additional, constantly view-able information.

Due to the minimalistic design, the visibility of system status is limited and could be improved to increase the user's awareness of the system. The main area of concern in this respect is knowing the status of the satellite connection. Without any interaction with the screen, users cannot determine whether location services are working correctly. This is quickly accessed by pulling down the system tool bar, but additional on screen cues could be added to eliminate this process.

The overall control of the application is instinctive, with rotations of the device resulting in expected responses from the system in terms of road visualisation movement. Of course the limitation of looking down a road doesn't fulfil this property, and without explanation of this shortcoming, users will become impatient with the application. Therefore flexibility of the system is limited, and should be an area of focus for future work.

The main error that can occur is a result of poor satellite accuracy, causing a road to be estimated in the wrong location as previously mentioned. Again an explanation of the application's capabilities in terms of where a road can be best viewed from will minimise the chance of this problem occurring. However in the event that a road is plotted incorrectly behind the user, no options for re-syncing coordinate information is available. This is another feature that could be added in the future.

Security

The main security concern with ASIST is the transmission and storage of sensitive information between tablet and server, as well as any malicious software on the device extracting locally stored information from the application. Due to this project primarily taking the form of a proof of concept, concerned with the validity of an augmented reality solution, this particular is outside the main scope of the project and so has not been an area of focus. The majority of data used throughout testing has been location data generated specifically for the illustrated testing environments, and is therefore all publically accessible via Google Maps. The remaining information has again been generated for the testing environments used and is thus insensitive data. The only valuable information used within this project is the structure of road data which defines what information is stored about New Zealand roads. However the majority of these attributes are very typical and what would be expected, for example "road width" and "surface material." If the ASIST application is developed further to the point where caching of remote information is implemented, then security and encryption measures should be put in place, but at this stage they are an unnecessary cost.

4.2 Future work

Various paths for future work can be taken. These can be split into two distinct categories: further development of the live view function, and development of complimentary features.

Future Live View Development

Firstly, the fundamental structure of the live view function should be assessed. If the current limitations of the application are at an acceptable level, then development can continue as usual. Alternatively, user options can be used to control this representation. In the simplest implementation, a button can be used to notify the system whether the user is looking down, or across a road. This will in turn rotate the road augmentation as required to eliminate the limitation of needing to look down the road. However if this is found unsuitable, then more time should be spent on the more comprehensive, less restrictive solution trialled earlier on in development. In either case, the next step is to then convey more information to the user on screen. This will be done in two ways: providing information that is always visible, and adding additional touch options for further information. On top of this, various preset options for constant display of different information should be made available. An example of these ideas include:

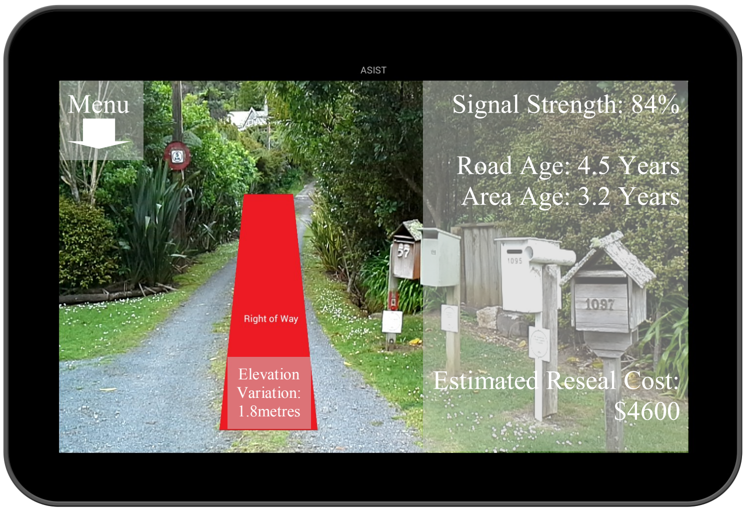
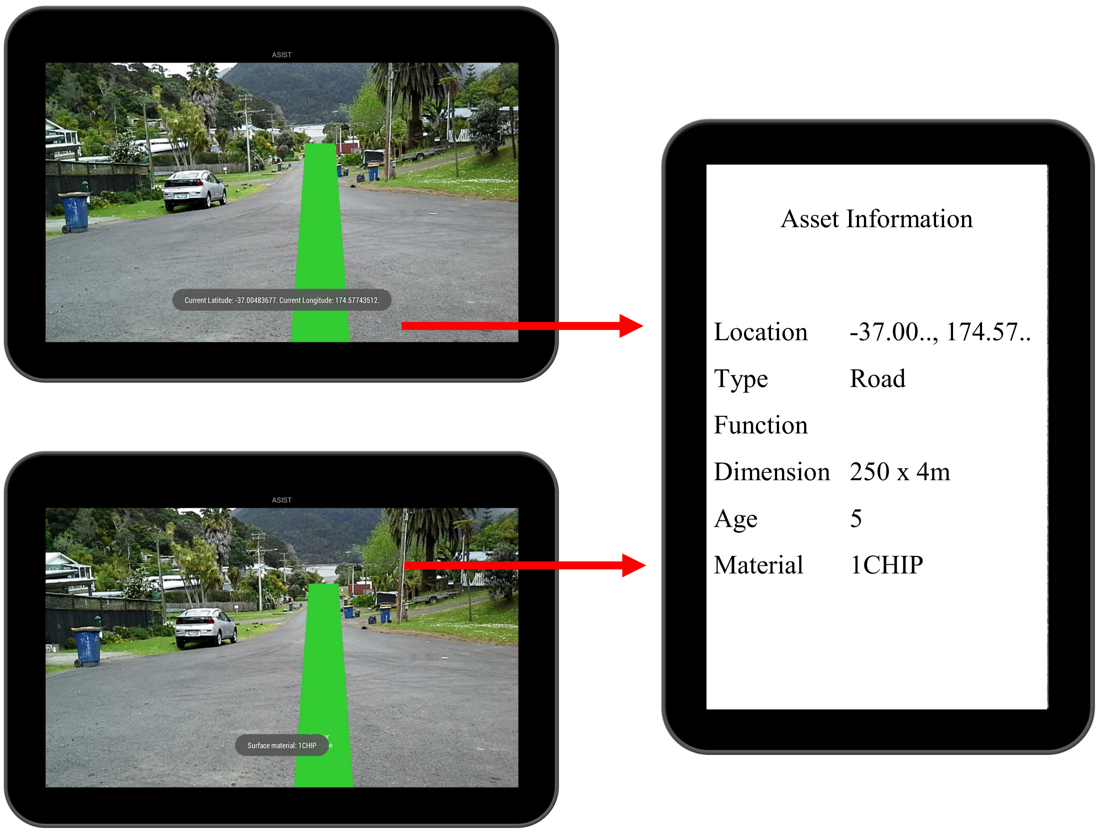


Fig 57. Information overlays

As shown in figure 54, the additional information includes that displayed on the road itself, as well as using the remaining screen real estate to display various information. In addition to a road colour representing the overall condition of the road, various textures representing the surface material can be displayed. Variations of this can then be used to indicate the age and other variables related to that road. Depending on the complexity of these graphics, a key or legend may need to be displayed to describe the texture. Other off-road information to be displayed includes that concerned with the road in question, as well as information based on the user's surroundings. For example, the average age of nearby roads (average time since last resurface) which can aid the decision of whether or not the road in question should be repaired.

As previously discussed, it was intended for the asset management features to follow the development of the live view function. Unfortunately due to time restraints and difficulties faced, this did not occur. Once any necessary additions have been made to the live view function, these behind the scenes processes which make the application a business viable product must be developed. Due to the majority of functions having a straight forward layout consisting of a range of text entry fields, the design of these sections shouldn't be time consuming. The challenge will come from correctly connecting these managerial functions with the live view. To make ASIST a viable solution, maximal automation between these functions is necessary. This will include auto-completion of text entry fields such as road name and location based on the perspective of the live view when requesting a road upgrade:



Current Latitude: -37.00483677. Current Longitude: 174.57743512.

Surface material: 1CHIP

Asset Information

Location	-37.00.., 174.57..
Type	Road
Function	
Dimension	250 x 4m
Age	5
Material	1CHIP

Fig 58. Live view - asset management connection

Once the local connections between functions are made, distributed connections between device and server will need to be implemented. This is where the security measures as mentioned previously must be administered.

4.3 Lessons Learned

From accepting this project to the time of writing, this entire endeavour has been an ongoing learning process. The usual early steps of the project design process were bypassed due to inheriting an already complete design from the given documentation, meaning evaluation of design and alternative data representations were unnecessary to explore. However this introduced other challenges in the form of project understanding and negotiation. Learning to interpret design ideas and transform these into manageable development tasks was the first main challenge to overcome. Parallel to this, and what became the biggest lesson learned overall was the general communication skills, both written and oral, required throughout the project. Ongoing communications were made with industry mentor Kodie in which technical details could be discussed frequently at a low level. However members from other parts of the firm were also involved, requiring information to be disseminated at a higher level. Ensuring all individuals involved regardless of their background and experience were kept on the same page throughout development was an important part of this project, and a valuable skill to take away. Documents were written with this in mind, only going into low level detail when required, and ensuring that this was explained clearly for all to understand. The second half of this communication aspect is in regard to exaggerating transparency in thought processes and reasoning. Because this is a real project, with real people involved, decisions on implementation cannot be made in secrecy. Again it was vital to keep everyone involved on the same page, and this extends to the reasoning behind decisions made throughout the project. In the scoping and design phase of the project, effort was made emphasising the elaboration regarding the research surrounding different mobile platforms and hardware, resulting in multiple reports being delivered to, and meetings being held with Opus.

Second to the communication and business skills learned throughout the project, software development knowledge was grown week upon week during the second phase of this project. The small amount of web development for mobile was a new experience, requiring Cordova scripts and new user interface frameworks to be used. However the main area of self-growth came from the ongoing Android development. Being new to Android development, initialising the project and it's respective files correctly was a learning experience in itself. Throughout development, many Android specific requirements were learned:

Use of hardware

To gain access to hardware features such as the camera or GPS, multiple requirements must be met. First the application must be made aware that this function is wanted by granting it access in the manifest file. Then due to the access being made through an interface, classes must be made to implement the interface, along with all methods defined by that interface. Finally any hardware specific restrictions must be abided to, such as only a single instance of the camera being requested at a time.

Location

Location tracking is becoming one of the most commonly used features in mobile development, and although straightforward to leverage in comparison to the other hardware used, being able to harness this information is very applicable when developing for an array of application types.

Rotation

As discussed, leveraging the rotation of the device correctly became part of the most difficult and time consuming endeavour throughout this project. This is a commonly used feature and a main input technique for smart devices however, so like location is a fundamental technique that can be transferred to a variety of applications.

General development skills were also grown as a result of completing this project. By working in a distributed environment where the industry supervisor was not physically available, source control was required. From the importance of correct commit messages, to setting up and working with repositories from directly within the IDE, this knowledge will undoubtedly be useful for future projects in the workplace.

REFERENCES

[1] The World Bank Group. (2014). Mobile cellular subscriptions. Retrieved from http://data.worldbank.org/indicator/IT.CEL.SETS.P2/countries?display=graph. Last visited 14.04.2015.

[2] Darcey, L. & Conder, S. Android Wireless Application Development: Introducing Android. ISBN-13: 978-0321627094. Addison-Wesley Professional Publishing. 2009.

[3] Statista: The Statistics Portal. (2014). Number of apps available in leading app stores as of July 2014. Retrieved from http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Last visited 15.04.2015.

[4] Statista: The Statistics Portal. (2015). Most popular Apple App Store categories in March 2015, by share of available apps. Retrieved from http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/. Last visited 15.04.2015.

[5] Dr. Dobb's: The World of Software Development. (2014). Developing Cross-Platform Mobile Apps with HTML5 and Intel XDK. Retrieved from http://www.drdobbs.com/mobile/developing-cross-platform-mobile-app. Last visited 18.06.2015.

[6] Laurila, J. K et al. The Mobile Data Challenge: Big Data for Mobile Computing Research. *Pervasive Computing*. No. EPFL-CONF-192489. 2012.

[7] GSM Arena. (2015). Samsung Galaxy S6. Retrieved from http://www.gsmarena.com/samsung_galaxy_s6-6849.php. Last visited 15.04.2015.

[8] Lindstrom, L. & Jeffries, R. Extreme Programming and Agile Software Methodologies. *Information Systems Management*. Volume 21, issue 3, pages 41-52, 2004.

[9] Chow, T. & Cao, Dac-Buu. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*. Volume 81, issue 6, pages 961-971, 2008.

[10] Wasserman, A. Software Engineering Issues for Mobile Application Development. *FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research.* Pages 397-400, 2010.

[11] Fling, B. Mobile Design and Development. ISBN-13: 978-0596155445. O'Reilly Media Publishing. 2009.

[12] Wu, H. et al. Current status, opportunities and challenges of augmented reality in education. *Computers & Education*. Volume 62, pages 41-49, 2013.

[13] Drascic, D. & Milgram, P. Perceptual issues in augmented reality. *Electronic Imaging: Science & Technology*. Pages 123-134, 1996.

[14] Kurt, S. (2013). Meet your child's new teacher: the iPhone. Retrieved from http://www.http://appletoolbox.com/2013/08/meet-ms-siri-johnnys-new-teacher/. Last visited 27.10.2015.

[15] Young, O. (2013). Augmented Reality Technology Shapes the Future of Retail Commerce. Retrieved from http://www.augmentedrealitytrends.com/augmented-reality-retail/future-of-retail-and-commerce.html. Last visited 24.10.2015.

[16] Tobi. (2015). Augmented Reality Dressing Room. Retrieved from http://modular4kc.com/2009/12/16/tobi-coms-augmented-reality-dressing-room/. Last visited 24.10.2015.

[17] Kishino, F. & Milgram, P. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*. Volume 77, issue 12, pages 1321-1329, 1994.

[18] Azuma, R. Tracking requirements for augmented reality. *Communications of the ACM - Special issue on computer augmented environments: back to the real world*. Volume 36, issue 7, pages 50-51, 1993.

[19] Wada, T. (2014). Trident International Graphics Workshop. Retrieved from http://www.slideshare.net/TakaoWada/graphics-workshop2014-2. Last visited 25.10.2015.

[20] Kutulakos, K. & Vallino, J. Affine Object Representation for Calibration-Free Augmented Reality. *Virtual Reality Annual International Symposium,1996., Proceedings of the IEEE 1996*. Pages 25-36.

[21] Satyanarayanan, M. Pervasive computing: Vision and challenges. *Personal Communications, IEEE.* Volume 8, issue 4, pages 10-17, 2001.

[22] Schmidt, A. Ubiquitous computing - Computing in context. *Ph.D. thesis, Lancaster University*. 2002.

[23] Olsson, T. et al. User evaluation of mobile augmented reality scenarios. *Journal of Ambient Intelligence and Smart Environments*. Volume 4, issue 1, pages 29-47, 2012.

[24] Chen, Hsinchun, Roger HL Chiang, and Veda C. Storey. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS quarterly* 36.4. Pages 1165-1188. 2012.

[25] Heim, S. The Resonant Interface: HCI Foundations for Interaction Design. ISBN-13: 978-0321375964. Addison-Wesley Publishing. 2008.

[26] Bruce, A. Big Ideas 2014 - ASIST. *Functional Requirements Report*. 2014.

[27] Miller, C. & Doering, A. The New Landscape of Mobile Learning: Redesigning Education in an App-Based World. First Edition. ISBN-13: 978-0415539241. Routledge Publishing. 2014.

[28] Madden, L. *Professional augmented reality browsers for smartphones: programming for junaio, layar and wikitude*. John Wiley & Sons, 2011.

[29] Analytical Graphics, Inc. (2011). Satellite AR. Retrieved from https://play.google.com/store/apps/details?id=com.agi.android.augmentedreality&hl=en. Last visited 22.08.2015.
[30] ESET. (2014). ESET Augmented Reality BETA. Retrieved from: https://play.google.com/store/apps/details?id=com.eset.ar&hl=en. Last visited 22.08.2015.
[31] Goodchild, M. Geographic information system. *Encyclopedia of Database Systems*. Springer US. 1231-1236. 2009.
[32] Augview. Retrieved from http://www.augview.net/. Last visited 09.07.2015..
[33] Fernandez, Wilkins, and Stephan Alber. Introduction. *Beginning App Development with Parse and Phonegap*. Apress. Pages 1-13. 2015.
[34] Karadimce, Aleksandar, and Dijana Capeska Bogatinoska. Using hybrid mobile applications for adaptive multimedia content delivery. *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*. IEEE, 2014.
[35] Hou, Lei, et al. Combining photogrammetry and augmented reality towards an integrated facility management system for the oil industry. *Proceedings of the IEEE* 102.2 (2014): 204-220.
[36] Niculescu, Dragos, and Badri Nath. Ad hoc positioning system (APS). *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*. Vol. 5. IEEE, 2001.
[37] Hofmann-Wellenhof, Bernhard, Herbert Lichtenegger, and Elmar Wasle. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media, 2007.
[38] Padre, J. (2012). GLONASS support in our latest Xperia phones. Retrieved from Retrieved from: http://developer.sonymobile.com/2012/01/19/glonass-support-in-our-latest-xperia-phones/. Last visited 13.08.2015.
[39] Qualcomm. (2015). Retrieved from https://www.qualcomm.com. Last visited 27.10.2015.
[40] iPad Pilot News. (2013). Do I really need an external GPS? Retrieved from: http://ipadpilotnews.com/2013/02/do-i-really-need-an-external-gps-2/. Last visited 10.08.2015.
[41] Burkert, C. (2013). iPad Air - GPS / GLONASS. Retrieved from: https://discussions.apple.com/thread/5550207. Last visited 10.08.2015.
[42] Steele, J. & To, N. The Android Developer's Cookbook: Building Applications with the Android SDK. Second edition, ISBN-13: 978-0321741233. Addison-Wesley Publishing. 2013.
[43] Android. (2015). Android Application Fundamentals. Retrieved from http://developer.android.com/guide/components/fundamentals.html. Last visited 24.10.2015.
[44] Social Compare. (2015). Android versions comparison. Retrieved from http://socialcompare.com/en/comparison/android-versions-comparison. Last visited 27.10.2015.
[45] Nielsen, J. 10 Usability Heuristics for User Interface Design. 1995.