

Augmented Annotation of Real-Time Video on a Mobile Phone

Author: Chris Fenlon

Supervisor: Dr. Kenneth Dawson-Howe

A dissertation submitted to the University of Dublin, in partial
fulfilment of the requirements for the degree of M.A.I (St.)

Submitted to the University of Dublin, Trinity College, April, 2014

I, Chris Fenlon, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signature:

Date:

Summary:

This project serves as a proof of concept for a larger application. The application is to be used on a mobile device with a camera, GPS module and mobile data connection. The application would allow a user to point the device's camera at a landmark, building, bus stop or other element of the urban environment and have useful information annotated onto the image on screen. This application would run solely on the mobile device with no back end system involved, using a publicly available database of images with labelled buildings, in the form of Google Street View.

This project examines the use of computer vision techniques to identify buildings in images and matching them between two images. The application takes the input image from the camera and a GPS coordinate as input. The GPS coordinate is used to download a Google Street View image of the current location. Buildings are then searched for in each image and any buildings found are compared. The chosen solution extracts the windows in the buildings present in the images. The layout of these windows is then used to describe the buildings layout.

To achieve this the reliability and accuracy of GPS coordinates from mobile devices and Google Street View are compared. After this the computer vision aspect of the application is considered. A window is defined as a roughly rectangular shape, subjected to perspective distortion. To extract these windows a new corner detector was developed to extract and classify corners of the four types that constitute a rectangle. The window extraction process then searches for groupings of these four corner types, in the correct ordering, that are linked together by edge pixels. These edge pixels mark the outline of the windows. Buildings are then described by the layout of the windows, based on grouping the windows into columns. The columns relative positions to one another is used to define a pattern. This pattern is then searched for in the second image. A score of how well two buildings match is calculated based on the aspect ratio and relative positions of the windows in each pair of matching columns.

The main findings of this report are considered in four sections; the GPS analysis, corner detection, window extraction and matching. The GPS tests indicated a number of potential issues. The GPS coordinates provided by different devices will not match. The GPS coordinates provided by the device may not yield a Google Street View image at the same location. This can generally be rectified by downloading multiple Google Street View images at the same location, but with different orientations. The mobile device's GPS readings are susceptible to spurious outputs. This can be rectified by maintaining average GPS positions over time.

The corner detector method works quite successfully. The extraction method can struggle when there are two lines of edge pixels very close together. The window extraction method requires all four corners to be found and a, mainly, continuous line of edge pixels outlining the window. The method is very successful, but is reliant on results from the previous stage. The matching algorithm for comparing buildings in two separate performs quite well, provided the majority of the windows are found. If too few are found the application struggles to extract enough positional information to produce a reliable score. The biggest issue found was the stitching methods used by Google in creating the images for Google Street View. This stitching has the potential to distort the image and remove details such as windows.

Overall the project succeeded in analysing if the GPS on a mobile device is suitable for this application. It also successfully trialled the techniques above for describing and comparing buildings. The overall application is not implemented here as this project merely sought to prove certain aspects were possible. These have been proved, with some reservations about using Google Street View in its publicly available form.

Acknowledgements:

I would like to thank my supervisor Dr. Kenneth Dawson-Howe for his support, guidance and direction throughout this project.

I would like to thank my family and friends for their support and encouragement throughout the duration of this project. A special word of thanks goes to my mother for all the hours spent proof reading this report.

Contents

Summary:.....	ii
Acknowledgements:.....	iii
1 Introduction	1
1.1 Aims:	1
1.2 Motivation:	2
1.3 Ethical Considerations:.....	4
1.3.1 Data:.....	4
1.3.2 Usage:.....	5
1.3.3 Summary:.....	6
2 Overview:	8
2.1 Overview of the Problem:	8
2.1.1 GPS:.....	8
2.1.2 Descriptive Features:	8
2.1.3 Building Façade Description:.....	8
2.1.4 Matching:.....	9
2.1.5 Summary:.....	9
2.2 Building identification	9
2.2.1 Existing Research:	9
2.2.2 Applicability to application:	12
2.2.3 Algorithm Selection:.....	12
2.2.4 Summary	13
3 Application Description:.....	15
3.1 Introduction:	15
3.2 Inputs:	15
3.3 Device requirements:.....	15
3.4 Application operation:	16
3.4.1 Pre-Processing:.....	17
3.4.2 Window Extraction:.....	17
3.4.3 Building Description:	17
3.4.4 Matching:	17
3.4.5 Annotating:	17
3.5 Conclusion:.....	17
4 Implementation:	18
4.1 GPS:.....	18

4.1.1	Introduction:	18
4.1.2	Google Street View:	18
4.1.3	Mobile Devices:.....	19
4.2	Window Extraction.....	19
4.2.1	Introduction:	19
4.2.2	Window Description:	19
4.2.3	Edge Detection:.....	20
4.2.4	Corner Detection:.....	23
4.2.5	Extracting corners:	23
4.2.6	Window extraction:	29
4.2.7	Conclusion:.....	30
4.3	Façade Description:.....	31
4.3.1	Introduction:	31
4.3.2	Columns of windows:.....	31
4.3.3	Conclusions:	35
4.4	Façade Matching.....	36
4.4.1	Introduction:	36
4.4.2	Matching Columns:	36
4.4.3	Matching Windows within Columns:	39
4.4.4	Overall score:	40
4.4.5	Conclusions:	41
4.5	Summary:.....	41
5	Results and Testing:	43
5.1	GPS.....	43
5.1.1	Test Case One:.....	43
5.1.2	Test Case Two:	45
5.1.3	Conclusions:	52
5.2	Corner Detection:.....	53
5.2.1	Test Case One:.....	53
5.2.2	Test Case Two:	55
5.2.3	Conclusions:	57
5.3	Window Extraction:	57
5.3.1	Test Case One:.....	57
5.3.2	Test Case Two:	58
5.3.3	Conclusions:	60

5.4	Façade Matching:.....	60
5.4.1	Test Case One:.....	60
5.4.2	Test Case Two:	62
5.4.3	Conclusion:.....	66
5.5	Conclusions:	66
5.6	Limitations:	67
6	Final Word:.....	69
6.1	Future Work:.....	69
6.2	Conclusions:	69
7	Bibliography	71
8	Appendix:	72
8.1	Appendix A: Alternative Approaches	72
8.1.1	Vanishing Points:.....	72
8.1.2	Line extraction:	72
8.1.3	Histogram of oriented gradients:.....	75
8.1.4	SIFT:.....	78
8.1.5	FAST Features:.....	78
8.2	Appendix B: Supplementary Results data.....	80
8.2.1	Corner Detector and Window Extractor:	80
8.2.2	GPS Data.....	86

1 Introduction

Computer vision is the branch of computer science dealing with the analysis of images, to extract knowledge. Researchers strive to develop this field to such an extent that simulating the human vision system would be possible, but at present this is nowhere near a realistic goal. While simulating the full human vision system is unrealistic, solving specific problems is a realistic goal. This project aims to solve such a specific problem using computer vision.

1.1 Aims:

This project aims to perform building identification on a mobile device. The identified building will then be annotated with useful information about the buildings function, such as contact details for the occupants, historical information about public landmarks and so forth. To achieve this, the goal is to compare the image taken with the on board camera, with images from a publically available database of images, Google Street View. Once the building has been identified the address will be extracted from Google Street View and this will allow the application to query a directory service or similar service to download information to annotate on the input image.

To illustrate this an example is considered below. Figure 1 below shows a person standing outside the Lloyd building in Trinity College Dublin. They are unfamiliar with the building and need more information about it. Using an application such as the one proposed in this project, they can be presented with an output such as that shown in Figure 2 below.



Figure 1 A person standing outside the Lloyd Building in Trinity College Dublin. They are using their phone to view the building.

The output in Figure 2 below displays the name of the building, some information about the different departments in the building and where they are located. The information is presented superimposed on the building and can contain links to external sites. These external links allow the user to find more in depth information.

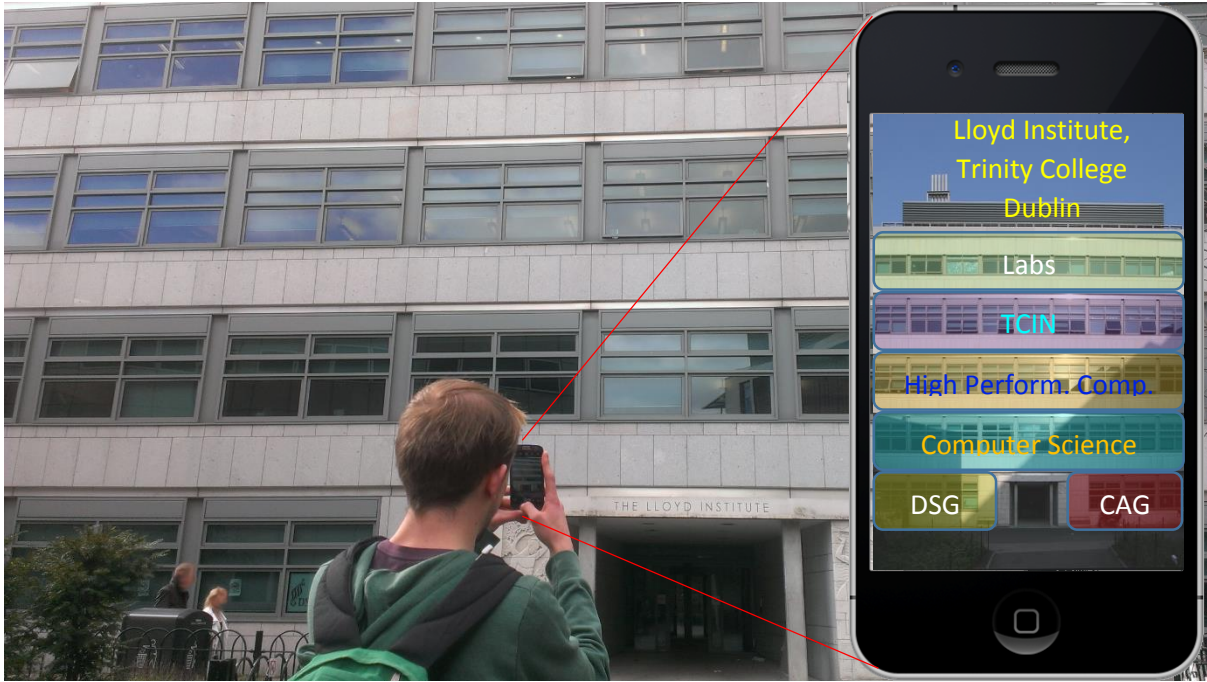


Figure 2 This image provides a view of the mobile device and the output displayed on the screen.

1.2 Motivation:

Today most people have access to mobile devices such as tablet computers and smartphones. These devices have quality cameras, GPS capabilities, internet access and relatively good processing power for their size. The internet contains a large amount of information about the many buildings, landmarks, bus stops, pubs, restaurants, etc. that people see as they traverse an urban environment. People unfamiliar with the environment may constantly need to ask passers-by for this information or search the internet as they walk.

This project examines whether it is possible to present this information in a timely and intuitive manner, using only the camera, GPS and internet connection on a mobile device. Consider if



Figure 3 Sample information for a public establishment

you were a tourist in a city you had never been to, you are walking down a main street in the city and are passing many restaurants and you want to know the menus, the specials, the quality of the food and atmosphere. You could research each one individually online, visit each and read the menu, but what if you could just turn on your camera, on your mobile device and have the opening times, links to menus, reviews, specials, etc. superimposed on the image feed from the camera, such as that shown for a public establishment in Figure 3 above.



Imagine you need to use the public transport system to explore the city, but you don't know which buses, trams or trains go to which destination. You could trawl the internet, tourist information offices and friendly drivers asking for help. Or you could simply look at the bus stop or train station you are currently at through the camera on your mobile device and see the routes serving this stop, when the next bus or train will arrive, where it is going and have links to route maps and the transport operators webpage as shown in Figure 4.

As you traverse the city and visit the many landmarks, it would be nice to learn some of the history of that landmark and its significance. You could carry a city guide book and leaflets from the

Figure 4 A sample of the annotations possible for bus stops
tourist information office, or look it up online, or simply use your mobile device and read the information, superimposed on the landmark in the video stream as in Figure 5 below.

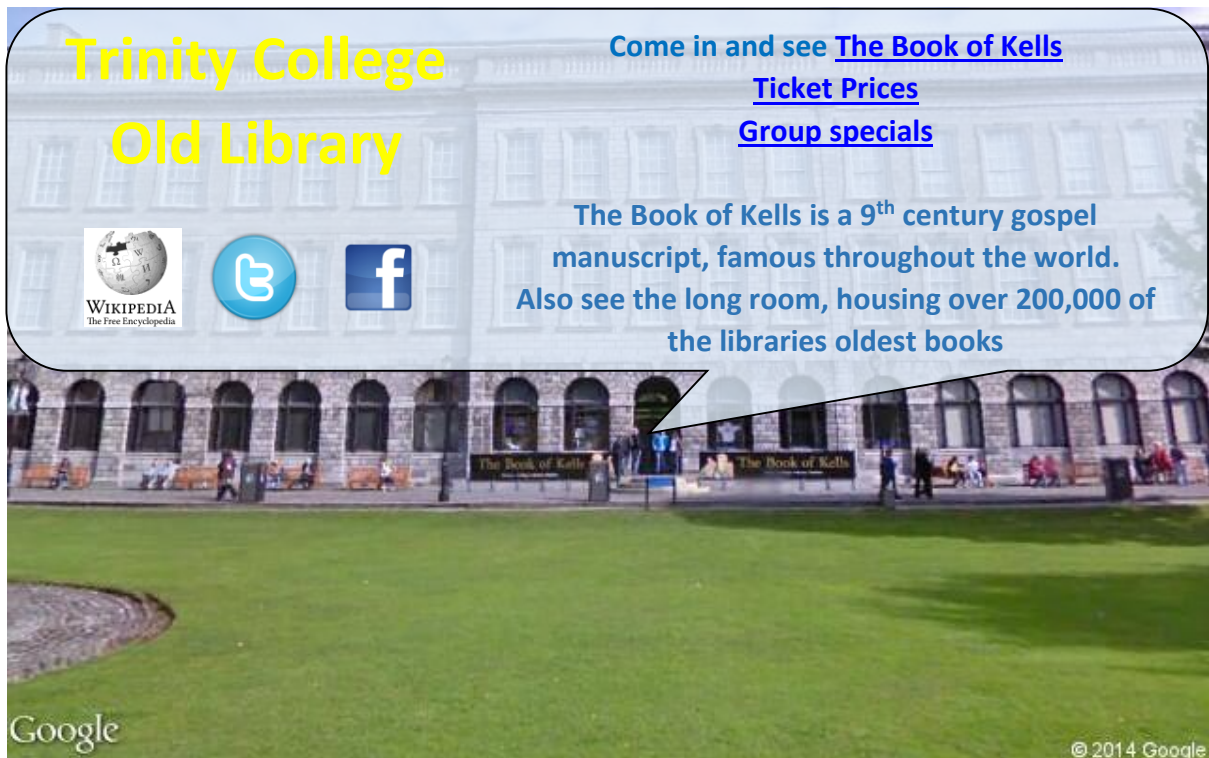


Figure 5 A image of the old library in trinity College Dublin, tourists are attracted to the Book of Kells and the long room.

This allows for a swift and intuitive traversal of the environment. The application provides brief information with links to allow for further reading. The idea being to link the visual with the vast amount of information available online, but to do it in a way that allows a user to only get as much of that information as they wish. This idea of linking the visual with information would be beneficial to many different parties, residents of the city traversing new areas, commuters gathering information about their route home, tourists visiting landmarks, revellers trying to find the best party in town and many more.

1.3 Ethical Considerations:

Research in general needs to be held to certain ethical standards. This includes the way in which experiments are carried out, the reporting of the results and acknowledging the work of others, where appropriate. On top of these considerations are specific ethical issues arising from the nature of the experiments carried out or the application being developed.

When considering this particular research project, there are a few key areas where ethical considerations need to be made. These key areas are; data collection of sample inputs for testing purposes and the ways in which the application could be used.

1.3.1 Data:

Data collection is a key part of the experimental and testing aspect of this project. It is necessary to allow for the reporting of how successful the techniques and algorithms presented, perform with real data. When collecting data, photographs and videos are recorded in public locations. While this is necessary to give examples of the type of inputs the application would expect, there is also the potential to record people at the same time. This becomes an ethical issue if the person can be easily identified by some visual features (such as their face) being present in the image. The storage of this data becomes subject to legal standards as this can be classified as personally identifiable information (PII). There are two ways of handling this, the first is to store the

data in a secure manner which meets the required legal standard, and the second is to edit the images in such a way that the PII features have been removed.

The first option presented is complex and would require a lot of time to first understand the required standards and then to implement them. In the second option, when altering the image in any way, it

needs to be considered if this will make the gathered data unsuitable for use in testing. Removal of the PII is achieved through blurring the image. The main PII that will be present in the data are people's faces, car licence plates and features of a similar



Figure 6 Google Street View image containing people and cars, note how the faces and licence plates have been blurred.

nature. Considering the application seeks to extract information about buildings, and the size of faces and licence plates in comparison to buildings, this blurring should not impact negatively on the performance of the application. This means the data with PII blurred out, will still be suitable for testing, as it remains representative of real world input the application would expect. This is the strategy employed by Google Street View and can be seen in Figure 6 above, in the image it can be seen that the licence plate of the car on the left has been blurred out and the people on the footpath have had their faces blurred. Thus the second option is less complex than option one and is therefore the adopted approach for this research.

1.3.2 Usage:

When this application is operational it is envisaged it will display information about the building(s) in view. This will include details about the occupants and their contact information, be they corporate or individual occupants. This information is publicly available through directory and internet services such as the Golden Pages. Thus there is nothing ethically wrong in gathering this information and presenting it to the public, it is after all publicly available. While the presentation of information in this new manner has many advantages, it should be considered if it could have negative implications. Some of the advantages and disadvantages of such an application are discussed below.

This application aims to gather information about the buildings and landmarks in view and present this in a timely and intuitive manner to the end user. This type of technology would useful to

a multitude of different user, from commuters examining timetables, to tourists exploring the city. This information is readily available online, but can be difficult to access while on the move. Another benefit is that companies have a lot of information they make available to the public, but actually getting the information to the right people at the right time can be difficult. An application such as this, allows a new route for companies to provide information to potential clients and customers.

The gathering of this information and displaying it in this manner is open to potential abuse. The possibility exists for this information to be used by opportunistic thieves to achieve their goals. Presenting the name and phone numbers of the occupants, as shown in *Figure 7* below, would allow



Figure 7 A mock-up of application output, using typical information found in a directory service a potential thief to try knocking on the door as well as phoning the residence to establish if it is empty or not. If the names could be used to find the occupants social media pages, this could be used to establish if the resident is away from home. Thus helping the thief establish if the residence is occupied or not.

Another consideration is the invasion of a person's privacy. All of this information is publicly available and seems innocuous in isolation. The question is whether gathering all of this information together creates too substantive a profile of the individual involved? Linking all of these pieces of data together does not violate any laws, but is it socially acceptable to generate such a profile?

It is important to raise the issues highlighted above, but they would not deter from building such an application. The issue is comparable to the ethics of building a computer, computers have greatly enhanced the lives of millions of people the world over. Though a small proportion of people use this technology in an objectionable manner, the overwhelming positive effect of computers would mean it is ethical to build a computer. The ethical issue is with the user abusing the computer, not the manufacturer for building it. A similar conclusion can be attributed to this application. It has the potential to be used in a positive manner to greatly enhance a user's experience of traversing the urban environment, especially in an unfamiliar environment. Issues such as those raised above need to be highlighted and considered, but would not lead to the abandonment of this technology.

1.3.3 Summary:

After considering the above issues, it is clear that some ethical issues need to be addressed when implementing such an application. The main concern surrounds the types of data gathered;

data collected and stored for testing purposes and data collected and displayed to the user. The issues surrounding the collected experimental data is that it may contain information which could personally identify an individual. To address this issue the collected data has all faces, licence plates and any other PII features blurred, as shown in Figure 6. The other potential issue is how this application could potentially be used. The application itself is not unethical, it only gathers publicly available data and its goal is to improve the user's experience of traversing an urban environment. While it is important to highlight such risks, it is also important to note that they should not stop the development of such an application. The majority of users of such an application would greatly benefit from using it as intended.

2 Overview:

2.1 Overview of the Problem:

This section of the report seeks to establish an overview of the main problems which need to be overcome in developing a solution for the proposed application. This section should provide a high level “road-map” of the project, with the aim of clarifying how different aspects of this project are linked together. While each section of this report is considered individually, it is important to remember the high level application and the requirements of that section. Considering each section with the broader application in mind, will allow assumptions to be made to simplify the processing techniques employed.

The main stages involved in the applications operation are listed below;

- Extract the GPS coordinates of the mobile device and download the corresponding Google Street View Image.
- Identify features to describe the buildings in the images.
- Design a method of describing a building façade using these features.
- Compare two images for similar buildings.

A brief overview of the purpose of each step is provided below.

2.1.1 GPS:

The application aims to compare an image from a mobile devices camera and the Google Street View image at that location. To download the Google Street View image the application requires a GPS coordinate.

There are a number of anticipated issues that may arise with regards the GPS coordinates and Google Street View. The first is inaccuracies in the GPS coordinates from mobile devices. Mobile devices are built with very different specifications and the same can be said about the GPS modules within the devices. This means it is not reasonable to expect to get the same GPS coordinates from two different devices used at the same location. There are also issues with regards Google Street Views data collection method. As the images are taken from on top of a car, the result is that images provided will all be at a high elevation and also from the middle of the road.

For the purposes of this project, these two issue will be examined. The goal is to get images that match to a satisfactory level to allow the application to identify the same building in the two images and match them.

2.1.2 Descriptive Features:

To describe what a building looks like the application will need certain features, which it can extract and then use to match two buildings across two different images. The word features is a very broad term, for this application this simply means some form of characteristic of the building which can be described in a unique manner. There are a number of options available for this, some of the standard approaches would be the use of a feature detector like SIFT, template matching, colour based matching, shape based matching and many others. In latter sections a number of these are discussed and one method chosen.

2.1.3 Building Façade Description:

Once the application has identified a number of characteristics of the building façade(s) in the image, a description of the entire façade is needed. This needs to contain information not just about the features of the building found, but how the building can be described when all of these features are grouped together to form a building façade.

2.1.4 Matching:

Up to this stage the two images have been processed in isolation, now the two images are compared with each other. The descriptions of the buildings contained within the images is used for this comparison. The application examines the description in one image and tries to find a similar description in the other. This comparison yields a score based on how well the two descriptions match and this score is used to indicate if the buildings match.

2.1.5 Summary:

In this section a brief overview of the main sections of the application considered in this project are presented. The four main sections are; GPS extraction, finding descriptive features, façade description and matching across multiple images.

The application extracts the GPS coordinates of the mobile devices location. This is used to download a Google Street View image for the location. The Google Street View image and input image from the camera are then processed individually before being compared. The descriptive features are extracted and combined to form a description of the buildings in the images. The two images are then compared, based on these descriptions.

2.2 Building identification

In this section a number of examples of previously trialled building identification techniques are presented. Through examining these and considering their applicability to this application, a number of possible approaches will be identified. Once these approaches have been identified they are evaluated and a suitable algorithm chosen for implementation in this project.

2.2.1 Existing Research:

There are many published works in the computer vision community which involve processing building façades. The majority of these works have focused on augmented reality navigation. This involves superimposing navigation supports, such as arrows, into a live video stream. As these applications intent to augment information onto the buildings they extract planar information about the buildings (Chen, et al., 2011) (McClellan & McDonald, 2013) (Robertson & Cipolla, 2004).

2.2.1.1 *Extracting building façades:*

The general approach taken by these authors is to find vanishing points associated with the plane(s) of the buildings in the image. The idea being to use these vanishing points to establish a coordinate system for the plane, which can then be aligned with the plane of the camera. This transformation from image plane to building plane is described with a matrix transformation, called a homography. A diagram illustrating how vanishing points are calculated and what they represent is shown below in Figure 8 below.

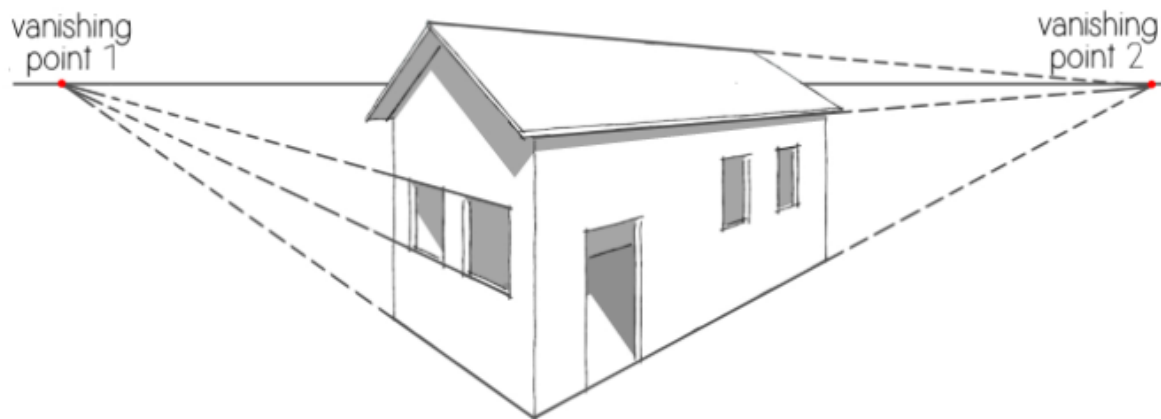


Figure 8 This image demonstrates how the horizontal vanishing points are extracted. The two facades for the building shown have individual horizontal vanishing points. A similar vertical vanishing point exists for the building. This is found by extending the vertical lines in a similar way to the horizontal lines shown. (Heaston, 2014)

(McClellan & McDonald, 2013) describe the process of extracting the homography in five stages;

1. Line segment extraction: the image is processed to extract line segments present in the image. These correspond to edges in the image such as those outlining the building, windows and other shapes of the façade.
2. Tilt rectification: when a user takes a photo with a camera the camera is usually tilted upwards. This stage removes this tilt, so as to give an image where the camera is looking straight on at the building.
3. Parallel line grouping: before examining the lines and finding vanishing points, they must be grouped together. Parallel lines are grouped together, as these groupings will yield the vanishing points.
4. Layout extraction: it is in this stage that the vanishing points are found. Once they are found, they are used to assign directions to sub regions within the image. It is from this that each façade within the building can be extracted.
5. Homography estimation: with the facades extracted the homography matrix can be estimated. This is achieved by calculating the transformation matrix between the façade plane and image plane. This transformation calculates the rotation and translations which transform points on the image plane onto the façade plane.

(Robertson & Cipolla, 2004) used a simpler version of this which is illustrated in Figure 9 below. In this diagram the homography matrix has been applied to the original image. This transformation converts the image so it appears as if the photo was taken from directly in front of the building discovered. While (McClellan & McDonald, 2013) developed a system which could identify multiple facades within an image, (Robertson & Cipolla, 2004) assume there is one dominant planar surface.

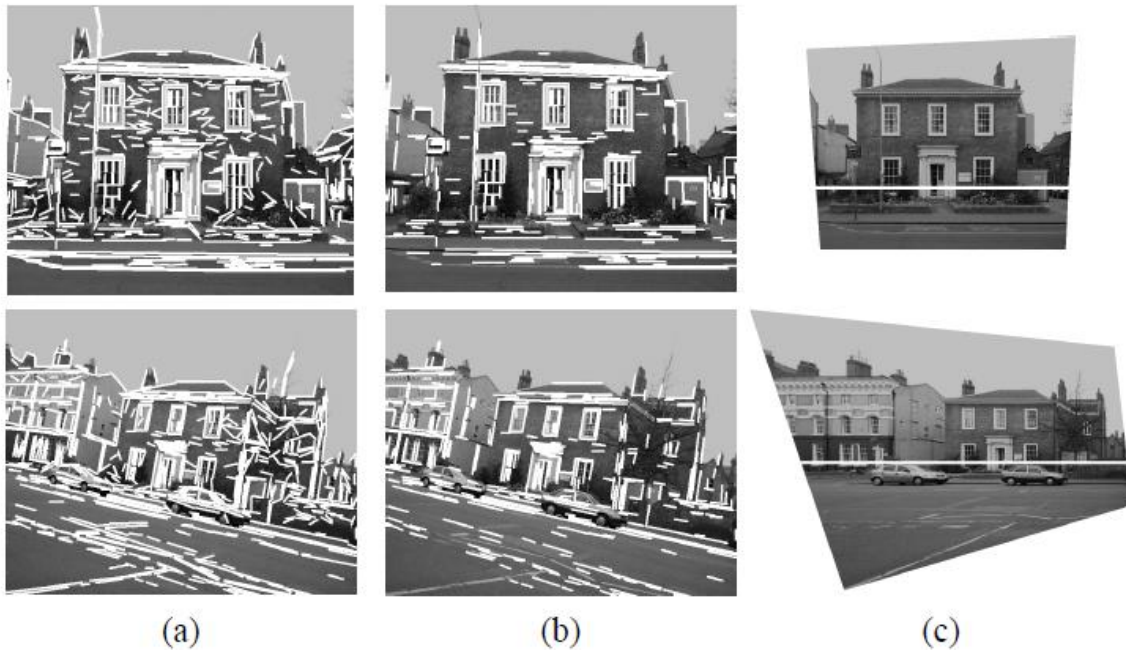


Figure 9 The leftmost images (a) show the original image with all the detected lines. The middle images (b) show only the lines which are associated with the horizontal and vertical vanishing points. The rightmost images (c) show the transformed output, where the homography matrix has been applied to the image. This transforms the image as if it were taken from a camera directly in front of the building.

2.2.1.2 Database:

The systems proposed by (McClean & McDonald, 2013) and (Robertson & Cipolla, 2004) all use a database of images of the urban environment to compare a sample image against. This requires a huge overhead on the part of the researchers as they must gather all this information, create the database and in some cases, pre-process all of this data before it enters the database.

(Chen, et al., 2011) discuss in detail how such a system is constructed. They talk about the equipment needed for the gathering of data, which includes LIDAR, panoramic cameras, GPS and inertial measurement unit (IMU). They discuss how the panoramic view collected must be transformed to perspective views. These views are then processed to make future comparisons easier and then added to the database. To create a database for the city of San Francisco, 1.7 million perspective pictures were generated.

(McClean & McDonald, 2013) present a novel method of constructing the database. Their system analyses new data presented as input and searches for matches within the database. If no matches are found the plane in the new input image is added to the database. The novel aspect is that if the new plane overlaps with one plane in the database, the two are connected to form a larger plane in the database. If the input plane matches two different database planes then the three planes are joined together to form one large plane in the database.

2.2.1.3 Matching across multiple images:

In all three projects presented the authors compare the input image with images from the database. (Robertson & Cipolla, 2004) find Harris features (Harris & Stephens, 1988) at multiple scales and compare features in one scale with features in all other scales to find the best match. Multiple scales refers to different levels of blurring applied to the image, to simulate the image being taken with a camera at a position further away from the building.

(McClean & McDonald, 2013) use a bag-of-words (Galvez-Lopez & Tardos, 2011) technique to match two images together. The bag-of-words technique divides the image into a number of

different features. A histogram is then created with one location for each feature, this histogram is then filled with values equal to the number of occurrences of each feature. This histogram is calculated for each image and the comparison is made on these histograms, the closer they match, the higher the probability of the two images containing the same building. (Chen, et al., 2011) use SIFT (Lowe, Object recognition from local scale-invariant features, 1999) features to describe the buildings. These descriptors are then matched across the two images to find matching buildings. They also discuss the how the database is searched in order to find images to compare with the input image. The search space is reduced by using the GPS coordinates of the input image.

2.2.1.4 Analysis:

All of the systems described above do not perform any of the image processing or matching on the mobile device. In all of the above examples an image is taken on the mobile device and then sent to a remote server. This server then performs all of the image processing and matching with images from the database. The result is then prepared on the server and returned to the mobile device for display.

The systems presented above also use their own database of images to compare the input image with. This database contains images which have been pre-processed to aid performance and accuracy.

As a result the techniques presented above represent a good starting point for this application, but may not be suitable. They may not be suitable as the application will run all of the image processing on the mobile device and use a publically available database of images, Google Street View. These images have no pre-processing applied specifically for use with this application.

2.2.2 Applicability to application:

For this project, applying transformations to the information which is superimposed on the building is not necessary. This assumption can be made based on the screen size of a mobile device. If the application is presenting text based information on a small screen, having it distorted would make it very hard to read. This means extracting the planar information for transforming the output onto the plane of the building façade is not necessary.

This application seeks to make use of Google Street View as the database of images. This will reduce performance when compared to the research presented. This is a result of the data in the database not having any pre-processing applied to aid the performance of this application. The application will perform all processing on the mobile device. This may have a limiting effect on the complexity of the algorithms that can be used.

2.2.3 Algorithm Selection:

As a starting point, a vanishing point methodology similar to that presented by (McClean & McDonald, 2013) was trialled. The results and a discussion on this trial is presented in Appendix A: Alternative Approaches. This method was not successful as the line segments extracted were not suitable for finding the vanishing points. A sample of the results are show below in Figure 10 below, the issue is the lack of lines generated by the tops and bottoms of the windows. The other problem are the noisy lines generated at the footpaths edge. The end result was it was not feasible to extract the vanishing points for the building façade.

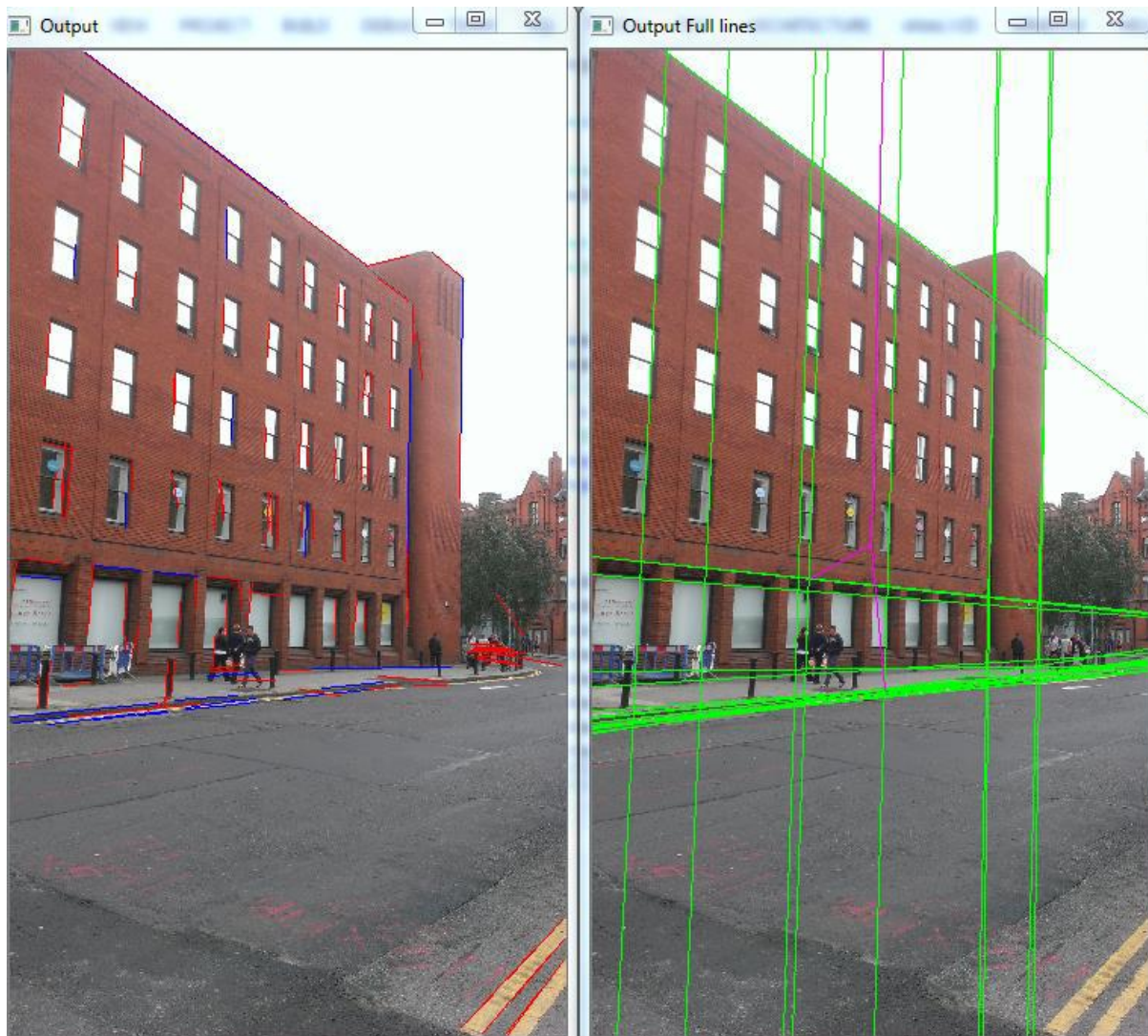


Figure 10 This image shows the lines extracted, left, and the lines used for vanishing point detection, right.

A number of other approaches were considered, for example using a regular feature detector such as SIFT (Lowe, Object recognition from local scale-invariant features, 1999), or FAST (Rosten & Drummond, Machine Learning for High-Speed Corner Detection, 2006) features. These methods proved unsuccessful and again are described in Appendix A: Alternative Approaches.

At this point a new approach was adopted. This approach has been implemented and is presented in this report. The method seeks to extract the windows within the image. Then extract the pattern of the windows and use this to describe the building. This approach was conceived after relative success in trialling the FAST feature detector, see Appendix A: Alternative Approaches for more details. This description is then used to compare buildings in multiple images.

2.2.4 Summary

In the above discussions a number of previous works are discussed. Each of these have trialled varying approaches to building identification. A common theme amongst these works is the use of purpose built databases, with back end systems performing the image processing. This is where the application under consideration differs from those presented above. This application will use Google Street View as its database and all processing will be performed on the mobile device.

Through experimentation it was found that the vanishing point approach was not going to work. A number of other techniques were trialed and are discussed in more detail in Appendix A: Alternative Approaches. The chosen method extracts the windows in the image and bases the description of the building on their layout. A more detailed explanation of the chosen approach follows in the next section.

3 Application Description:

3.1 Introduction:

In this chapter the overall application is described. This project will act as a proof of concept of this application. The inputs the application requires and the demands this places on the mobile device are discussed. The application itself is then discussed, with the five main stages briefly introduced. These stages are; pre-processing, windows extraction, building description, matching and annotation. Each stage builds on the previous stage and how this progression from stage to stage is discussed.

3.2 Inputs:

The application requires only three inputs, an image from a camera on the device, a GPS location and an orientation. To start identifying the building two images are required, one from the camera and one from Google Street View. The GPS and orientation information are used to download the correct Google Street View image.

3.3 Device requirements:

For the application to successfully operate there are a number of demands placed on the device to be used. The four main elements required are a standard camera, a GPS receiver and a quality internet connection.

The device camera will provide the main image with which the Google Street View images will be compared. Google use high quality cameras when acquiring images for their Street View product. As identifying buildings will require the extraction of identifying features, the quality of the two images must be somewhat similar to allow for the extraction of similar features from both images. This relates to the resolution of the camera on the mobile device.

The mobile device will need a GPS receiver that is accurate to within a few meters. This is essential as to download one or more images from Google Street View requires a longitude, latitude and orientation. It will also improve performance if the extraction of orientation information is possible. While it is possible to download a number of images at the same location that would encompass a 360° range, processing these extra images would affect performance. To counteract devices with poor GPS hardware, a number of extra images could be downloaded to examine locations “near” the presented GPS coordinates.

The mobile device will require a reliable internet connection as it will be necessary when downloading pictures from Google Street View. The images are downloaded one at a time, but depending on how accurate the GPS coordinates are a large number of image may need to be downloaded to achieve a satisfactory result. To reduce the bandwidth used, smaller images could be downloaded, but this will decrease the resolution of the image, which limits the features that can be extracted. Thus reducing image size produces a trade-off between speed and accuracy.

3.4 Application operation:

The application has a number of different stages, each of which build on the output of the last. The process starts by taking the inputs and processing the resulting images. These images are then compared to find matching building(s). If a match is found the input image is annotated and displayed to the user. If no match is found the GPS coordinates or orientation are adjusted and the process starts again.

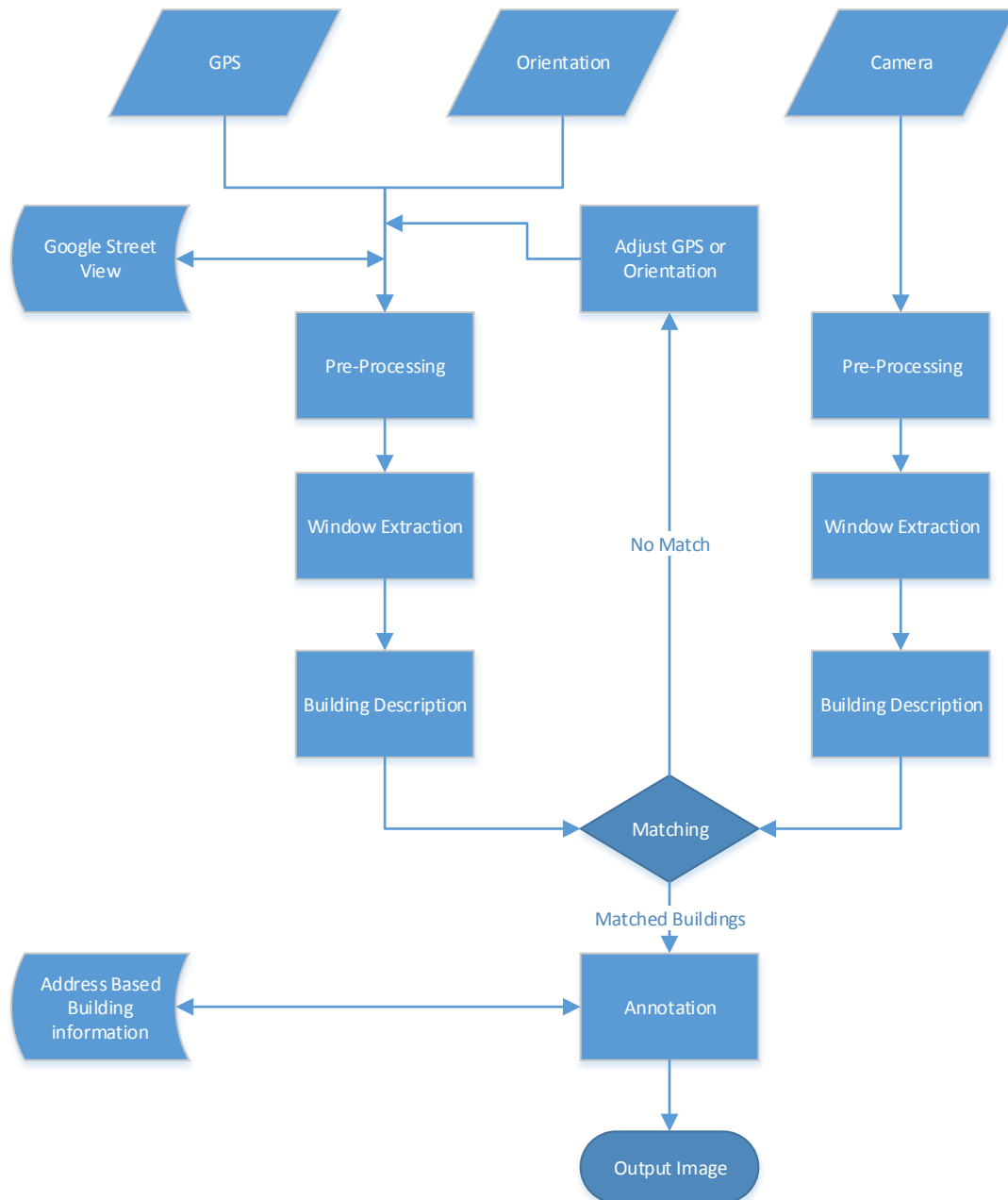


Figure 11 System diagram, note the two external information sources required and the iterative process which adjusts the GPS and/or orientation information.

The application diagram, shown in Figure 11 above, illustrates the different stages involved and how the application adjusts the GPS and/or orientation information to allow for inaccuracies in the input data. Below the main functionality of each stage is described.

3.4.1 Pre-Processing:

The input images are subjected to a number of techniques to generate suitable inputs for the window extraction stage. The input image is converted to greyscale and subjected to a 2-D edge detector. This produces an edge image which contains the outline of the windows. This is used as input to the window extraction process.

3.4.2 Window Extraction:

The window extraction process follows on from the pre-processing and uses the edge image produced by that stage. This stage has two main parts; corner detection and window extraction. The first part finds the corners which form part of the definition of a window. The window extraction is then performed using the detected corners and edge image.

3.4.3 Building Description:

Before two images can be compared, the information about any buildings in the image must be described. The description must be designed in such a way as to allow comparisons across multiple images, captured in a variety of different conditions. The description of the building is based on the windows discovered, their relative size and their position relative to the other windows.

3.4.4 Matching:

Once the two input images have been processed and described, the matching process can proceed. The Building description stage builds a description of how the buildings windows are laid out, both in terms of size and relative position. Comparing two images involves examining how similar the two descriptors are, and calculating a score based on this. This needs to be able to handle missing windows in one image and extra windows in another image.

3.4.5 Annotating:

Once the buildings have been matched, the address needs to be extracted from Google Street View. Using this address a directory service will be queried to extract information about the occupants of the building, or the significance of the building. This information is then parsed and annotated onto the input image. This input image will then be displayed to the user.

3.5 Conclusion:

The above section briefly introduces how the entire application operates and the different stages involved. Each stages takes the output of the previous as its input. The application takes only three inputs; an image, a GPS coordinate and an orientation. Based on this a Google Street View image is downloaded. The goal is then to compare buildings in the camera image and the Google Street View image. Before comparing these images a description of the buildings present is generated. This is based on the windows which make up the building's facade, using the size of the windows and their relative positions. Based on this description the comparisons can be made and a score calculated to indicate how well the buildings in the images match. If a match is found, information about that building is downloaded from a directory service, parsed and annotated onto the input camera image. This is image then displayed on the mobile device for the user.

4 Implementation:

This section provides a detailed and in depth description of the implemented application. There are four main parts and each are presented individually. These are; GPS coordinates, window extraction, façade description and matching. In each section the potential issues are discussed along with a detailed explanation of the different stages involved. This is accompanied with an in depth explanation, with examples, of how the different stages achieve their goals.

4.1 GPS:

4.1.1 Introduction:

The application requires a Google Street View image to compare with the current camera image. To download this image a GPS coordinate is required, this is read from the GPS device on board the mobile device. An accurate GPS coordinate allows for an image to be downloaded from Google Street View, which will contain the building in the input camera image. Inaccurate GPS coordinates will result in the application having to adjust the GPS coordinates a number of times, until a suitable image is downloaded. This requires excessive bandwidth as extra data is downloaded and will also reduce performance as more images must be processed.

In this section the GPS devices on a number of mobile devices are tested. The aim is to evaluate the accuracy of the GPS devices and that of Google Street View and highlight any issues that may arise due to inaccuracies.

4.1.2 Google Street View:

This short section aims to briefly introduce how Google create the Street View product and highlight some of the issues arising from the data collection method.

Google use cars similar to the one pictured in Figure 12, right. The car uses GPS systems, cameras, lasers, accelerometers and other sensors to gather the data for the Street View service. The GPS coordinates, speed and direction of the car are used to calculate where each image was taken and create the online database that is accessible via the google maps service or via a URL. When the images are taken, the following image has a small region of overlap with the previous image. This then allows Google to “stitch” the images together, after post processing Google claim the result is a “smooth transition” (Google, 2014).

The URL access method allows a user to specify a locations longitude, latitude, field of view, heading, pitch and the size of the downloaded image. These field can be seen in the sample URL given below;

```
http://maps.googleapis.com/maps/api/streetview?size=800x800&location=53.33783,%20-6.5385&fov=60&heading=345&pitch=10&sensor=false
```

The manner in which Google gathers this data leaves the potential for errors to be introduced in the correlation between image location and GPS coordinate. Another issue is that Google take pictures at regular intervals as they drive along, thus in reality the images are taken at



Figure 12 (Google, 2014) An example of a car used by Google to record images for their Street View service. Note the laser sensors and panoramic cameras and the height of the camera

finite locations. This means any image viewed in between these locations is an approximation, which further introduces the potential for errors. This also applies when querying Google Street View for an image at a specific location, the image returned will be in a different position. This is because Google Street View will return the image taken at the location the vehicle took a photo from, which is closest to the given coordinates. This instantly means a photo taken from a footpath will not match perfectly to the Google Street View image, which is taken from the middle of the road.

4.1.3 Mobile Devices:

Mobile devices come in many shapes, sizes and, most importantly, specifications. The key point is that various devices use a diverse range of GPS chips and will perform differently under similar circumstances. These differences will be illustrated in field testing, discussed later. The GPS device on a phone needs time to initialise, as it connects to more satellites. This means readings taken immediately after the GPS sensor is activated will be unreliable.

For this project it is necessary to ascertain if the accuracy of the GPS on a range of mobile devices is suitable for this application. The accuracy need not be absolutely perfect, but it will need to be accurate enough to allow for the gathering of suitable images from Google Street View.

4.2 Window Extraction

4.2.1 Introduction:

This section of the application takes in an image and extracts the windows that are present in the buildings in the image. The window extraction process involves a number of steps. The image is first processed to extract single pixel wide edges. After extracting the edges, the application searches for the four different types of corners that define a window. With all occurrences of these corners labelled according to which of the four types of corners they represent, the process of searching for actual windows can then occur. To find a single window the application searches for a group of corners which includes one of each type of corner, in the correct order/pattern, that are linked together by an edge. This edge would correspond to the top, bottom or sides of the window.

4.2.2 Window Description:

Before examining how the windows are extracted, the definition of a window needs to be discussed. In the modern urban environment windows come in many shapes and sizes, some of which are illustrated in Figure 13.

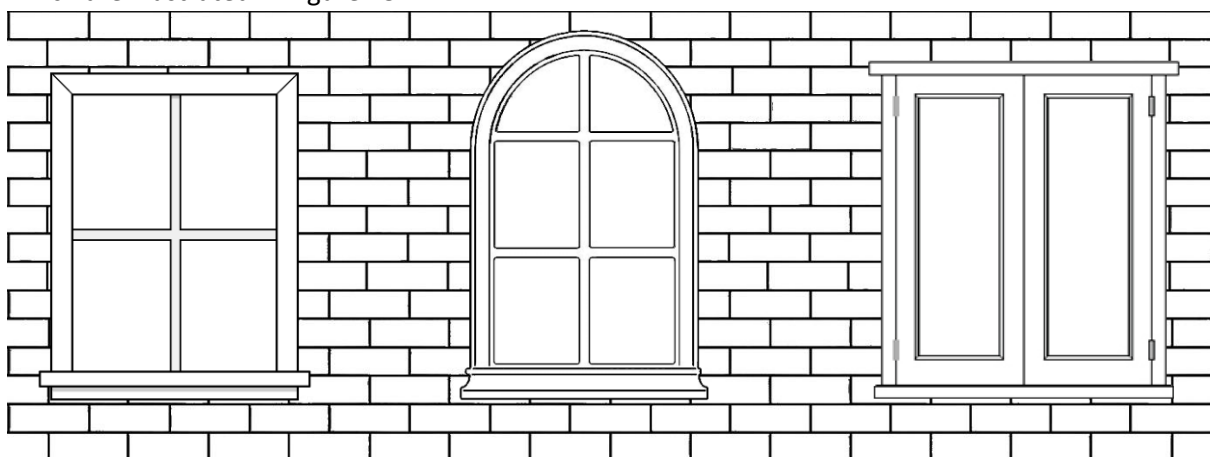


Figure 13 An illustration of a selection of different types of windows that may be found in the modern urban environment.

To allow for the windows to be extracted programmatically, their shape needs to be precisely described. Examining the windows above it is clear that there is a lot of variation in the shape of a window. Some windows have a rectangular outline, some have curved outlines and most

have added features around the edges, such as window sills. The internal window shape also has a lot of variation, with the window panes broken up by vertical and horizontal partitions.

For this project, with given time constraints, the description of a windows shape has been made quite simple. The window will be described as a roughly rectangular shape under the following conditions:

- Four, roughly ninety degree, corners. The condition is not strictly ninety degrees as to allow for perspective distortion.
- Each of the four corners are of a different shape, i.e. one top left corner, one top right corner, one bottom left corner and one bottom right corner.
- The corners must be connected by the outline of the window.
- The corners must be correctly located relative to one another.

This description means that of the three windows presented in Figure 13 above, one of the windows would not be extracted. This is because the arched shape will fail the test of having four corners, which create a rectangle. A more complex description of windows would allow for more detailed matching of windows in different images. But this is left as future work, as this project seeks to act as a proof of concept on the larger application.

4.2.3 Edge Detection:

Edge points occur at locations with a sudden change in brightness, and processing is generally performed on a single channel greyscale image, as opposed to a multi-channel coloured image. As edges represent areas of sudden change, they are considered as 2-D derivatives (rates of change). Edges contain two pieces of information, a gradient and an orientation. The gradient represents the rate of change and the orientation represents the direction in which the largest growth occurs or the direction orthogonal to this.

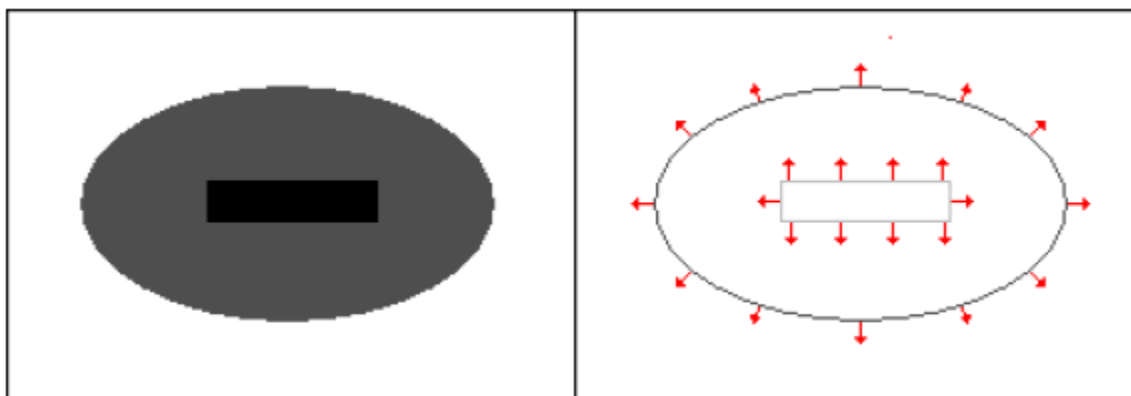


Figure 14 The original image is shown, left, with the gradient and orientation information shown, right. The gradient is stronger on the white to dark grey transition than on the dark grey to black transition. (Dawson-Howe, 2012)

The diagram above shows an idealised edge detection, where the resulting edges are exactly one pixel wide and are located precisely on the edge of the shape. This is because the specific example is computer generated. When performing edge detection on real world images the edges detected will be more than one pixel wide. This is a result of the colour changing gradually over a number of pixels as opposed to the single pixel boundary seen in Figure 14. This means the extracted gradients will need to be post-processed to extract a single pixel wide edge image, with these edges correctly located. This is illustrated in Figure 15 below.

When reducing the initial edge image to single pixel wide edges, two techniques are used; thresholding and non-maxima suppression. Thresholding involves iterating through each pixel in the

edge image and examining its value. The pixels value is set to zero if its gradient value is less than a predefined threshold, according to the following equation;

$$pixel(i,j) = \begin{cases} pixel(i,j) & , pixel(i,j) > threshold \\ 0 & , pixel(i,j) < threshold \end{cases}$$

This removes faint edges such as those edges due to the brick work in Figure 15. This should leave only the strong edge responses, which will be further reduced in size by non-maxima suppression.

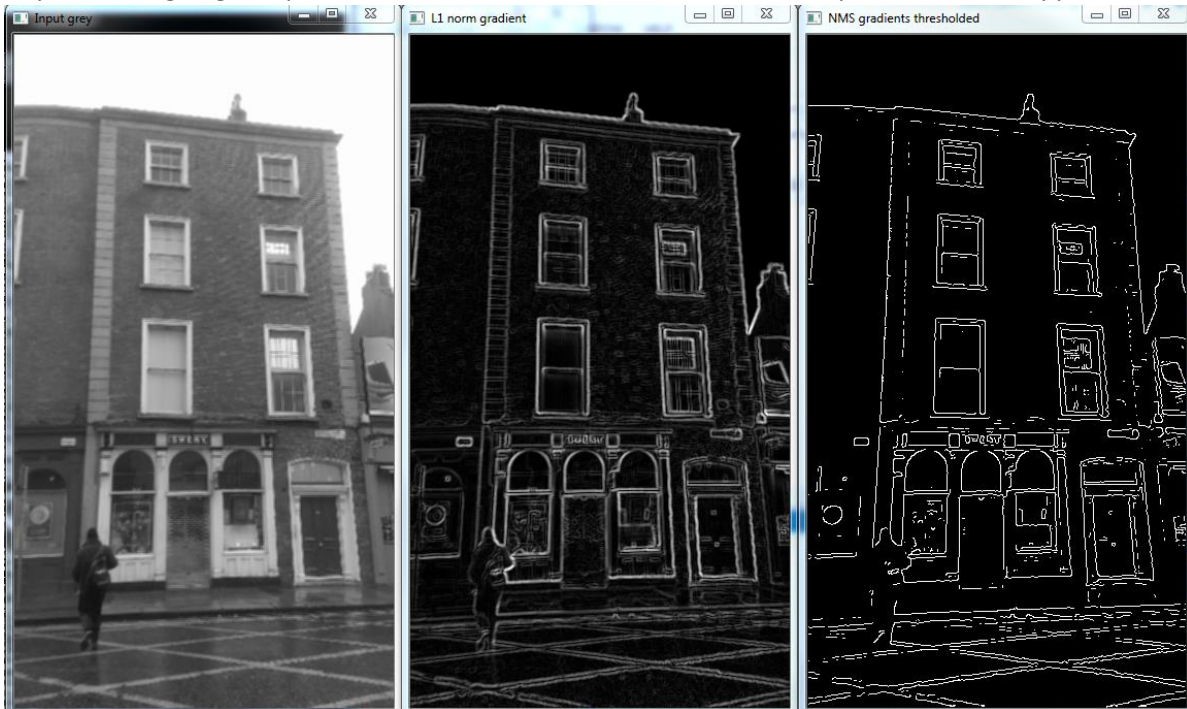


Figure 15 The three images above show the edge extraction process in full. Edges are searched for in a greyscale image (left). The edges found (middle) are not noisy and wide. These edges are subjected to thresholding and non-maxima suppression to extract single width edge pixels (right)

Non-maxima suppression is a widely used technique in computer vision. It is used when the processing applied to the image highlights regions of interest instead of a single point. This roughly equates to trying to find the peak in a surface. Non-maxima suppression works by iterating through each pixel in the image and examining the values of the eight pixels surrounding the current pixel. This is shown in Figure 16 below.

	0	1	2	
	7	X	3	
	6	5	4	

Figure 16 Non-maxima suppression examines the eight pixels surrounding the current pixel, X.

If the current pixel has the largest value compared to its eight neighbours, it is left unchanged, if it is not the largest value amongst the eight neighbours it is set to zero. Figure 17 below, shows an

example of non-maxima suppression. As the application iterates through each pixel the values around the central value of ten are all suppressed to zero. The issue now becomes how this can be applied to extract a line, with a width of one pixel, as opposed to a single point. To use non-maxima suppression to find a line, the application needs to examine the pixels only on both sides, and not those surrounding, the current pixel. This is where the orientation information becomes necessary.

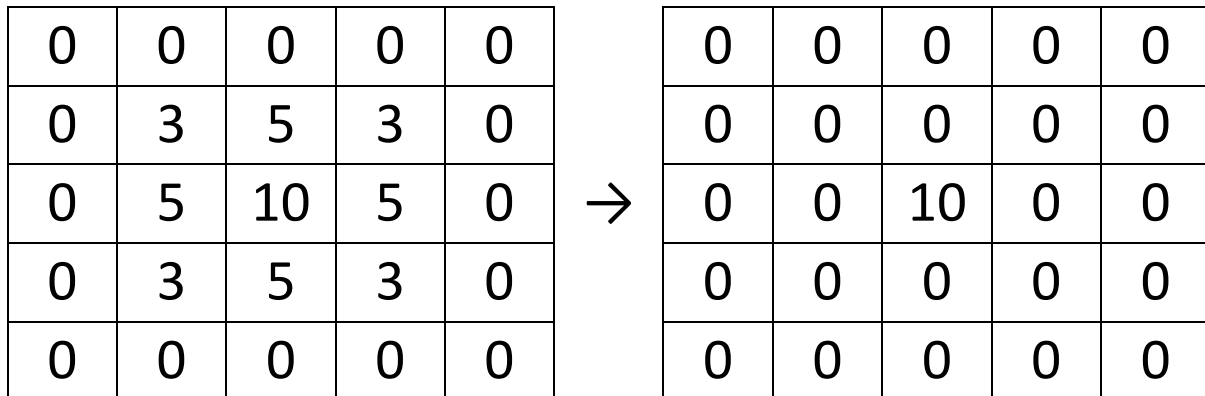


Figure 17 An example of non-maxima suppression

Using non-maxima suppression, examining only the pixels that are perpendicular to the line, will give an edge line with a width of one pixel. The orientation of the edge at the pixel under consideration is used to decide which pixels values should be compared with the current pixel. For example in Figure 18 below, the pixel with a value of thirteen will have an orientation going upwards. This means when applying non-maxima suppression, the pixel above and the pixel below are compared with that pixels value, i.e. using the pixel in the direction of the orientation and the pixel in the opposite direction.

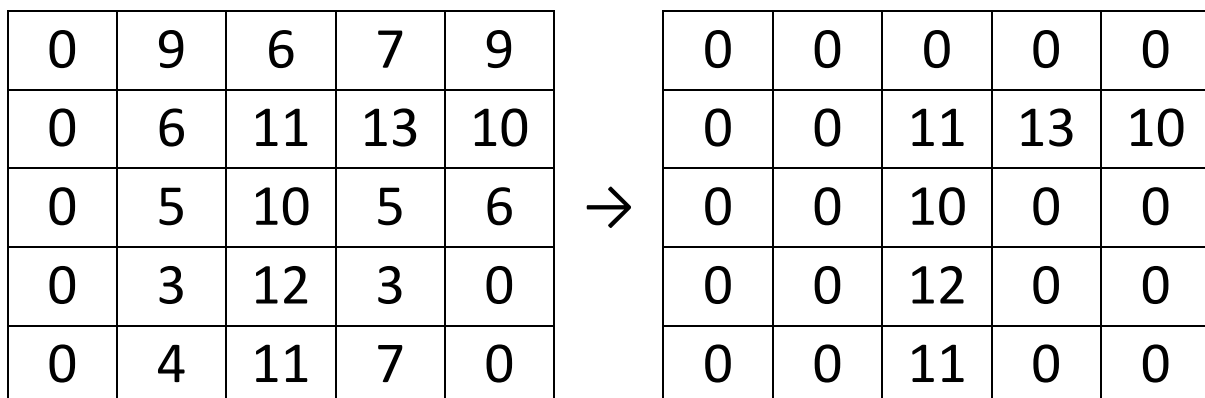


Figure 18 An example of non-maxima suppression applied to a line, in this case the pixels perpendicular to the line need to be examined. These are the pixels in the direction of the orientation of the edge.

In this section the edge extraction process has been discussed. The edges are extracted as they will be used in the following sections to find windows. The edge extraction process has a number of steps in order for the result to be single width edge pixels, representing strong edges found in the image. Firstly a 2-D edge detector is used to extract gradient and orientation information for each pixel in the image. These gradients are then subjected to thresholding to remove weak responses. This is then followed by non-maxima suppression, which reduces the width of the edges to a single pixel. After this processing, the result is the image on the far right in Figure 15 above.

4.2.4 Corner Detection:

The window shape the application is seeking to extract is defined as a rectangle with four specific types of corners. The previous stage extracted the edges of the windows, thus giving an outline of the window. The next stage in processing the image is to search the edge image for corners that could correspond to one of the four corners of a window.

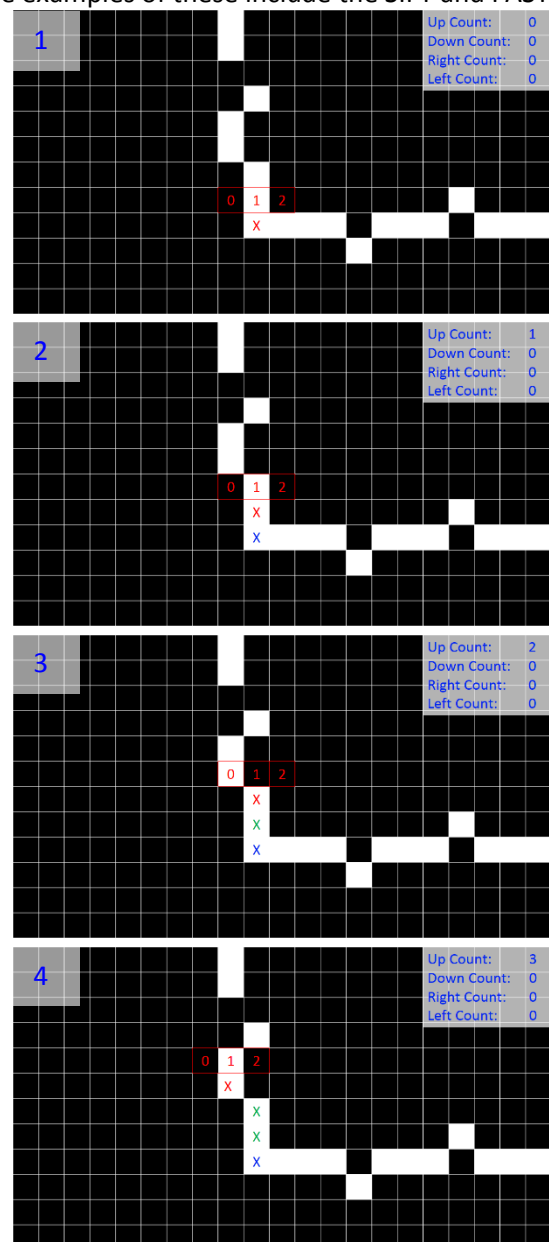
Corner detection is a standard problem in the computer vision community. Corners are usually described as features. When trying to match objects in two different images, features are extracted that will be easy to localise within an image. Corners are used instead of points on a line, as it is very difficult to define a point's position on a line, but it is much easier to define the location of a corner, at the meeting of two lines of edge pixels. As a result of how common a problem this is, a number of corner/feature detectors exist today. Some examples of these include the SIFT and FAST feature detectors, both of which were tested for this application. The results of this experimentation can be seen in Appendix A: Alternative Approaches. The results of these standard approaches were not suitable for this application, thus a new approach was created.

4.2.5 Extracting corners:

The method for extracting corners is designed around the type of corners that are being extracted. The windows have been defined as roughly rectangular regions, with four corners which are roughly ninety degree corners. These will be subject to perspective distortion and thus will not be exactly ninety degree angles. There are four types of corners, one corresponding to each of the four different corners that form a rectangle. To calculate if a corner of one of these types exists at a pixel location in the image, each pixel in the image needs to be iterated over and a score calculated at that location. To establish if a corner exists the application looks for lines of edge pixels that arrive at the point in the four primary directions; up, down, left and right. A corner then exists if there is a line in either - but not both - the up or down direction and a line in either - but not both - the left and right direction. After iterating across the entire image the corners found can be classified according to which of the four shapes they correspond to.

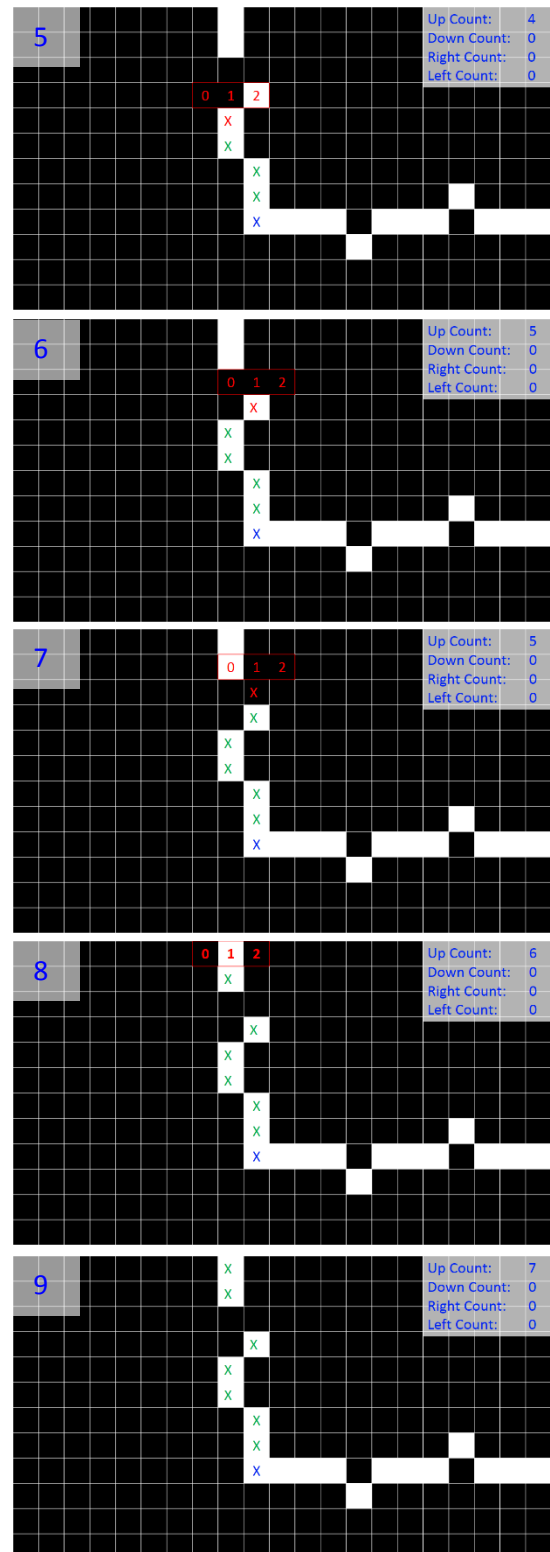
4.2.5.1 Scoring:

The method works by using four counters and iterating over every pixel in the image. At each pixel, a measure of the cornerness of the pixel needs to be calculated. To calculate a measure of cornerness, the application searches for evidence of the current location being a corner. To do this the application looks for lines of edge pixels which go through the pixel in either the up, down, left or right directions.



Each counter is used to track the amount of evidence for a line in the given direction. Thus each counter is labelled as the up, down, left and right counter.

To demonstrate how evidence is counted, the images, above and right, will examine a sample search for evidence of a line in the upward direction from a pixel. These images show a grid with black and white squares. The white squares represent edge pixels and the black represent areas with no edges. The images should be considered as a small part of a much larger picture, such as those above in Figure 15, where it has zoomed in on a 21 by 12 pixel region at a corner. In the top image it can be seen that the current pixel under consideration is marked with a red 'X'. The next step is to iteratively search for evidence of a line of edge pixels in the upward direction, from this pixel. This involves examining the three pixels above the current pixel, the one directly above, above and one to the right and above and one to the left. These are labelled 0, 1 and 2 in the top image. The application is searching for an edge pixel in one of these three locations. Finding an edge pixel will increase the counter, as each edge pixel found is one piece of evidence of a line existing in the upward direction. In the first image there is an edge pixel in the pixel directly above the current pixel, whose cornerness score is being calculated. This is accepted as evidence of a line of edge pixels in the upward direction and the up counter is incremented. The application then moves onto the next iteration. For this iteration the pixels examined are the three above the pixel where evidence was found on the last iteration. This can be seen in the second image. In this image the pixel whose cornerness score is being calculated is indicated by a blue 'X', the location of the last piece of evidence is marked with a red 'X' and the three pixels being examined for further evidence are marked in red and number 0, 1 and 2. An edge pixel is again found at location 1. The counter is incremented and the process moves onto the next iteration. Image three shows the increase in the counter and the red and blue 'X's as before, but now the previous evidence is shown with a green 'X'. The process continues as before, the only difference being the edge is found in location 0. As this process continues, the only unknown is the scenario in image seven, where there is no edge pixel in any of the locations 0, 1 or 2. The application needs to be capable of finding corners, even when continuous edges are not available. As a result of this the application needs to allow for the scenario where there is a break in the edge. To handle this the idea of a miss is introduced and a new counter used. Thus the application tracks the evidence and



miss counters. To progress to the next iteration, the application behaves as if an edge pixel was found in the location directly above the current location, but does not increment the up counter. This can be seen in image eight right.

This iterative process continues until one of the following three conditions have been met;

- The edge of the image has been reach,
- The miss counter has reached a predetermined threshold,
- The maximum number of iterations allowed, has been reached.

This same approach is used to calculate the down, right and left counter values. The process iterates through, looking at the three pixels in the direction under consideration. The final result of searching in all four directions is shown below in Figure 19. The main points to note are the counter values and how the application can handle lines that are not perfectly vertical or horizontal and how it can accommodate broken edges.

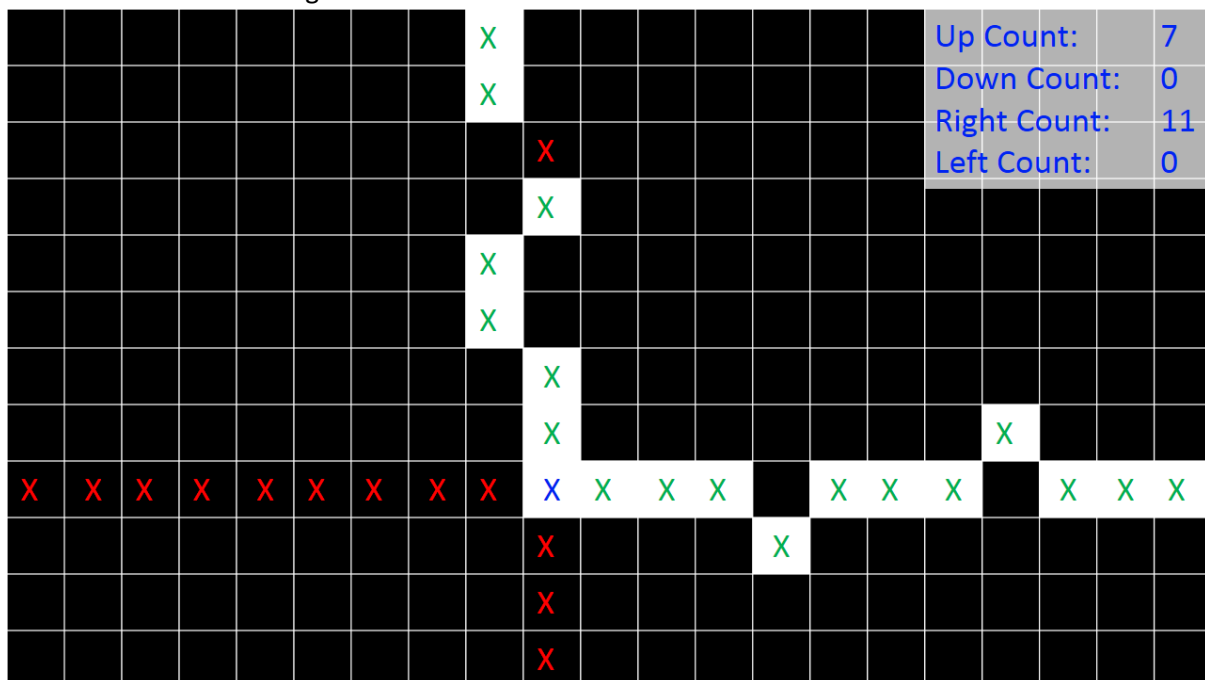


Figure 19 The final result of the cornerness calculation for the pixel marked with a blue 'X'. The red 'X's represent the misses that occurred and green 'X's marking the evidence found. Notice the final counter values in the top right corner.

4.2.5.2 Corner Classification:

Once the four counter values have been calculated for the pixel in question, they need to be evaluated to classify if a corner has been found and if so, what type of corner has been found. Using the four counters there are a number of different shapes that can be defined, based on the counter values. In considering the counters the actual value is not hugely important, but what is of importance is the value when compared to the value of the counter for the opposite direction. This is because if there is a large counter value in one direction and a small counter value in the opposite direction, the current pixel is near the end of a line of edge pixels. In this section the many different shapes that can be constructed based on the counter values are discussed and the ones of interest highlighted. These results will be labelled and then used later when the application searches for windows.

Figure 20 below, shows the four corner types the application is searching for and how each can be described in terms of the four counter values. The corners can be summarised as having a very large counter value in one direction and a very low counter value in the opposite direction. The

corner can be localised with higher accuracy if a threshold is applied to the lower counters value. This means that a pixel will only be recognised as a corner if the one direction has a very small, e.g. less than two, counter value and a very large counter value in the opposite direction.

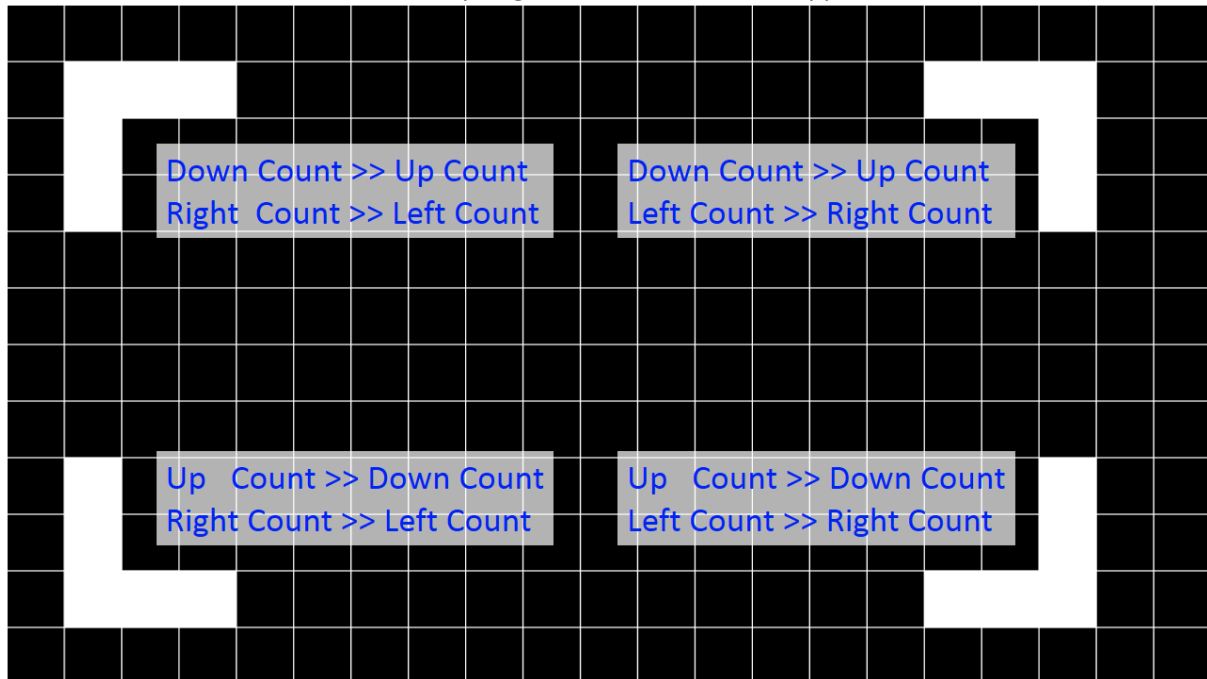


Figure 20 The four corners that the application is trying to find, they can all be expressed in terms of the counter values. Note how the combination of these corners creates a rectangular region.

Figure 21 below, shows some of the other shapes that can be described by the combination of counter values. These shapes are four different types of T-junctions that can occur. They all occur when one pair of opposite counters, i.e. up and down or right and left, are equal and the other pair has one large and one small counter value. As these shapes do not feature in the description of a window, they are all rejected at this stage.

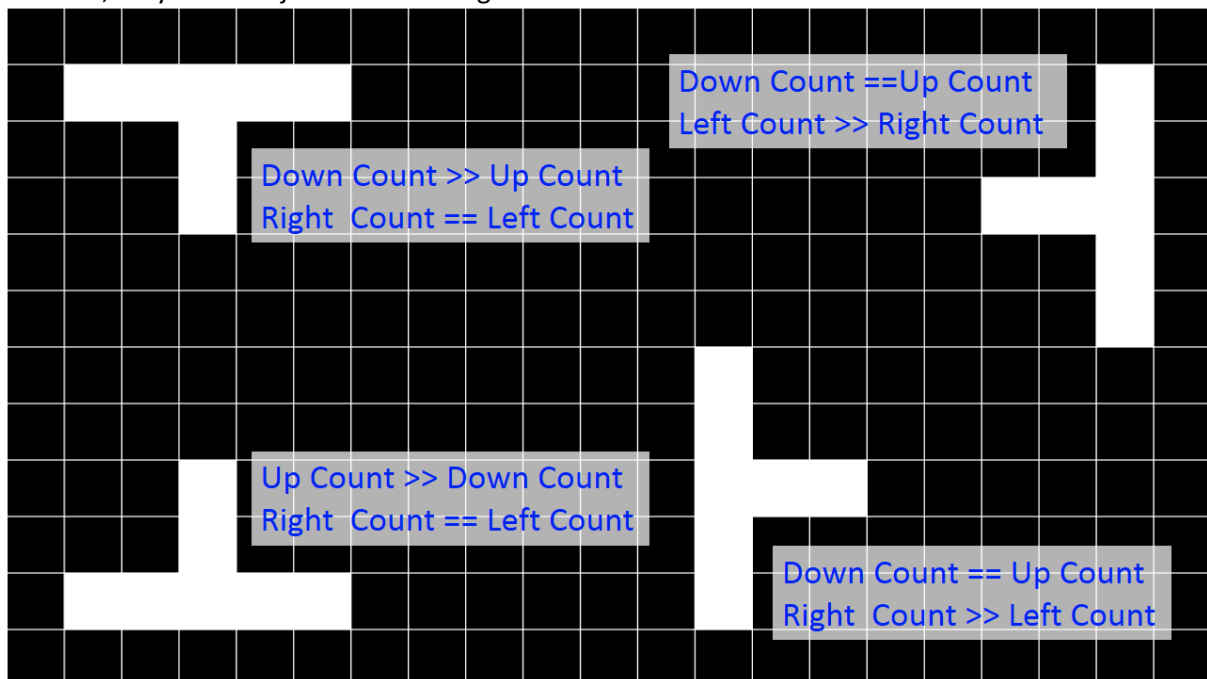


Figure 21 The four T-junctions occur when one pair of opposite counters are equal or roughly equal. The application is not interested in these shapes.

Figure 22 below, shows the two other combinations that are possible with the counters. The scenario where all the counter values are equal. This comes in two forms; where all counters are equal and greater than zero and where all counter value are equal to zero. In the first scenario the shape created by the edge pixels is a crossroads shape. The second scenario is where there is no evidence of any edge pixels around the current pixel. Both of these scenarios are not used in finding windows and thus, both are rejected.



Figure 22 Shows the remaining two scenarios where all counters are equal and the result is a crossroad shape and when all the counters are zero and no cornered shape exists.

After examining the counter values and analysing the shapes implied by the relationships between the counters for opposite directions, it is possible to extract a number of different shapes. For this application four of these shapes are relevant, those in Figure 20 above. Thus when the application iterates through each pixel in the image, calculates the four counter values and then determines which type of shape exists at the location, the pixel can be labelled accordingly. The labelling application numbers the four desired corners as non-zero values, e.g. 1, 2, 3 and 4, and all other shapes are labelled as zero. This means that after iterating across the image, the resultant new output image has pixels with one of five values and based on these five values, it can be determined what, if any, corner was found at that location.

4.2.5.3 Corner Localisation:

At this stage the output is a new image of the exact same size as the edge image. The difference being the new image has the pixels labelled as one of four different types of corner, or as not having a corner located at that pixel. The remaining issue is that this method does not determine the corner location to be at one pixel location, but that it could be located at any pixel in a small region around the corner. This is illustrated in Figure 23 below, where there is a small region of pixels, around the exact corner location, which have been labelled as corner points. To progress further with this and use these results to extract windows, the corners exact location needs to be localised to a much higher degree. Ideally the region of possible locations would be reduced to a single pixel.

A number of approaches could be taken to solve this problem. One such solution is to give a score to each pixel, based on the counter values. It is reasonable to assume the closer to the true

location of the edge pixel, the higher this score would be and the further away from the true corner location, the lower the score would be. The scoring application trialled was;

$$\text{pixel score} = \text{abs}(\text{up counter} - \text{down counter}) + \text{abs}(\text{right counter} - \text{left counter})$$

This would then be suitable for non-maxima suppression, which, ideally, would reduce the region to a single point. Experimentally, it has been discovered that while this approach reduces the region size, it does not reduce the region to a single pixel. This is because the pixels surrounding the exact corner location, have the same score. Thus instead of having a peak, which non-maxima suppression can find, there is a plateau which non maxima suppression can find. The end result is simply a smaller region of possible corner location.

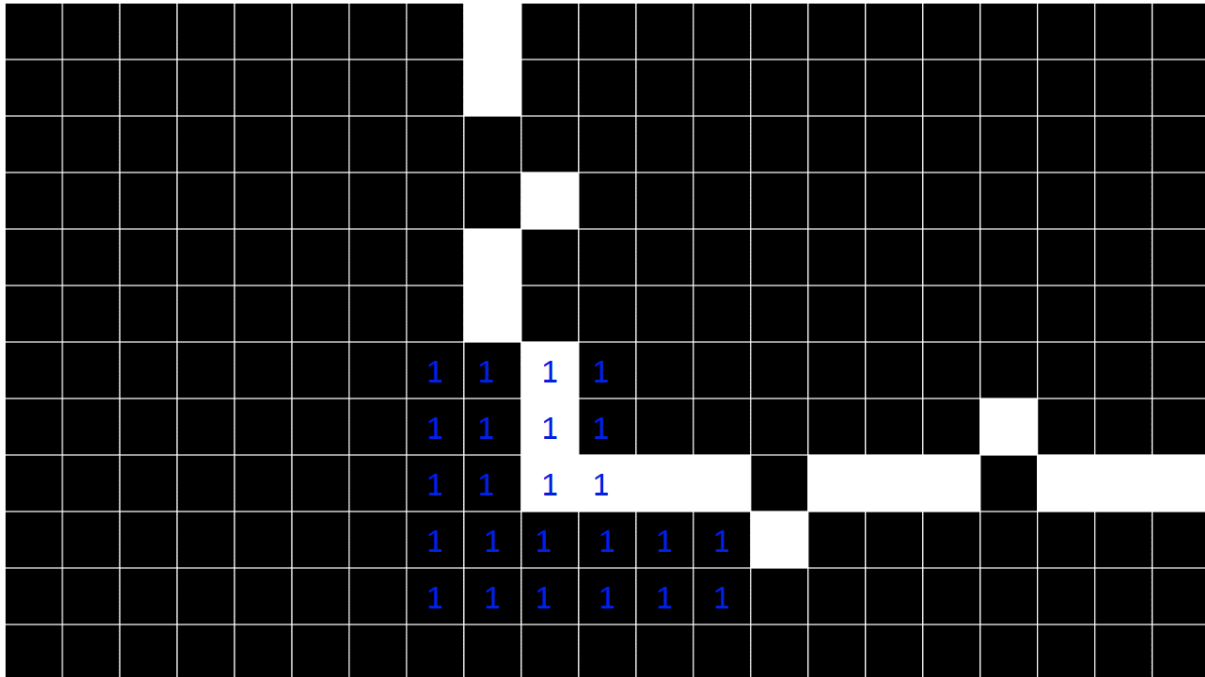


Figure 23 A labelled output image. Notice how the corner is not localised to one point, but that a response has been detected across a region. The pixels are all labelled with the number one to indicate the type of corner detected.

This approach could be employed, but isn't 100% suited to the needs of this application. A second method trialled is illustrated in Figure 24 below. The image in Figure 24 shows a mock-up of the edges which outline a window. The central image shows the regions around each corner, where corner scores have been detected. The rightmost figure in the image shows the proposed solution, which is discussed below.

The proposed method involves examining the region of possible corner locations and the pixels in this region, which are also edge pixels. Only pixels in the suggested corner region which are on an edge are accepted and all other responses are discarded and set to zero. This is illustrated in Figure 24, as the large region shown in the central illustration, is reduced to a much smaller region in the rightmost illustration. The advantage with this approach is that it lends itself very easily to the definition of a window, discussed at the start of this chapter. The definition is four corners, all connected by edge pixels. This means having the corners located on the edge itself will aid in the searching process. As another example, examining Figure 23 above, only four of the pixels labelled with a one, would be left. These are the four on the white squares, representing edges.

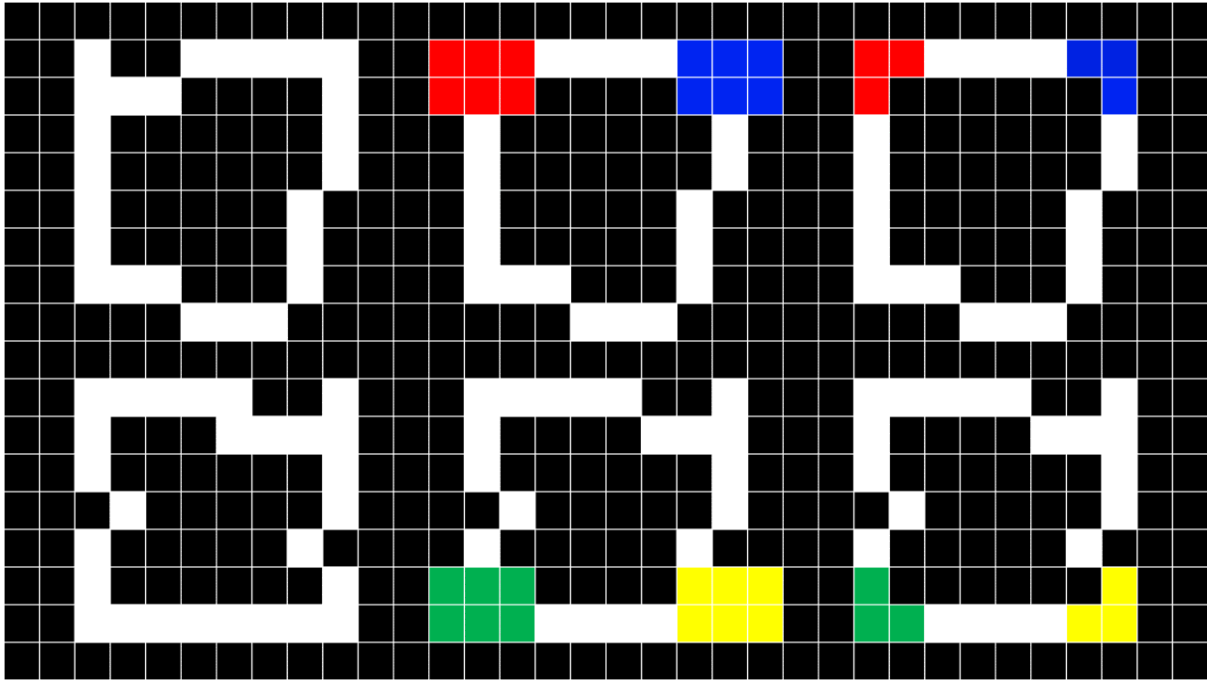


Figure 24 A mock-up of a window outline, showing the edges in white. The central window shows the region of possible corner locations, all colour coded by corner type. The rightmost window shows the result of only accepting corner pixels which are on edges.

4.2.6 Window extraction:

In the previous section the process of extracting corners was discussed. That section highlighted the four different types of corners the application searches for. After the corner extraction stage, the labelled output corner image is used as input to the windows extraction stage. At the start of this chapter, the definition of a window was discussed, this section will look at how, using that definition and the corner image, windows can be extracted from the image.

To find a window the application needs to locate four corners, one of each type. These four corners then need to be positioned correctly relative to each other. Finally they must be joined by edge pixels, which mark the outline of the window. For the application to extract windows an algorithmic approach is taken. The algorithm goes through a number of steps and it is only if all of the steps and tests succeed, that it is determined a window has been found. The application iterates through each pixel in the image and follows the steps given below;

- 1) The current pixel has been labelled as a bottom left corner.
- 2) Follow the edge upward, similar to the up counter calculation above, until;
 - a) Reach a pixel labelled as a top left corner,
 - b) The vertical search limit has been reached,
 - c) The maximum allowable missed edges has been reached.
- 3) If the top left corner is located, follow the edge pixels to the right of this until;
 - a) Reach a pixel labelled as a top right corner,
 - b) The horizontal search limit has been reached,
 - c) The maximum allowable missed edges has been reached.
- 4) Return to the bottom left pixel from step one, follow the edge pixels to the left until;
 - a) Reach a pixel labelled as a bottom right corner,
 - b) The horizontal search limit has been reached,
 - c) The maximum allowable missed edges has been reached.

- 5) Check that the difference between the x coordinates, columns, of the bottom right and top right corners is less than a predetermined threshold. This ensures the right hand corners of the window are roughly vertically aligned.

If all six steps are successfully executed, the four corners are deemed to define a window. These steps are illustrated in the right hand side diagram in Figure 25 below.

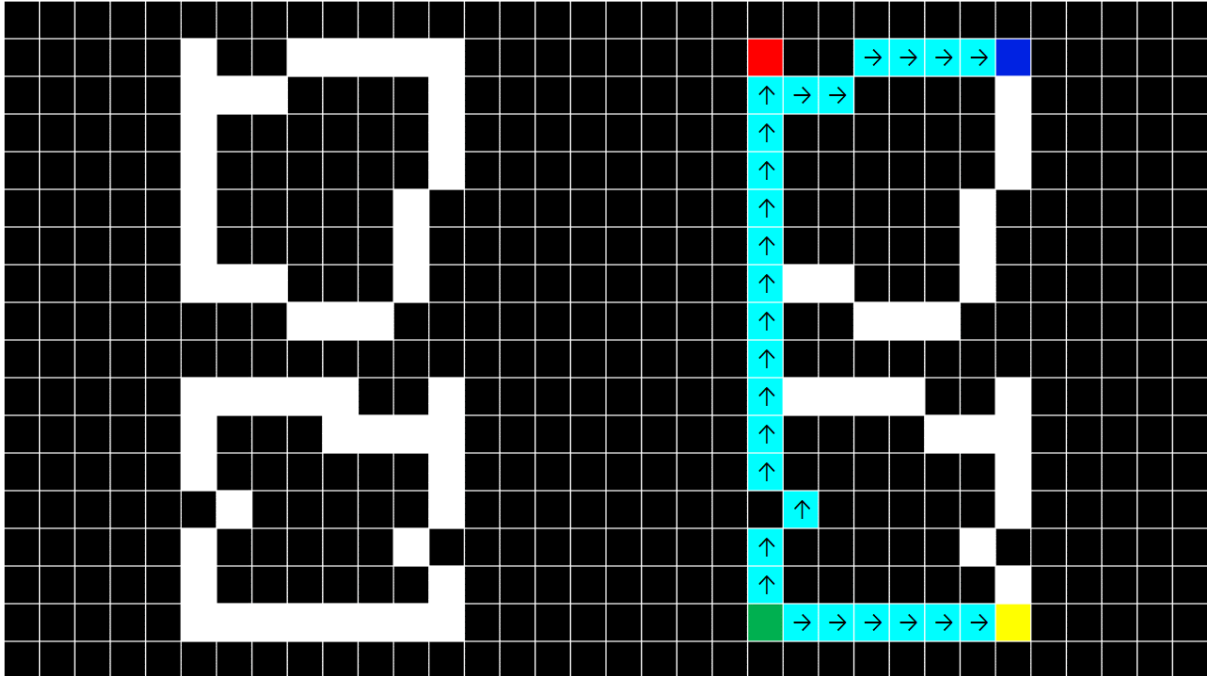


Figure 25 An example showing how the application searches an image to find a window. The light blue paths shows the application searching for corners, in the correct position relative to the other corners.

Once the windows have been detected, they need to be stored. The windows are described by their four corners, using the x, y coordinates of these corners. The application creates a vector of OpenCV points, which will hold the four points in the following order;

Vector Location	0	1	2	3
Corner:	Top Left	Top Right	Bottom Left	Bottom Right

To store all of the windows found within the image, all of these vectors are stored in another C++ vector. This allows easy access to all of the windows and in the next chapter these will be used to create a description of the building.

4.2.7 Conclusion:

In this chapter the process of extracting windows was discussed. The overall application requires the windows of a building to be extracted, as they form the basis of how the application compares two buildings. The method for extracting the windows is broken into a number of different stages. Firstly the image is converted to greyscale, and an edge detector is used to find the edges in the image. These edges are then subject to thresholding and non-maxima suppression. This removes weak responses and gives edges that are one pixel wide. The goal here is to have the outline of the window clearly defined by edge pixels. Next the application begins searching for the corners that make up a window. The application uses four counters in order to calculate the evidence for a line of edge pixels going from the current pixel in the up, down, right and left directions. Based on the values stored in the four counters, each pixel can be classified by the shape discovered. There are many shapes possible, as shown above, but only four of these are of interest

for finding windows. These four types of shapes are labelled accordingly and all other shapes are removed. At this stage the application can search for the window itself. The window is defined by having one of each type of corner, a top left, top right, bottom right and bottom left corner. These corners must be joined by lines of edge pixels in the correct order. The application searches from any bottom left corner found and seeks to find the other three corners, by following the edge pixels which denote the outline of the window.

The result is a C++ vector which contains all the windows found, the windows are described by the x and y coordinates of their four corners. This is then passed on to the next stage in the overall application, building description.

4.3 Façade Description:

4.3.1 Introduction:

In the previous section the application extracted the windows present in the image. These windows were stored as four points, one for each corner of the rectangular region that defines a window. This section takes the extracted windows and builds a description of the building facade(s) present in the image. This description is based on the layout of the windows, how they are positioned relative to one another and their size. The description of a building is broken into two parts; the first is a description of the columns of windows in the image and the second is a description of each column.

4.3.2 Columns of windows:

The first step in describing the building is to group all of the windows into columns. This involves identifying which windows are below other windows and vertically aligned with each other.

4.3.2.1 *Extracting Columns:*

To form columns from the windows found, it is necessary to find the topmost window of each column and then find the other windows which belong to this column. This means all the other windows will be below the first window. Thus the first step is to identify the top most windows in the image. This means the vector containing the windows needs to be ordered so that the windows closest to the top of the image are first and the windows closest to the bottom of the image last.

With this ordering, the list of windows is iterated through until all windows have been assigned to a column. To assign windows to columns, the application needs to test if one window is below another and if they are vertically aligned. The strategy for creating columns is given by the following steps;

1. Find the topmost window in the image and remove from the list.
2. This is a new column.
3. Iterate through all other windows and check if they are vertically aligned.
 - a. If they are, add it to the column and remove it from the list.
 - b. If not, leave them in the list.
4. Find the next topmost window and remove it from the list
5. This is a new column
6. Iterate through all other windows and check if they are vertically aligned.
 - a. If they are, add it to the column and remove it from the list.
 - b. If not, leave them in the list.
7. Repeat steps 4, 5 and 6 until all windows have been assigned to a column.

The ordered list of windows now guarantees the windows at the top of the list are above all the windows that follow in the list. Provided the list is iterated through from start to finish, and only

windows after the current window are considered, it can be guaranteed that all windows considered are below the current window.

This leaves only the task of check vertical alignment. This means that the window being consider is not just below the top window, but vertically in line with the top window. To check for vertical alignment the slope of the side of the top window is calculated, this becomes the reference angle, θ_R . Now the application iterates over all the other windows, which are below this top window. On each iteration a line is constructed between the centre point of the top window and the current window for this iteration. The slope of this line is calculated as θ_T , the test angle. The angles θ_R and θ_T are calculated using the following equation, which uses the two end points, p1 and p2, of the line;

$$\theta = \text{atan2}(p1, p2)$$

This returns a value in the range $-180^\circ \rightarrow +180^\circ$ in radians. For the purposes of this application the range is ideally $0^\circ \rightarrow 360^\circ$. To achieve this, 2π is added to any negative results, thus achieving a range of $0^\circ \rightarrow 360^\circ$. To calculate if the two windows are vertically aligned the two slope angles are compared. If their absolute difference is less than a predetermined threshold, the windows are deemed to be vertically aligned.

Figure 26 below, shows an example of this method. It shows only three of the tested windows and how the angles are calculated, in red. The blue angle represents the slope of the side of the topmost window. In this example it can be seen that only the window directly below will produce a θ_T value which is similar to θ_R .

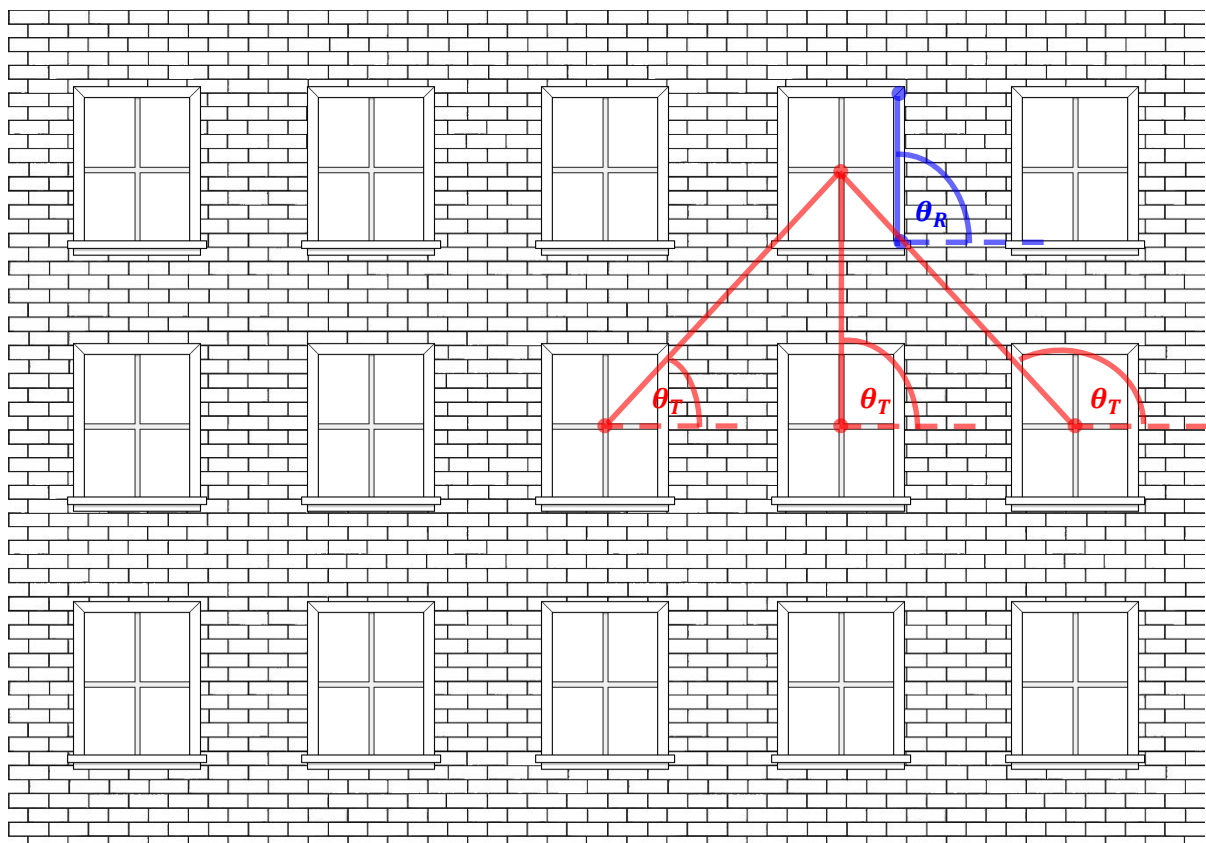


Figure 26 An example of the vertical alignment test. Note the two theta values extracted for comparison. The reference angle is shown in blue and the test angles in red.

The result of this is shown below in Figure 27 below. This highlights which windows belong to which column and these have been colour coded for illustration purposes. It should be noted that this

image is an idealised version of the type of window layout that would be expected. The image has no perspective distortion and all the windows are aligned perfectly into columns. The application needs to be capable of performing the same calculations on real world images with distortion and irregular window layouts. This is catered for by thresholding the absolute difference of the reference and test angles, instead of searching for perfect matches between the angles.

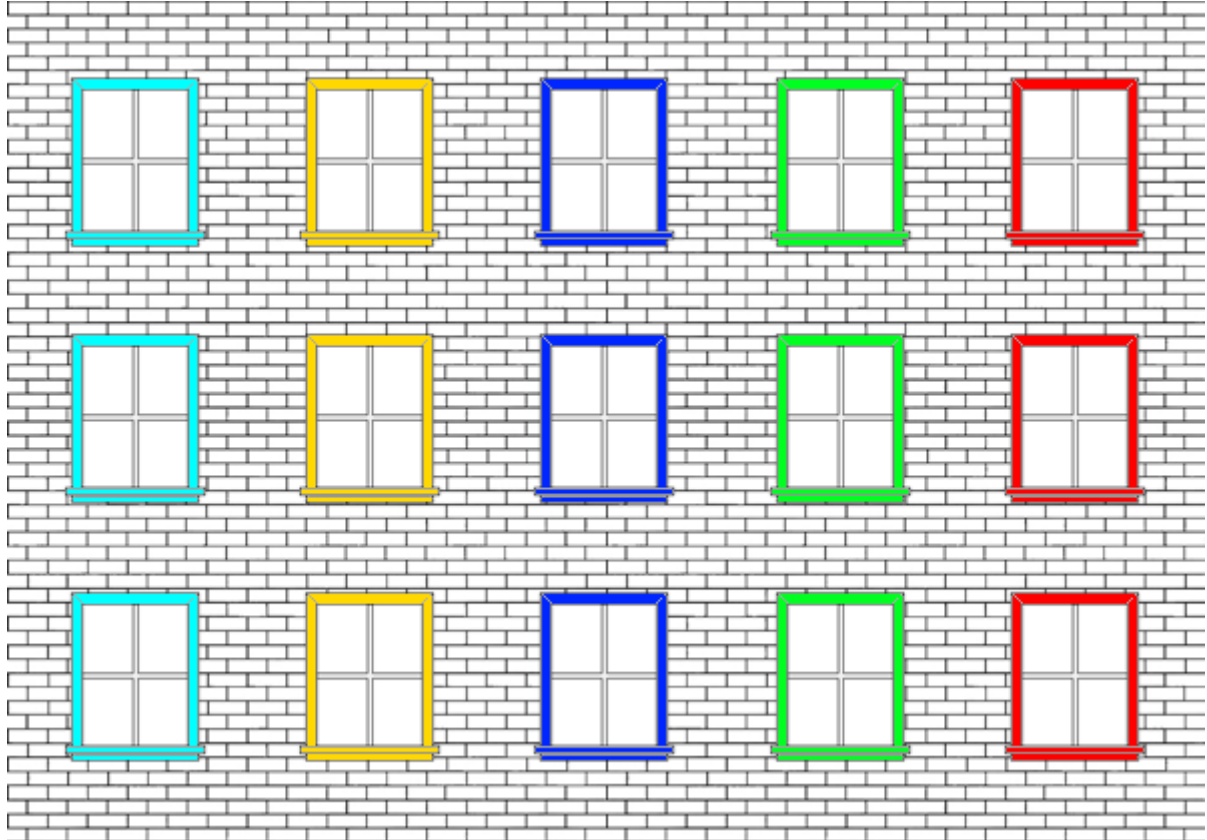


Figure 27 The image highlights, in colour, which windows are grouped together into columns.

4.3.2.2 Column Description:

Once the columns have been extracted, a method for describing an individual column is needed. This needs to encompass information about the windows in the column and their relative positions with respect to one another.

The windows size needs to be described in a manner that is invariant to changes in perspective distortion. This means describing the size using an exact pixel measurement is not appropriate. Thus the approach used is to calculate the aspect ratio of the window, the ratio of the width to the height of the window. To describe the layout of the windows another ratio is used. This time the layout being describe is based on where the other windows are, relative to one window. The values compared are the distance between the windows and the height of the reference window.

Figure 28 right, shows an example of this, with the middle window being used as the reference window. The ratio to describe how far away window A is from window B is given by the following ratio;

$$distance\ ratio_{BA} = \frac{h_{BA}}{h_B}$$

4.3.2.3 Layout Description:

With the windows grouped into columns, the next tasks is to build a description of what the building façade looks like. This description needs to be general, as to allow a similar description to be extracted from different images of the same scene. These different images may be subject to many distortions such as different lighting conditions and perspective distortion.

The goal is to describe the building in terms of the layout of the columns. This means that for each column, it is necessary to establish a standard method of describing how many columns are on either side of a particular column and roughly how far away from the current column they are. This requires a standard way of measuring the distance between the columns that will be constant, or roughly constant, across many different images.

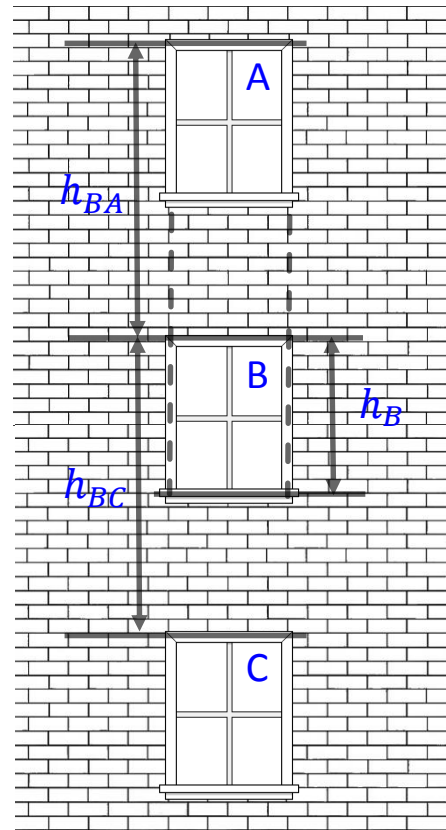


Figure 28 This image shows how the positioning of windows and the windows sizes are calculated

The simplest method would be to measure the distance between columns in pixels, e.g. column B is 10 pixels to the left of column A. The issue with this method is these distances can be elongated and contracted by perspective distortion and errors in the edge extraction techniques. The chosen method is to relate the distance between the columns to the width of the current column. This equates to saying column B is a distance of four times the width of column A to the left

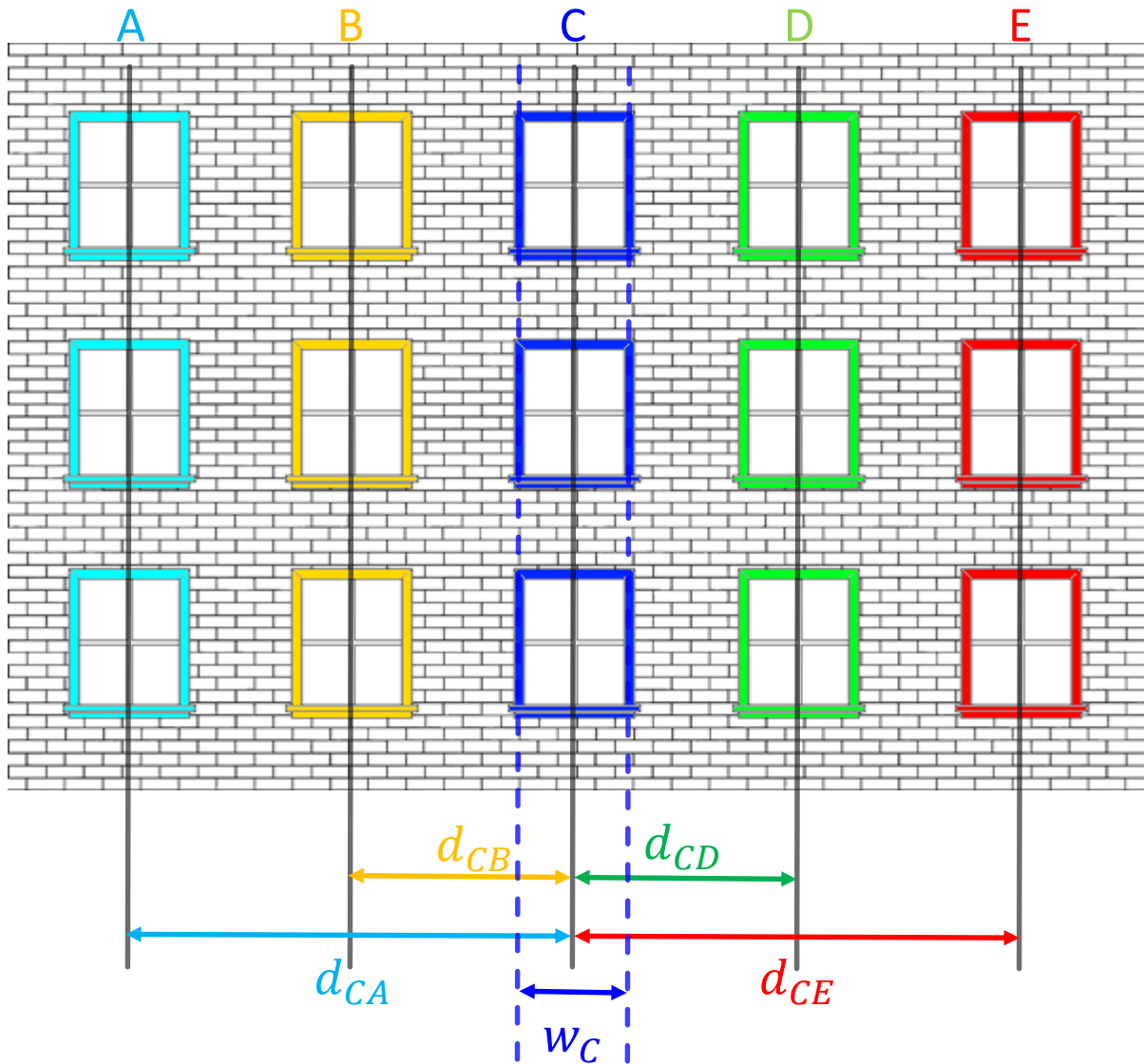


Figure 29 An example of how the columns are described in relation to one another. The example above shows how the layout of the columns relative to column C is described.

from column B. This is based on the idea these ratios will be more stable than fixed pixel lengths, when compared between images taken with different levels of perspective distortion.

$$distance\ ratio_{Column\ C\ and\ X} = \frac{d_{CA}}{w_C}$$

4.3.3 Conclusions:

In this section three main elements of the application are discussed. The method of grouping windows into columns, describing the layout of the windows in each column and describing the layout of, all of the columns. The windows are grouped into columns based on vertical alignment.

The process of calculating this needs to be flexible to allow for minor deviations caused by perspective distortion. To describe the layout of the windows within a column two measures are used, the first is the aspect ratio of the windows. This gives a representation of the size of each window. The second measurement is a ratio of the vertical distance between two windows, divided by the height of the reference window. To describe the layout of the columns a width ratio is used. This is the ratio of the vertical distance between the windows divided by the width of the reference window. These methods give a defined way of describing these layouts and will be used in the next section to match buildings found in two different images.

Ratios are used because they should remain roughly constant under minor changes in perspective distortion. Based on the intended usage of this application, it is acceptable for the application to fall short when recognising buildings subjected to large perspective distortions. A user will not expect good performance when the building is not view roughly straight on, if the viewing angle is quite steep, it is acceptable for results not to be shown.

4.4 Façade Matching

4.4.1 Introduction:

In this section the process of comparing two images, to find matching buildings is discussed. The matching between images uses the description metric discussed in the previous section. The previous section discuss how the layout of the columns relative to one another is described and how the layout of windows within a column are described. This section uses these descriptions to calculate a score for two images, this score indicates the likelihood of the same building being in both images.

The strategy for calculating this score is broken into two parts. The first part seeks to identify which columns in image one, match to which columns in image two, based on column alignment. The second part seeks to measure how well the windows match, within the matched columns. To match the columns the application uses an iterative approach where each column in image one is matched to each column in image two. On each iteration evidence is gathered to support the possibility of the two columns matching. Gathering evidence involves searching for the other columns around the current column in image one and trying to find corresponding columns in image two, positioned relative to the current column in image two. To calculate a score of how well the windows in two columns match, all windows in the first column are matched against all the windows in the second column. The application compares them on aspect ratio and the spacing between the windows.

4.4.2 Matching Columns:

In this section the goal is to determine which columns in image one match most closely, with which columns in image two. This equates to comparing the layout of the building in image one with the layout of the building in image two. The process works iteratively, over each iteration a column in image one is matched to a column in image two and evidence supporting this match is calculated. This is repeated until every column in image one has been matched against every column in image two.

On each of these iterations evidence is gathered, to discover how much supporting evidence exists to for the match under consideration. The process will be illustrated using the two images below as an example. These images are perfect copies of one another and offer an idealised scenario on which to demonstrate the process. The process iterates through the columns in image one, left. On each iteration the application compares the current column in image one, to every other column in image two, right.

On each comparison of a column in image one with a column in image two, evidence is gathered for this match. The evidence is based on locating columns surrounding the column in image one, and finding corresponding columns in image two. The corresponding columns need to be in roughly the same location as those in image one, with respect to the column under consideration. For example, consider the case when comparing column one in image one with column one in image two. First the other columns in image one are considered individually. The first column considered is column two in image one. The distance ratio between column one and two is calculated. Then the columns in image two are considered. The application iterates through columns two, three, four and five and for each column calculates the distance ratio between column one and the other column. These four ratio values are compared with the ratio from image one and the ratio which most closely matches that from image two is selected, provided the difference between the two ratios is below a predefined threshold. The absolute difference between the ratios is added to the overall score. In this case the ratio corresponding to column two in image two would be selected. Then the application considers column one and three in image one, and calculates the distance ratio. Then the application searches image two again, examining the distance ratios for columns three, four and five (not two, as column two already has a match) in image two. The ratio most closely matching the ratio from image one is selected, provided the difference of the two ratios is less than the threshold. This absolute difference is then added to the overall score as before. The process continues until all columns in image one have been tried and either a match found for them, or not.

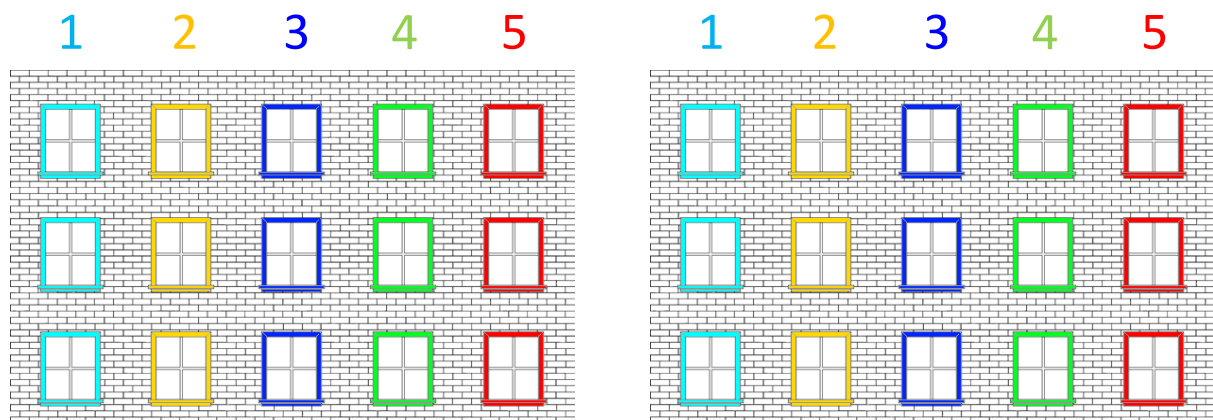


Figure 30 An idealised example of two images which are under comparison. The goal is to determine which columns in image one match to which columns in image two.

Once this completes a score for this iteration, column one in image one matching column one in image two, is calculated. This score is the sum of the absolute difference of the matching distance ratios, divided by the number of columns matched. This is where the overall score variable used above is needed, this contains the sum of the absolute difference between each of the accepted ratios. This is then divided by the number of matched columns, to normalise the score to an average score per column found. This score is then stored in an array of size N by M, where N is the number of columns in image one and M is the number of columns in image two. In this case the application was matching column one in image one with column one in image two, thus the result is stored at location (N-1, M-1), (0,0).

		Column Two				
		1	2	3	4	5
Column One	1	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
	2	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
	3	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
	4	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
	5	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Figure 31 This table shows how the scores for each possible match is stored, the score at location (X,Y) is the score for column X in image one, matching column Y in image two

This iterative process of matching every column in image one to every column in image two, searching for evidence and building a score based on this evidence is repeated until the table above is full. The next step is to examine the table above and establish what match is best.

In this case the lower the score the better, as it indicates the distance ratios are very similar. Thus the application searches through the array for the location with the lowest score. This score is then accepted as a match between two columns. The lowest score occurs at a location (x, y) and this indicates that column x in image one matches to column y in image two. Once this match is accepted, there are now some impossible combinations in the array which need to be removed. For example if column three in image one matches to column three in image two, then column three in image one cannot match to another column. Thus the row in the array corresponding to column three in image one is removed and likewise with the column in the array corresponding to column three in image two. There are some more impossible combinations to discount. As the columns are numbered in order, this infers one column is to the left or right of the others. Thus it can be concluded that column three in image one matches column three in image two, it is impossible for column one or two in image one to match column four or five in image two. This is because the ordering means columns one and two are to the left of column three. This means the area of the array to the left and down of the match, and to the right and up of the match need to be removed. These are highlighted in the table below.

		Column Two				
		1	2	3	4	5
Column One	1			X	X	X
	2			X	X	X
	3	X	X	MATCH	X	X
	4	X	X	X		
	5	X	X	X		

This reduces the remaining search space and to find further matches again the lowest value is found and accepted as a match. The new impossibilities are removed and the process continues until all spaces in the array are marked as either a match or an impossibility. At this point the matching columns between the two images have been identified. A final example table, based on the example above is shown below.

		Column Two				
		1	2	3	4	5
Column One	1	MATCH	X	X	X	X
	2	X	MATCH	X	X	X
	3	X	X	MATCH	X	X
	4	X	X	X	MATCH	X
	5	X	X	X	X	MATCH

4.4.3 Matching Windows within Columns:

After the column matching stage above, the application knows which columns match between the two images. The next step is to calculate a measure of how well these two columns match. This measure will be based on the window sizes and the relative positions of the windows.

To calculate a score, the first step is to establish which windows in the two columns match. This is achieved by iteratively comparing every window in column one, with every window in column two. An N by M array is then constructed, where N is the number of windows in column one and M is the number of windows in column two. This array is then used in a similar manner as the one used to identify which columns in the two images match. The method for calculating the scores to fill this array is broken into two phases, the first sweep and the second sweep. For the first sweep there are no available matches, thus no positional information is available. This means the application cannot use the distance ratio between windows to calculate a score. Thus for the first sweep only the difference between the two aspect ratios is used to populate the array. This will yield one match and this match will then be accepted, providing it is below the threshold. The impossible options are then removed from the array. Finally the scores are recalculated, this time using the absolute difference of the aspect ratios summed with the absolute difference in the height ratio discussed above, using the first match as the reference window. This process of accepting matches, removing impossible options and recalculating scores is repeated until all options in the array are marked as either matched or impossible. From these matches a score is calculated based on the height ratio and aspect ratio.

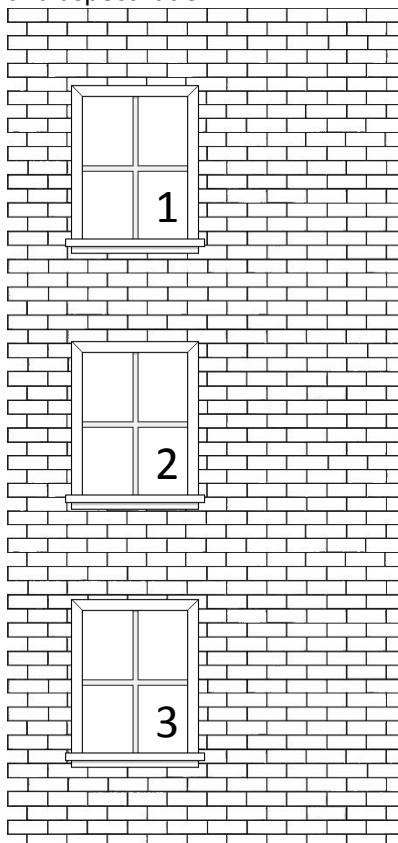


Figure 32 The first column for the example. Note the three windows on the building and their spacing.

To examine the window matching algorithm more closely, an example is provided below. Figure 33 and Figure 32 show a single column of windows. At this stage the application has identified that these two columns match, according to the layout of the columns. Now the application begins to examine how well the windows within the column match.

In this example there are three windows in the first column ($N=3$) and two in the second column ($M=2$). Thus the array will have three rows and two columns. The first sweep of the application compares window one in column one with windows one and two in column two, then compares window two in column one with windows one and two in column two and

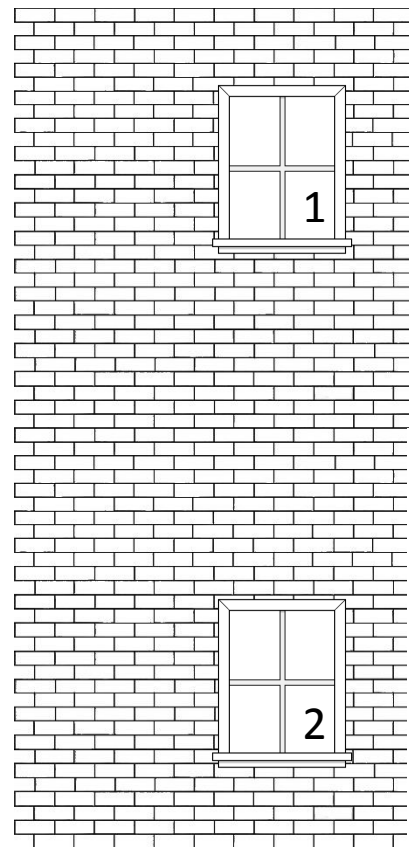


Figure 33 The second column for the example. Note that the middle window was not found, this can occur and the application needs to be capable of allowing for this.

likewise for window three. The comparisons for the first sweep are made entirely based on the absolute difference of the aspect ratio. Thus each value in the array, for the first sweep the values in the score array are described by the following equation:

$$score[X][Y] = abs(AR_{window X column 1} - AR_{window Y column 2})$$

After all iterations have completed the results table will be full and the matching process can begin. For this example assume the lowest score occurred at (0, 0), thus window one in column one is matched with window one in column two. The table after the matching and removal of impossible options is shown below;

		Column two	
		Window 1	Window 2
Column one	Window 1	Match	X
	Window 2	X	
	Window 3	X	

At this stage the second sweep can begin, in this sweep only the remaining options are considered. The score for matching windows is now based on two factors; the difference in aspect ratios and the difference in the distance ratio described in the last section. This means the score is based on the window sizes and positions. The values for the score array locations are given by the following equation;

$$score[X][Y] = abs(AR_{window X column 1} - AR_{window Y column 2}) + abs(DR_{window X column 1} - DR_{window Y column 2})$$

In this case the lowest score in the array will be at location (2, 1) in the score array. This means window three in column one is matched to window two in column two. The final score table is shown below;

		Column two	
		Window 1	Window 2
Column one	Window 1	Match	X
	Window 2	X	X
	Window 3	X	Match

After the above steps have been completed, all matches have been found and the final score can be calculated. The advantage of this technique where two separate sweeps are used, is illustrated in this example. There is a possibility that if the score was calculated purely on aspect ratio on the second sweep, that window two in column one would have been matched to window two in column two. Visually readers can see this is wrong, but to a computer using only aspect ratios for comparison, this could appear correct. Adding the relative position information into the calculation has helped to guarantee that window three in column one is matched to window two in column two, as window two in column one will have a very poor distance ratio score. This is due to the two ratios being very different.

4.4.4 Overall score:

Upon reaching this step, the columns have been matched and the windows within the columns have been matched. The final task is to extract a percentage score which gives an indication of how well the buildings in the two images match.

Each pair of matching columns contribute an individual part to the overall score. The value contributed by a column is the sum of the scores in the "match" locations in the score array for the pair of columns, scaled by the number of windows matched. Thus in the example below, the pair of

columns would return a score equal to the sum of the scores at array locations (0, 0) and (2, 1). This value would then be scaled by a factor of 2/3 as only two of the three windows in the larger column have matches. This score is subtracted from one and then converted to a percentage. If the sum of scores times the scaling factor is greater than one, the percentage score is set to zero.

$$Column\ Score = \left[1 - \frac{No.\ windows\ matched}{No.\ windows\ in\ largest\ Column} * \sum\ matched\ scores \right] * 100$$

The scores for each column pair is summed together and divided by the difference between the number of columns in the image with least amount of columns, minus the number of columns matched. Thus giving a score based on the window layout, weighted in favour of layouts using more columns, i.e. using more evidence. The application can then use this score to judge if the buildings match or not.

4.4.5 Conclusions:

In this section the matching algorithm is discussed. The method involves matching columns between the images based on finding similar layouts of columns in both images. After the columns are matched the windows within the columns are matched and a score calculated based on the distance ratio and aspect ratios of the windows.

The application uses the ratio based descriptors discussed in the previous section to compare windows and columns in different images. The first stage is to identify the columns in image one and their matching columns in image 2. The method chosen is to compare every column in image one with every image in column two and gather evidence for that potential match. The evidence to support a match is based on finding surrounding columns in image two, which correspond to the surrounding columns in image one. Once the columns have been matched, a score is extracted from each column pair based on the windows within the columns. The windows are matched based on aspect ratio and distance ratio. These two measures are used to calculate a score based on how closely the alignment and sizes of the windows match between the two columns.

4.5 Summary:

This section describes the implemented application in detail. It examines the four main stages of the application; GPS extraction, windows extraction, façade description and façade matching.

The GPS data received is going to have errors present. These errors are generated by a number of different source. The GPS modules on different mobile devices will perform differently under the same conditions and the data collection methods employed by Google mean data is available at fixed intervals, not continuous locations. The extent of these sources of errors needs to be evaluated, to identify what (if any) rectifying steps need to be taken.

The window extraction process involves a number of stages. The first is a corner detector, specifically implemented for this application. The corner detector finds corners and classifies them by their shape. The next step looks for four corners, grouped together in the pattern of a rectangle. These corners must be joined by edge pixels, outlining the window.

The facades are described by the layout of the windows extracted. The windows are grouped into columns. The positioning of the columns relative to each other is used when comparing buildings across images.

The matching algorithm searches for the layout of columns in one image, in the other. Then a score is calculated based on the layout of the windows within matched columns. The comparisons are made using ratios to more accurately represent distances under perspective distortion.

These techniques are untested and now need to be examined from a performance perspective. This examination and testing is carried out in the next chapter.

5 Results and Testing:

In this section the experimental results are presented for the application described above. The results are broken into four sections; GPS, the corner detector, the window extractor and façade matcher. As each section builds on the results of the previous section, the performance analysis needs to consider errors propagating through all stages.

5.1 GPS

In this section two tests on GPS coordinates are presented. The first examines the variability in the GPS coordinates provided by a mobile device over time. The second test compares the image from the camera on the device and the Google Street View image at the corresponding location. The aims of these tests are to examine the stability of the GPS coordinates provided by a mobile device and compare the camera image to the Google Street View image.

The GPS coordinates provided by mobile devices will differ from device to device. Google Street View has data at a series of points and returns data for the point closest to the actual coordinate given. These two facts raise a big issue around ground truth. Ideally when comparing the data from Google Street View and a number of mobile devices, the GPS coordinates returned would be compared with some ground truth values. This would allow for measurable errors to be presented, but such a ground truth is not easily acquired.

This means comparisons will have to be based on different criteria than the exact GPS coordinate. To establish this it is useful to consider the needs of the application. The application requires that the same buildings be present in the camera and Google Street View images. Thus when evaluating the performance of the devices and the Google Street View platform the contents of the images compared with each other will be the main factor.

5.1.1 Test Case One:

5.1.1.1 Method:

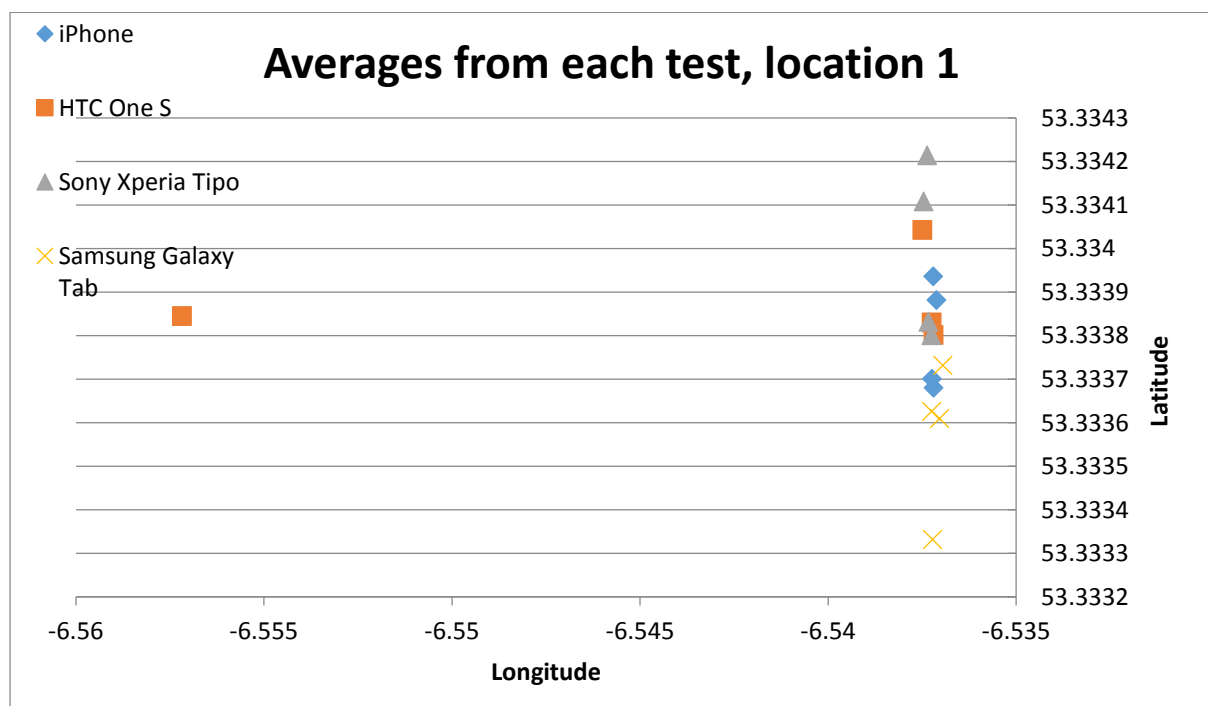
The goal of this test is to study the stability of the GPS signal from a number of different devices over time and establish any settling time issues. This test uses two locations and involves four mobile devices; a HTC One S, a Samsung Galaxy Tab, a Sony Xperia Tipo and an iPhone 4S. Each device is numbered one to four. Testing is performed by going to location one and taking six photos at the location. Each device is used in the prescribed order. This is repeated, using the same ordering at location two. To establish any issues with stabilisation of the GPS signal the tests are repeated using a different ordering as shown in the table below. Great care was taken to stand in as close to the exact same position on each iteration of the test, but inevitably a small error in the results could be attributed to this.

Device number:		Device:	
1		HTC One S	
2		Samsung Galaxy Tab	
3		Sony Xperia Tipo	
4		iPhone 4S	
Ordering of devices used for each test			
Test 1	Test 2	Test 3	Test 4
1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

5.1.1.2 Evaluation of results:

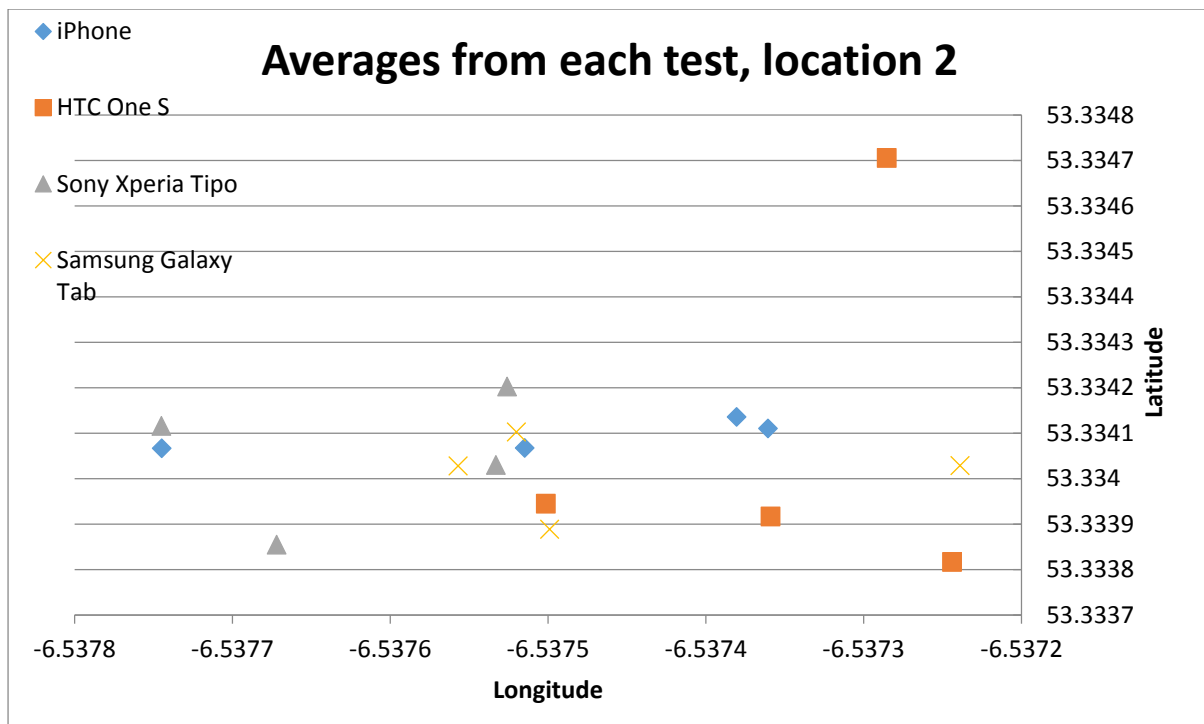
The table below, shows the average values for longitude and latitude for each device at each location for the four tests. The value presented is the average value across the six photographs taken. Note that the longitude values are negative, but have been converted to positive values for presentation purposes. The values used in the graphs below are correct.

Iphone		HTC One S		Sony Xperia Tipo		Samsung Galaxy Tab	
Lat.	Long.	Lat.	Long.	Lat.	Long.	Lat.	Long.
Test Location One							
53.333680	6.5372027	53.333844	6.557181	53.333801	6.537262	53.333610	6.537036
53.333881	6.5371259	53.333801	6.537195	53.334108	6.537462	53.333332	6.537222
53.333700	6.5372425	53.333830	6.537246	53.334214	6.537367	53.333626	6.537246
53.333936	6.5372055	53.334042	6.537496	53.333832	6.537336	53.333732	6.536953
Test Location Two							
53.334067	6.5377449	53.333944	6.537501	53.333855	6.537672	53.333889	6.537499
53.334110	6.537360	53.333817	6.537244	53.334203	6.537526	53.334029	6.537239
53.334136	6.5373805	53.334705	6.537285	53.334116	6.537745	53.334028	6.537557
53.334067	6.5375148	53.333917	6.537359	53.334030	6.537533	53.334103	6.537520



The above graphs shows the data for location one, with one data point for each of the four tests. It is interesting to see the main differences in the position of the data points is only in the latitude of the points. This could be due to the environmental conditions at location one. At this location there are houses directly behind and houses across the road in front. This could be restricting coordinates in a direction parallel to the road.

Another observation is that all of the devices scores are clustered around different points. Thus highlighting that different devices will produce different values. Interestingly the spurious result on the far left from the HTC One S occurred when it was the first device used. The Samsung galaxy Tab produced the spurious result at the bottom of the graph when it was first in the ordering. This seems to support the idea of a settling time for the GPS on the mobile device



In this test the variability is clearly in both axes. This could be explained by the environment at location two. There is again a house behind this location, but this time there is a T-junction in front of the camera. Thus this more open environment seems to generate variability in both directions. In general three of the data points are clustered fairly close together with a fourth a larger distance away. In the case of the iPhone and the Sony, these occur when they are the first device used. It occurs when the HTC is fourth and the Samsung third. This suggests a settling time does exist, but there are other factors involved too.

5.1.1.3 Conclusions:

This tests set out to establish if a settling time exists and if it the devices would produce a stable output. The results above, show that a settling time seems to exist, but it is not the only factor in generating spurious results. The test has highlighted the environmental conditions, such as surrounding buildings, appear to have an impact on the stability of the results. This is highlighted by the stability of the results in one axis at location one, but the instability in the results in both axes at location two. The results do seem to cluster around a central point in general. The minor differences could be attributed to a number of factors. These include errors in standing in exactly the same position for each iteration of the test and similar orientation changes.

5.1.2 Test Case Two:

5.1.2.1 Method:

The tests needs to examine how the GPS systems on mobile devices, perform in providing coordinates that allow for a suitable image to be downloaded from Google Street View. The goal is to study the stability and accuracy of the GPS coordinates provided by the devices over a three minute period. This will give representative data of the inputs the application would receive when accessed by a user. For this test two devices were used, a HTC One S and a Sony Xperia Tipo. The tests were conducted at two locations, which are familiar to the author in order for informed feedback to be provided. At each location a number of steps were performed to log the data in a consistent manner and are described below;

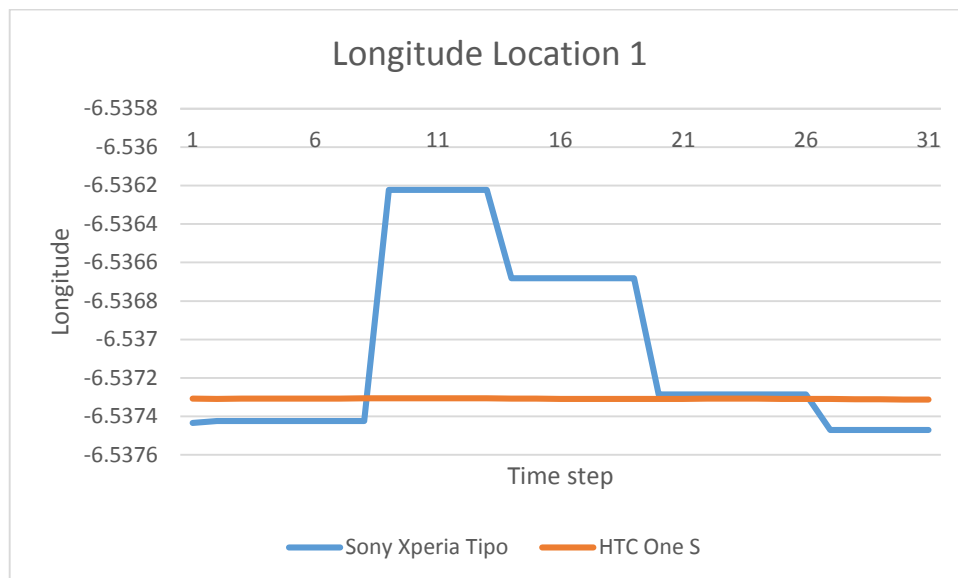
- Move to position one.
- Choose first device.

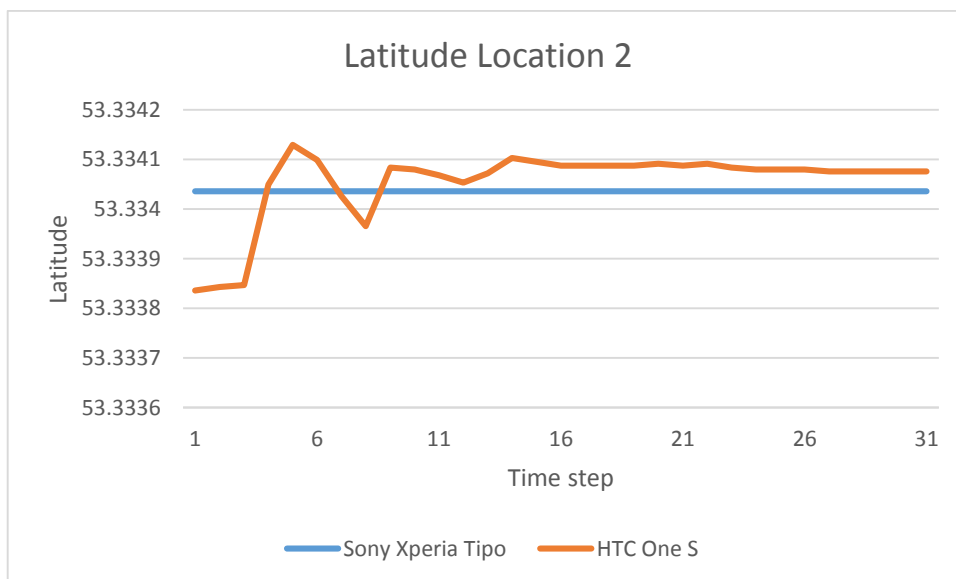
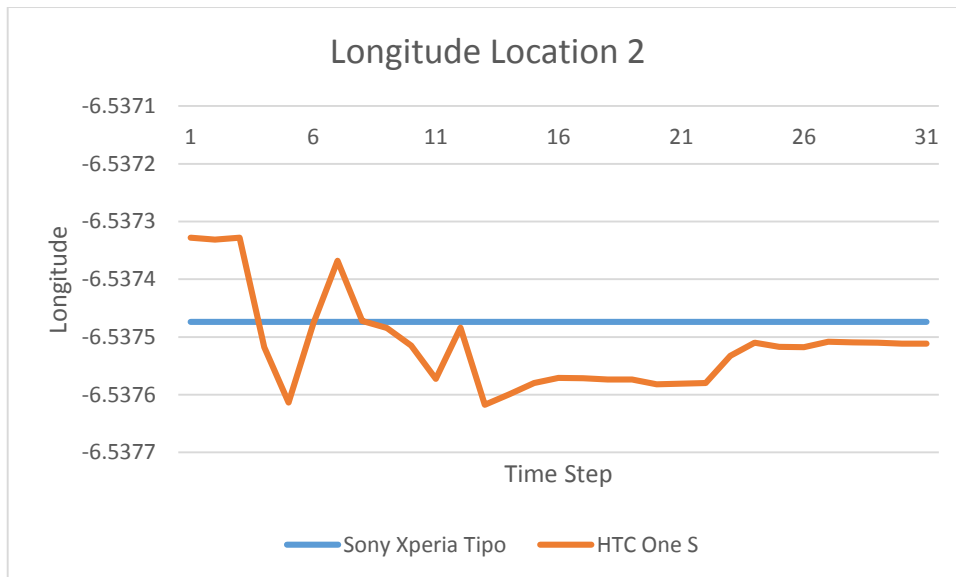
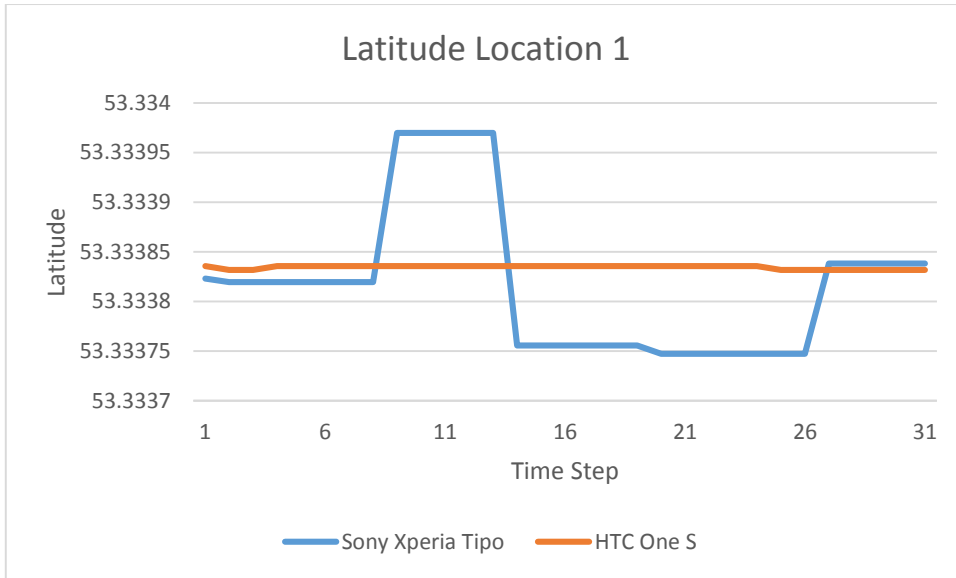
- Start the GPS system on the device.
- Enter the camera application.
- Wait until a GPS location is available.
- Start a three minute timer (on a stopwatch).
- Take a photograph every ten seconds.
- Exit the camera application.
- Turn off the GPS.
- Repeat with second device.
- Move to second location and repeat.

The full data is provided in Appendix B: Supplementary Results data. In the next section an evaluation of the data is provided.

5.1.2.2 Evaluation of Results:

The data points collected over the three minute interval for both devices are presented in the two graphs below. The first graph shows the recorded longitude and the second shows the recorded latitude. The results of the two devices are presented on the same graphs to allow for comparisons in the data to be made.

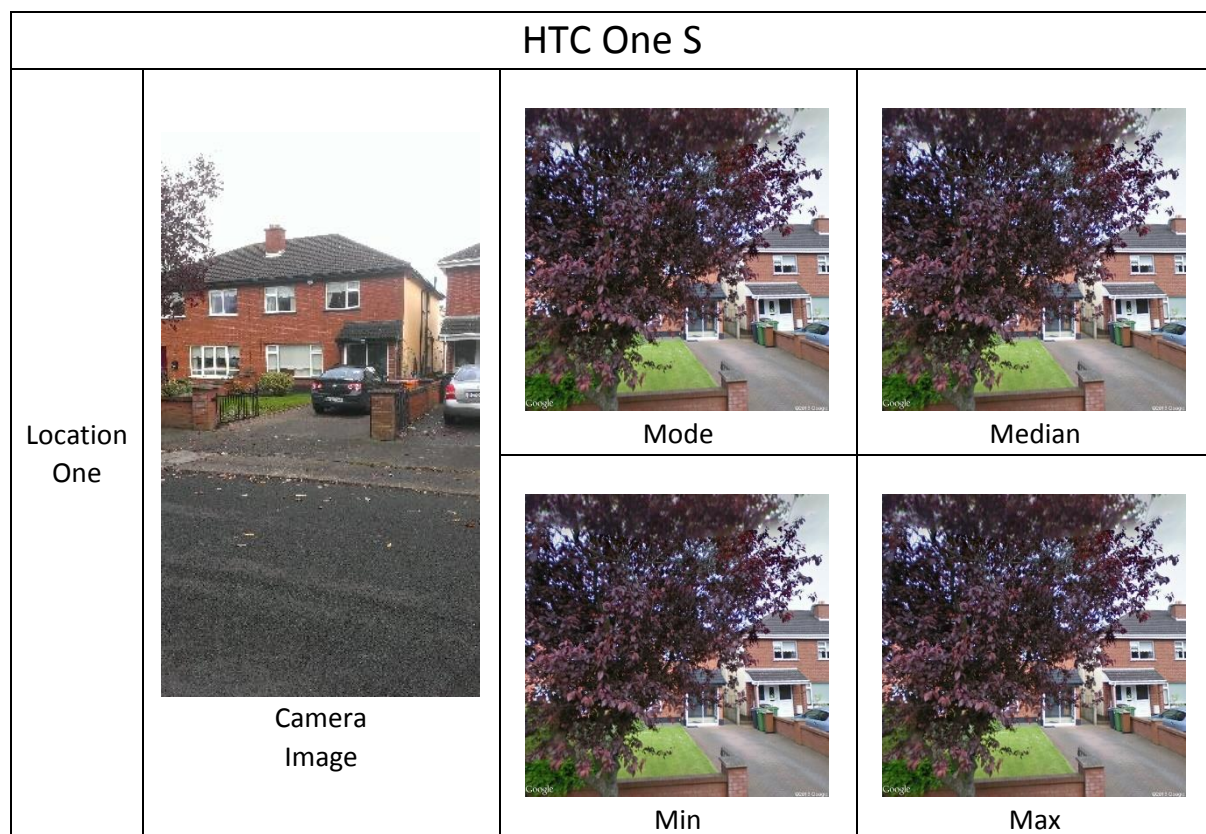









From the graphs above there are a few key findings to note. Firstly it is important to note that neither device provided a perfectly consistent reading across the three minute period. The HTC One S provided much more consistent readings than the Sony Xperia Tipo at location one, but this was reversed at location two.

This test has generated a number of data points, for each device at each location. This data now needs to be compared with Google Street View to establish if it is reliable and if the Google Street View image will contain the same buildings as the image taken with the camera. There are a few issues arising from the data presented. The huge variance seen between the two devices needs to be analysed, to see if the GPS coordinates at the extremes of this variance are acceptable for the application to run successfully. The other issue to examine is whether the GPS coordinates given, actually correspond to the same location on Google Street View.

Below are a series of images based on the GPS coordinates gathered in the test at the two locations. The camera image is shown and a number of images based on the GPS data gathered. There are four images downloaded from Google Street View. These images are using the mode GPS position, the median GPS position, the maximum longitude and latitude values seen and the minimum longitude and latitude values seen. These we chosen to illustrate the average behaviour and the behaviour at the absolute extremities of the variance in the data. The orientation information was not available so this was altered to produce the best result.



The above data highlights a discrepancy between the Google Street View locations and the camera location. The tree that is just visible in the top left corner of the camera image is the tree visible in the Google Street View images. This gives an indication of how big the difference in location is. The saving grace in this example is that by changing the orientation it is possible to see the building in the camera image.

HTC One S			
Location Two	 <p>Camera Image</p>	 <p>Mode</p>	 <p>Median</p>
		 <p>Min</p>	 <p>Max</p>






This set of images is much better than the images above, the mode and median results are almost identical to the Google Street View images. The maximum and minimum value do show significant variation from the actual camera position. The camera location of the minimum image is in front of the house, seen in the camera image and the maximum image has moved further down the side of the building. Again in both cases a change in orientation brings the building back into view. Thus it is possible to recover from these errors.

Overall the HTC One S has provided GPS coordinates that can be successfully used to get an image of the building in the camera image. The drawback is in some scenarios the orientation needs to be edited to achieve this. The interesting thing to note is that in the graphs presented above, the HTC One S provided the most stable coordinates, but it seems it is stable around a point which does not match Google Street Views coordinates correctly.

Sony Xperia Tipo

Location One			
	Camera Image	Mode	Median
			
		Min	Max

These results are very interesting, the min, mode and median images are almost identical to the image taken by the camera. The discrepancy comes in the maximum image, the coordinates are on the wrong street. This image corresponds to a location on the next parallel street to the north of the camera location. This is a huge distance and no amount of changes to the orientation will give a view of the building photographed. As this happened at an extreme data point, it indicates that the application may need to ignore sudden jumps in the input data and ignore new readings until they stabilise again.

Sony Xperia Tipo				
Location Two				
		Mode	Median	
				
		Min	Max	

For this experiment the Sony Xperia performed very well. All of the images, including the extreme GPS coordinates, have given an image almost identical to the one taken by the camera.

From the data above the effect of the variance in the data can be clearly seen through the images representing the extremities of the data variance. The two devices performed very differently at the two different location. As a result it is necessary to examine the two different locations. The first location is at the side of a road on a straight road with trees on both sides and above the position of the camera. There are houses on both sides of the road. The second location is at a T-junction, looking down the road coming off the T-junction. This location has tree cover and houses behind it, but has much more open space in front of it. This is illustrated, along with the large discrepancy seen in the Sony Xeria Tipo, in Figure 34 below.



Figure 34 An image showing the area where the photos were taken. The two locations are highlighted and a third location is highlighted as it is where the major Sony Xperia Tipo discrepancy was located. This is displayed to show how large a jump this corresponds to.

5.1.2.3 Conclusions:

After the above discussions and testing there are a few key points to highlight. There is a discrepancy between Google Street View and any mobile device as Google records images from the centre of the road and at a height, while pedestrians take photographs at head height on the footpath. This is the minimum discrepancy that will be visible. The other issue is no mobile device gives perfectly consistent data over time and this data is subject to severe jumps at times. The redeeming feature of this is the average over time provides a good approximation to the devices current location. This means the application may need to keep an average over a period of time and ignore sudden jumps in position until the position stabilises again. The other corrective measure that can be taken is to search across the whole 360 degree orientation space. As simply changing the orientation was enough to get an image with the same building present in it as is in the camera image.

5.1.3 Conclusions:

In this section the aim was to establish the different sources of errors in the GPS data. Through the two tests above a number of these have been highlighted. The main errors investigated were; settling times, consistency of the data and errors in Google Street View. Through testing another issue was highlighted, the environmental conditions at the location.

The above testing has shown that a settling time seems to exist on most devices, but that this is not the only factor which generates spurious results. With regards stability the two tests have shown that the average value over times is a very good approximation to the current location. The

problem is instantaneous values may be inaccurate. Thus averaging the GPS coordinates over time seems a logical approach to handling this issue.

The Downloaded Google Street view data is generally, sufficiently accurate for the purposes of this application. For all of the cases presented above, where an average GPS coordinate value is used, a change in orientation will yield an image with the desired image present. This means the application should be able to function correctly if it downloads images to cover the full 360 degree range of orientations.

Overall the above results indicate that it is possible to get images from Google Street View that contain the same building as those in the camera image. This means the data for the following stages in the application can be gathered, but with the above considerations in mind.

5.2 Corner Detection:

In the images presented below an input image and an 'overlay' image are provided. The overlay image shows the extracted edge pixels in green. The corners extracted are shown and are colour coded;

- Red: bottom left corner
- White: top left corner
- Purple: top right corner
- Blue: bottom right corner

The example below illustrates the corner extractor's performance on only a few images. A larger set of examples is provided in Appendix B: Supplementary Results data.

5.2.1 Test Case One:

Figure 35 and Figure 36 below, show a sample input image and its corresponding overlay image. This overlay image displays the corners detected and the edges extracted. When examining the input image there is clear distortion in the brickwork due to the camera used. This can be seen as a wave like effect on the entire image. When examining the windows in the input image, most have a bright illumination on the right hand side building. This is due to weather conditions on the day, it is interesting to note that five windows in this building are not illuminated in this manner; the four on the bottom row and leftmost window in the second row from the top of the building.

Examining the overlay image, these five windows have edges which are significantly less well defined than the others. This has led to some of the outline of these windows being omitted in the edge image. As a result it is much less likely that the application will successfully extract these window.

When examining the corners detected by the application the windows in the building to the left have had all the desired corners found for every window. On the building on the right hand side of the image, the corners of the windows have been found for all of the windows with the bright illumination. Of the five windows without this illumination only one has had the four corners of its outline extracted. This is the window third in from the left in the bottom row. The other four windows are missing edge pixels at the corners. Missing edge pixels will result in corners being lost. It is interesting to note in the second window from the right along the bottom, there is a small area of strong illumination on the window pane and this has artificially created an edge halfway through the pane in the overlay image.

While the results look very promising there are a number of incorrect corners found. For example a number of the illuminated windows have corners detected in the middle of the window, where the two panes of glass join. The corner detection method is designed to illuminate these, but the T-junction shape was not detected. There is one main reason for this, perspective distortion. In most cases these corners in the middle of the window are located where the upward or downward

search would miss the actual edge of window. This is as a result of the edge being a few pixels left or right of the pixel identified as a corner. This is due to the line being rotated, as a result of the perspective distortion and is not directly below the pixel.

At the bottom of the image on the left hand side, there are some spurious results. These results are generated by a series of straight lines which are very close together. This simply represents random corners detected in a noisy environment. Thankfully they are located away from most of the windows and should not interfere in the subsequent window extraction process.

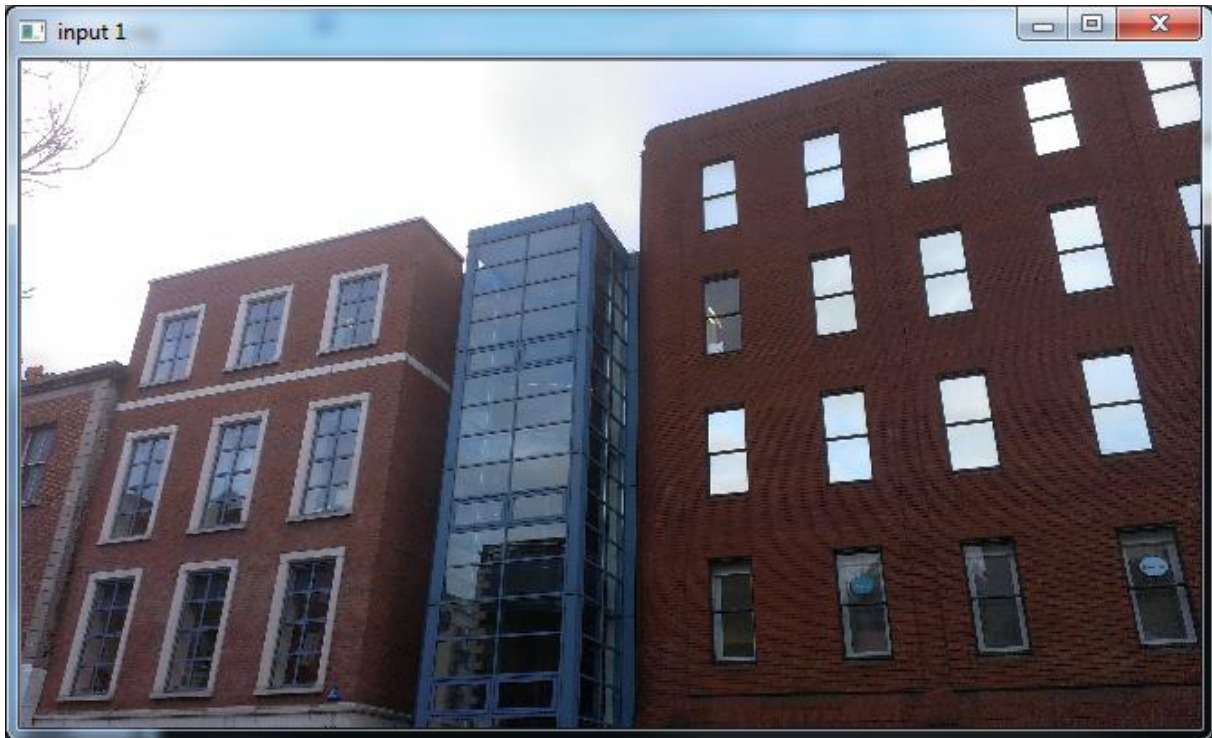


Figure 35 The input image to the corner detector.

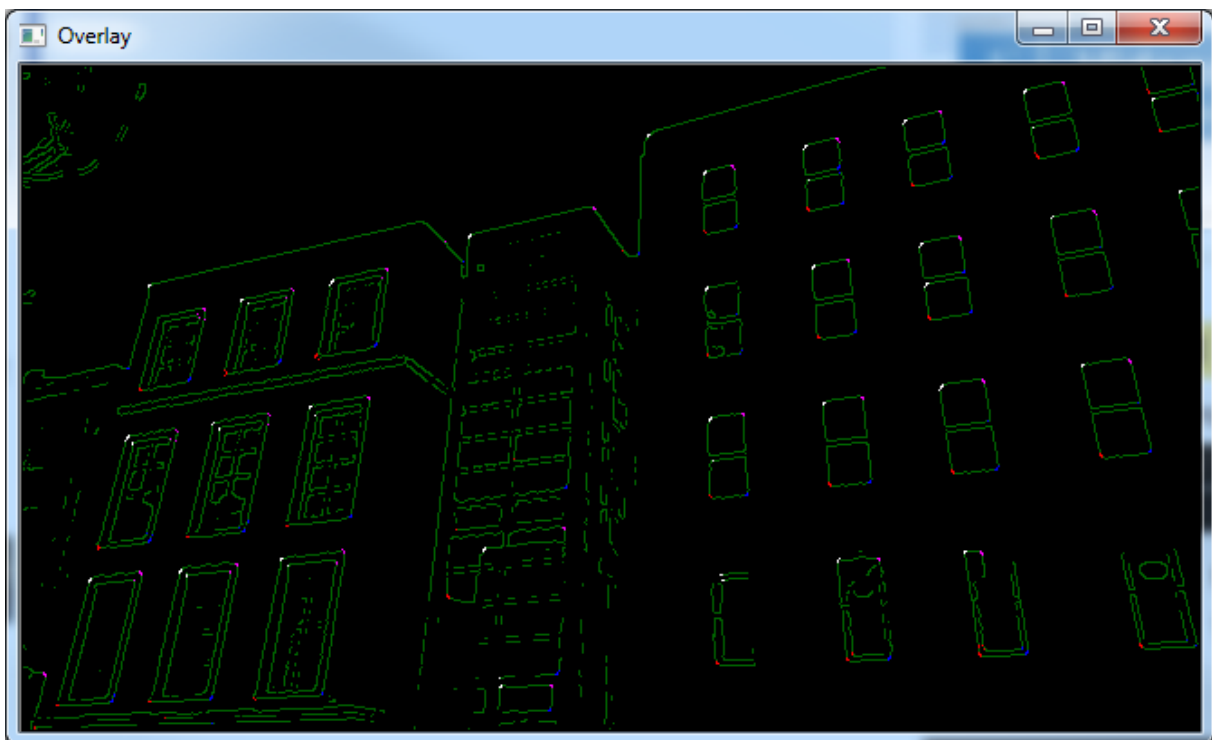


Figure 36 The overlay images for this location. Notice the corners, highlighted by colour.

5.2.2 Test Case Two:

Figure 37 and Figure 38 below, show a new set of input and overlay images. This input image was downloaded from Google Street View. The camera distortion seen in the first test case is not present in this image, but a much larger problem exists. At this location Google have stitched a number of images together. This has resulted in the top story of the building being shifted to the left. The stitching occurs across the second row of windows from the top and removes almost any chance of the application finding them. The image above the stitch line is blurred compared to the rest of the building, which may also effect performance of the application.

As can be seen in the overlay image below, the edges extracted from this image are quite noisy. There appears to be double edges detected, where the edge of the window consists of two edge lines with a gap between. This occurs as a result of the dark regions around the windows, which generate an edge where this region meets the wall and again where this region meets the window pane.

For the top three rows of windows the corner detector has found the corners according to the definition it is based on. The top row, or at least what parts of it remain, has had almost all of the corners present detected. The second and third rows have produced good corners, but have struggled with an unforeseen shape. At most of these corners where the double edges join, a rounded edge joining the two parallel lines has appeared. This produces a lot of edge pixels in a very small region and thus the counter values will potentially be high in every direction. As a result of this the corner detector has failed in some, but not all of these cases. These edges will not be suitable for window extraction as the line following algorithm will have too many misses or have too many option and not find the corner on the opposite side of the window.

The area of the image below the third row of windows is very noisy. The edge pixels generated are vague and indistinct. There are a number of reasons for this, firstly there are illumination changes in the windows. This is due to shadows from the flower baskets and the pillars beside the windows. The second issue are the flower baskets and the people. The flowers are irregularly shaped and cause a large amount of spurious edges. The people cause extra shadows and random edge pixels. The net result is a region of noisy edge pixels and the corner detectors performance is accordingly poor. This is to be expected as any system given poor inputs will produce poor results.

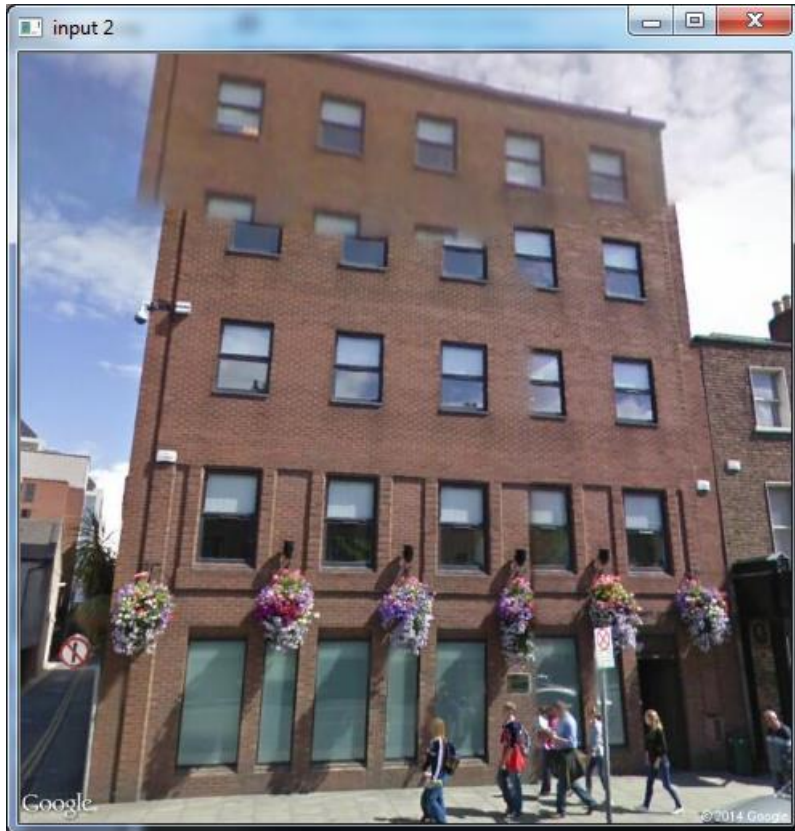


Figure 37 The input image, this image is taken from Google Street View, notice the effects of stitching on the image. Some windows have been shifted and blurred.

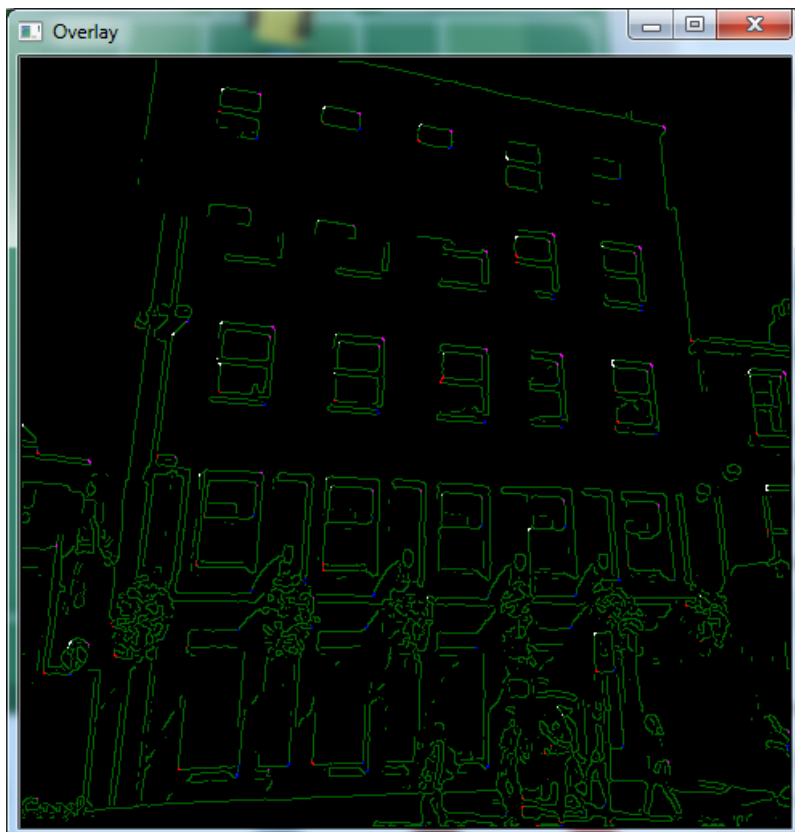


Figure 38 The overlay image, the corners are highlighted along with the edges. Notice the effects of the stitching seen in the input image.

5.2.3 Conclusions:

The corner detector has performed very well according to the definition of a corner. Figure 36 above, demonstrates how the corner detector can perform very well when presented with different types of windows. The corners of the windows that have noisy interiors were correctly extracted, such as those on the left hand side of the image. The only problem being the noise led to both very wide regions and very small regions for the corners. This could be troublesome in the next section. The second type of corners found were those for windows with very crisp edges, as seen on the right of Figure 36. The only issue here being the corners detected in the middle of the window, where the two panes meet. The major cause of failure in the corner detector is noisy edges. This was illustrated in Figure 38, where the bottom of the image has vague and random edges. Thus for the corner detector to succeed the pre-processing of the image needs to reduce the presence of noisy edges.

5.3 Window Extraction:

Below are number of different pairs of images. Each pair of images show the corner image and an image displaying the successfully extracted windows. The image displaying the extracted windows is a mock up displaying the results. The actual application highlights the extracted windows by colouring the four corner pixels red and setting the pixel at the centre of the window to red. If the reader wishes to run the application, provided on the attached CD, this is how extracted windows are presented. For this section the windows extracted are shown in green with the remaining edges shown in white.

5.3.1 Test Case One:

The image used is the same as that used for test one above. This test shows what windows have actually been extracted, based on the overlay image presented in Figure 36 above. The application found eighteen windows out of the twenty five windows present in the image. This represents a success rate of 72%. The other important observation is that no edges were deemed to be windows, which in reality are not windows. Thus the false positive rate is zero.

The four windows not extracted on the right hand side building, are the four windows with low illumination. These windows produced poor edge results and are missing some corners. Thus the failure of the application to extract them lies with the edge detection results. For the three windows not extracted on the left hand side building, there are two reasons for the failure. The two windows on the bottom row were not detected because the bottom right corner is not located along the line of the bottom of the window. They are located a short distance along the right hand side of the window. This would cause the horizontal search to miss the corner. The window top left of this building was not extracted as the top left corner was not extracted in the previous stage. This means the window extraction process has no chance of associating a window pattern with this group of edge pixels.

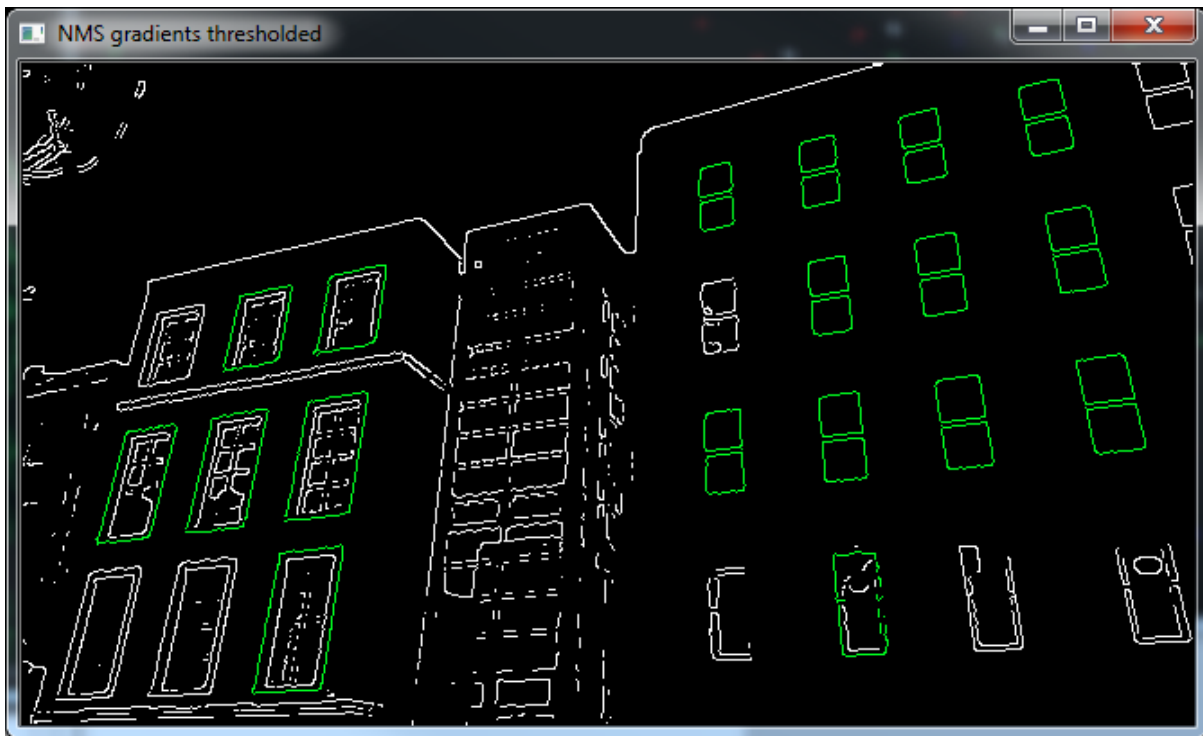


Figure 39 The extracted windows are coloured green with the other edges presented in white

5.3.2 Test Case Two:

This section also uses the same input image as test case two above. This image shows how damaging Google Street Views use of stitching is to the end result. Two images are shown below, Figure 40 and Figure 41. Figure 40 is the same image as used in case study two above and Figure 41 is an image taken on a mobile device of the same building. The contrast in the resulting number of windows extracted is used to highlight the effect of the stitching.

In Figure 40 below, there is only one window extracted from a building with twenty four windows. This is low enough to render the end result null and void. At this point the application would simply not have enough information to proceed to the matching stage. The reason for the failure to extract more windows was highlighted in test case two above. The corner detector failed to detect the four corners of most windows because of the distorted and blurred windows at the top of the image and the noisy windows at the bottom of the window.

To highlight the extent of the problem caused by the Google Street View stitching Figure 41 below, offers a view of the same building but through a camera. This means the edges extracted for the windows in the top three rows are much clearer and well defined. The net result is that of the fifteen windows in the top three rows, eleven were successfully extracted, representing a 73% detection rate. The camera image only shows the top four rows of windows, omitting the bottom row. This yields an overall detection rate of twelve out of twenty windows, or 60% success rate. This includes the noisy windows which were discussed above. Again the application had no false positive detections.



Figure 40 The extracted windows are coloured green with the other edges presented in white



Figure 41 A contrasting example to highlight the effects of the stitching employed by Google Street View

5.3.3 Conclusions:

The window extraction method is not perfect. Between the two tests the success rate in realistic condition was about 72%. But this is only representative of images which have single edges, with no noisy edge points along the outline of the window. The examples above show that this is not an unrealistic expectation.

The biggest problem with this section was Google Street Views stitching. It effectively removed ten windows from the building. This is because the stitching line went through them and blurred the remaining windows. This is a very serious issue, as it could render many other locations impossible to match, if the input image is as distorted as seen here.

5.4 Façade Matching:

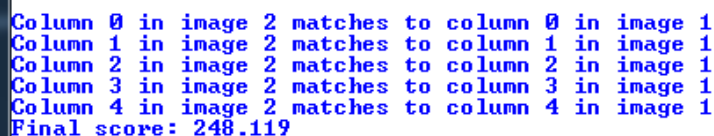
In this section the façade matching algorithm is put to the test. Two test cases are examined, but only one of these follows on from the test cases above. Test case one is continued but test case two is swapped for a new example as the data extracted was not sufficient to proceed any further.

The test data is broken into two parts, the column groupings and the subsequent window matching. The application highlights columns by displaying the centre point of the window with red centres at the top of the column and green for subsequent window. For illustration purposes these results have been converted to the images shown below. The end result and window matching is output in text format and have also been converted to the images below for illustrative purposes.

5.4.1 Test Case One:

This test case has so far proved quite successful with eighteen of the twenty five windows present extracted. Figure 43 below, shows the image used up to this stage, the image from the camera. In Figure 43 the extracted windows are colour coded by column. There are seven different columns of windows in the image. Figure 44 below, is the corresponding Google Street View image. This image has been processed in the same manner as the camera image. In Figure 44 there are some issue from the stitching employed by Google Street View. This has had a very bad effect on one window, shifting the top to the left of the bottom half of the window. There is also an extra line of edge pixels in the green column. The columns are number zero to N from right to left, e.g. column zero is the red column, one the blue, two the green and so on. The images are labelled image one and image two in the output, the camera image, is image one and the Google Street View image, is image two.

The result is given below in Figure 42. It can be seen that the columns have been matched correctly.



```
Column 0 in image 2 matches to column 0 in image 1
Column 1 in image 2 matches to column 1 in image 1
Column 2 in image 2 matches to column 2 in image 1
Column 3 in image 2 matches to column 3 in image 1
Column 4 in image 2 matches to column 4 in image 1
Final score: 248.119
```

Figure 42 The result at the end of the test.

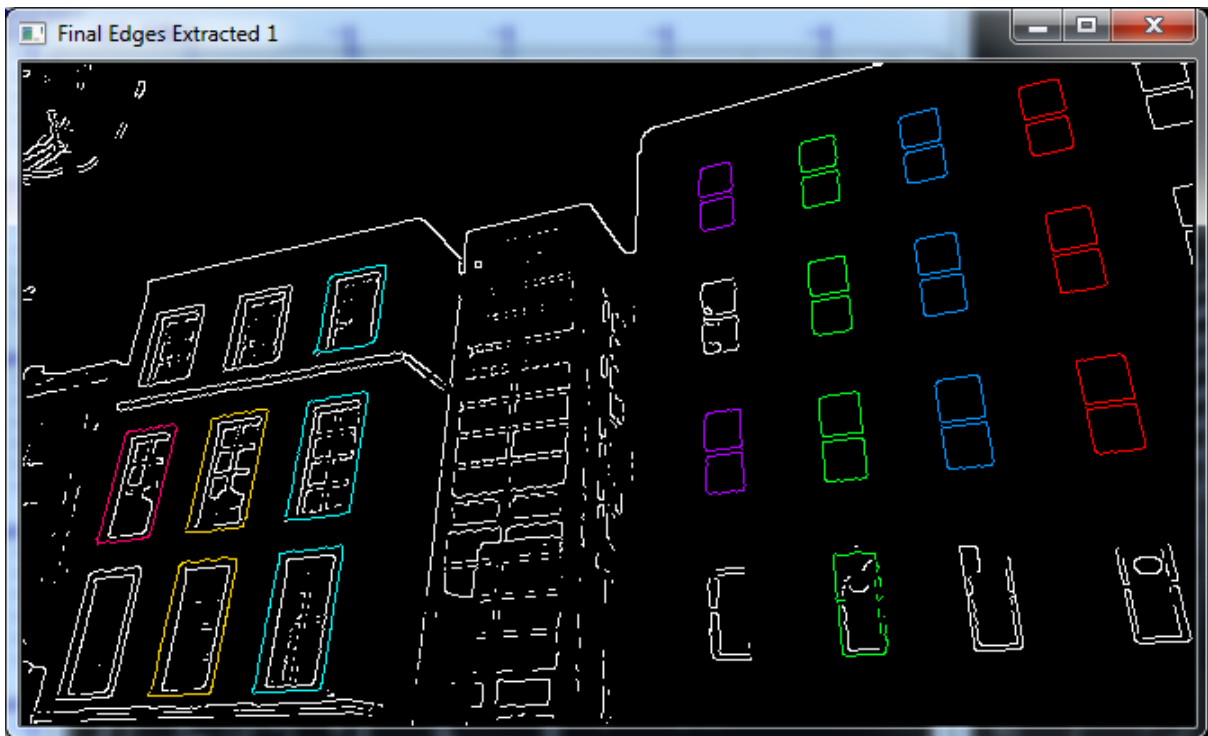


Figure 43 Image from camera, the extracted windows are colour coded by column

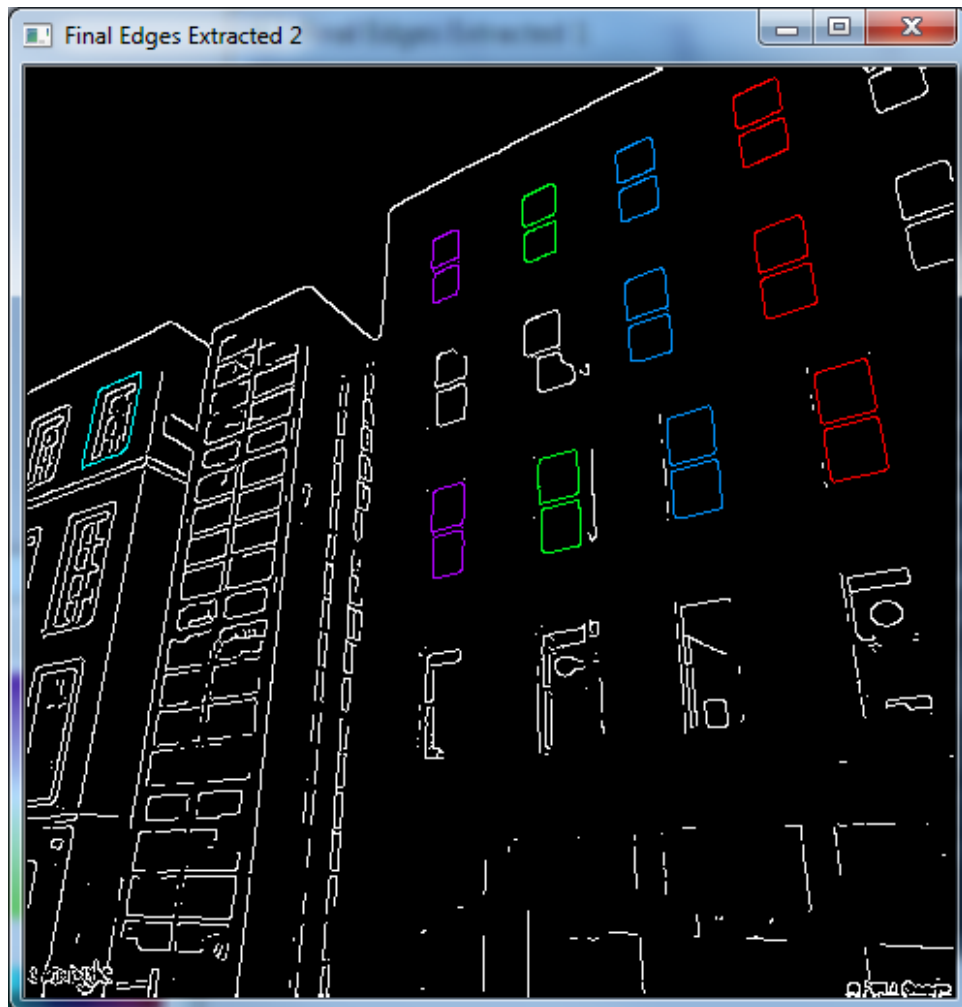


Figure 44 Image from Google Street View, colour coded by column

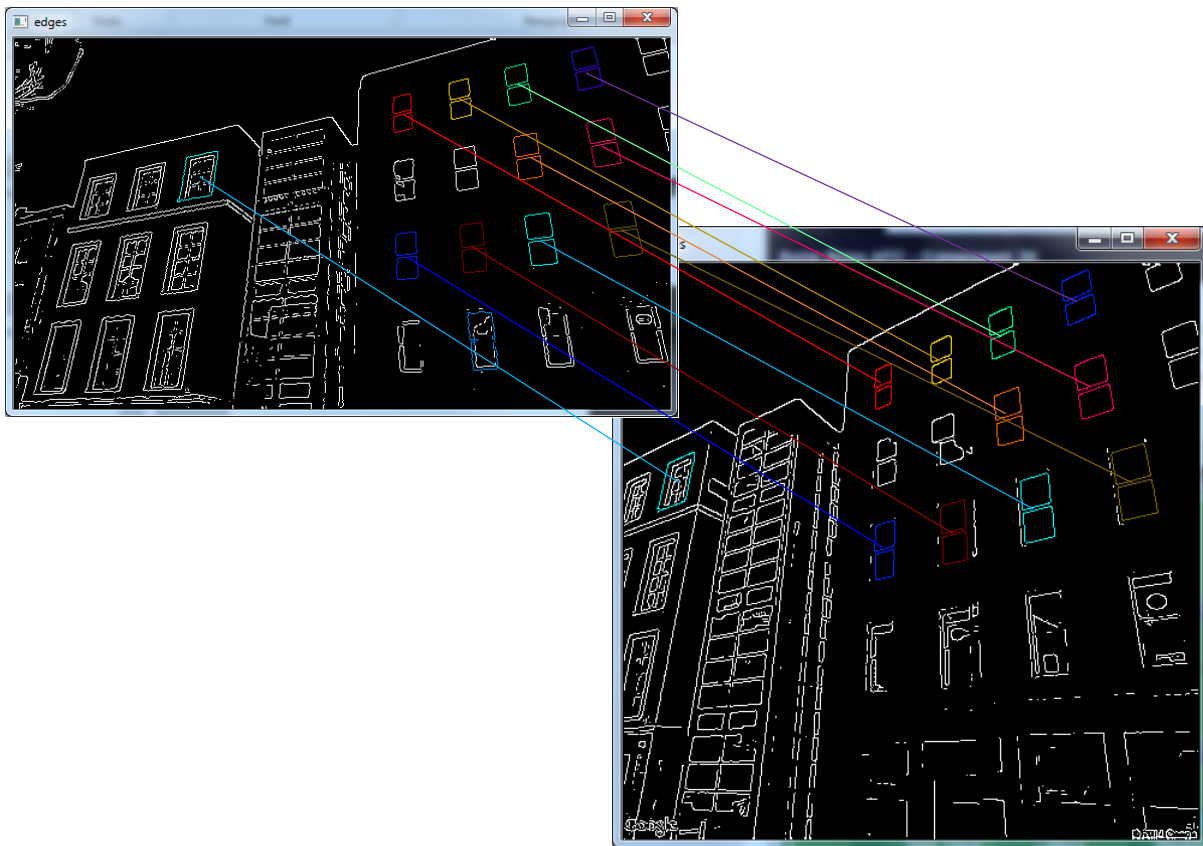


Figure 45 The image above displays the matching windows

5.4.2 Test Case Two:

The image set used for the previous test case two example, was abandoned after it failed to yield sufficient extracted windows in the Google Street View image. As a result, for this example a new set of images will be used.

Figure 47 below, shows the camera image used, with extracted windows colour coded by column. Of the nine windows in the façade, eight have been successfully extracted, a success rate of 88%. In Figure 48 below, another camera image is displayed. Again the extracted windows have been colour coded by column. The application extracted all nine of the windows and also the inner outline of the window in the bottom left of the image. This is because the inner outline conforms to the definition of a window, but since the two are overlapping it would be preferable if the inner window was discarded. In this case the window numbering of the columns has been reversed, due to the change in the direction of the perspective distortion. Thus the red column is column zero, the green number one and blue number three.

The result is shown below in Figure 46. The application has successfully matched the columns together. The windows matched are shown Figure 49, all windows found, with a corresponding window in the other image has been matched. The application extracted the bottom right window in the second image, thus it has no pair. Another point to note is how the bottom left window in image two has both the inner and outer outlines of the window classified as a window. When matching occurs the outer outline of the window is matched to the corresponding window in image 1. This test is very successful using two camera image, thus highlighting the method can handle a small amount of missing information and still produce correct results.

```

Score Matrix:
215.819  124.633  12.3399
116.079  215.819  124.633
42.0793  116.079  215.819

Score Matrix:
0        -1        -1
-1       215.819  124.633
-1       116.079  215.819

Score Matrix:
0        -1        -1
-1       0        -1
-1       -1       215.819

Score Matrix:
0        -1        -1
-1       0        -1
-1       -1        0

Column 0 in image 2 matches to column 0 in image 1
Column 1 in image 2 matches to column 1 in image 1
Column 2 in image 2 matches to column 2 in image 1
Final score: 215.819

```

Figure 46 The result at the end of the test. The score matrix is displayed. It can be seen how the largest value is selected and set to 0 and all impossibilities set to -1.

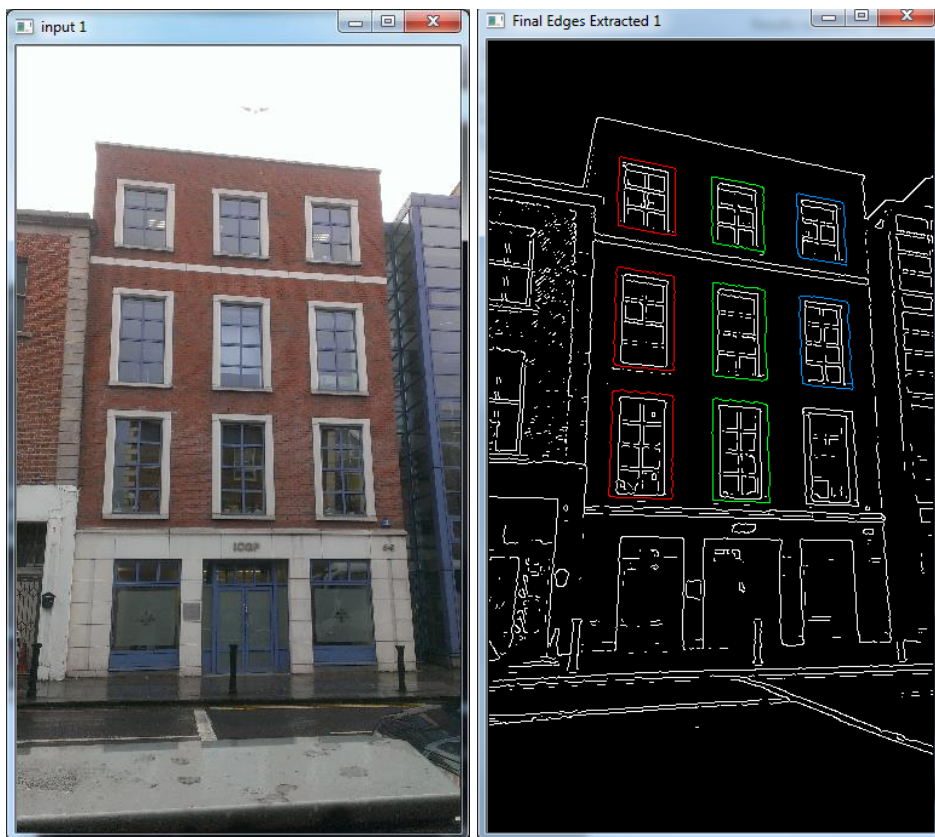


Figure 47 The camera image, with extracted windows colour coded by column.

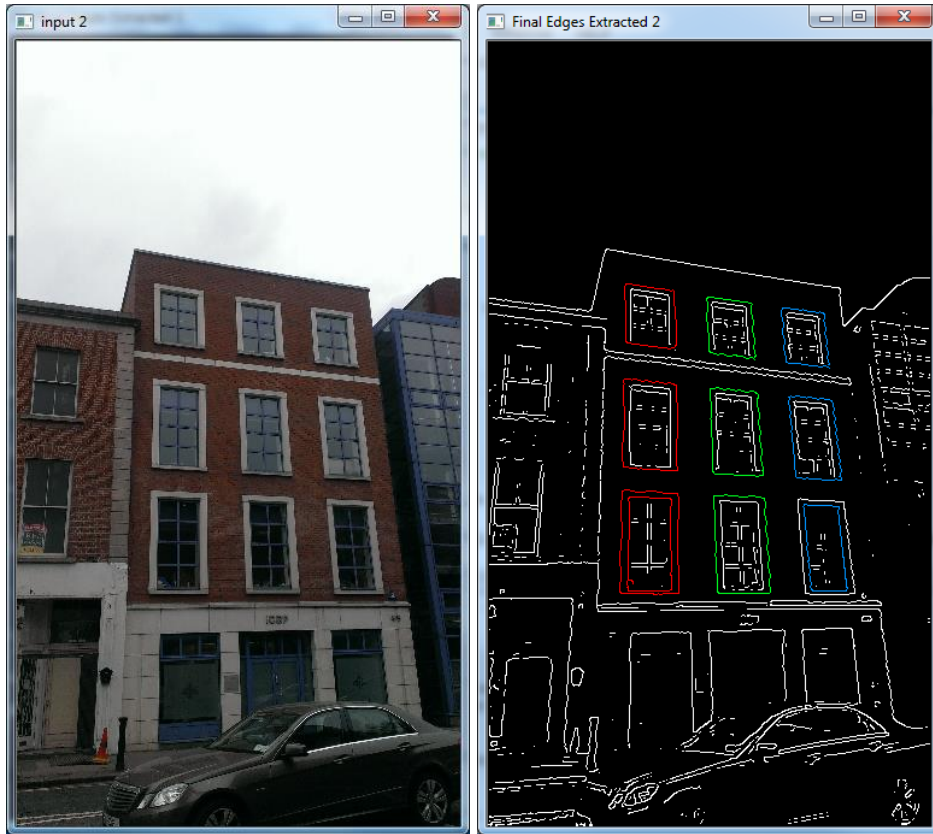


Figure 48 The second camera image, with extracted windows colour coded by column. Notice how the bottom left window has the inner and outer edges identified as windows. Also note that the bottom right window has only the inner edges extracted as a window.

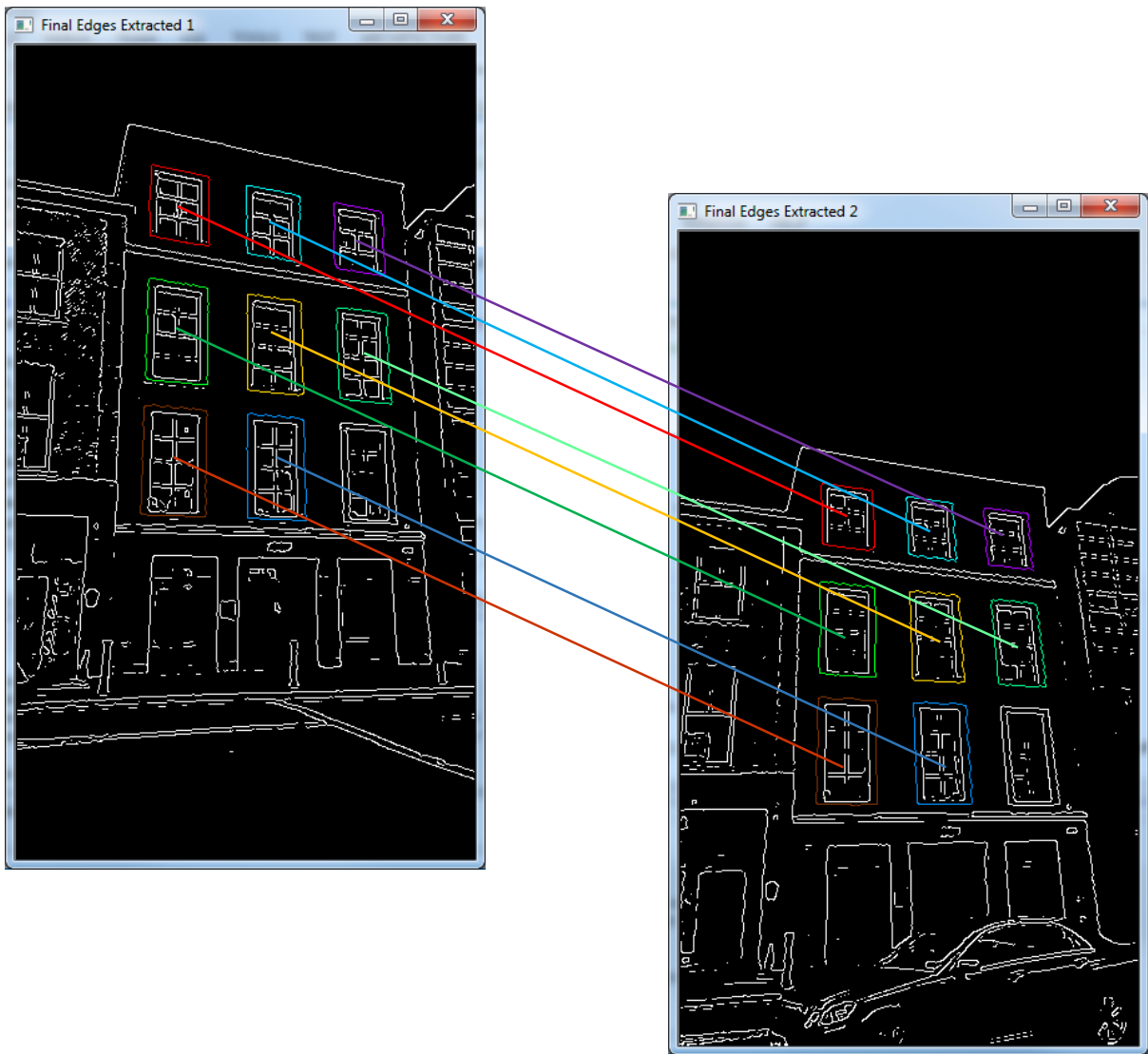


Figure 49 The above image colour codes the matching windows extracted in both images. Note the bottom left window in the second image. While the inner and outer outlines were classified as windows, only the out window generated a match with the outer window in the first image

A second examination is presented below. The first image below is also compared to a Google Street View image. This image has been significantly distorted by stitching and as a result significantly less windows have been discovered. The outcome is a successful match, but not with the level of certainty required. Note that in this case the Google Street View image has had its columns labelled zero, two and one from left to right, this is a result of the distortions through the centre column of windows.

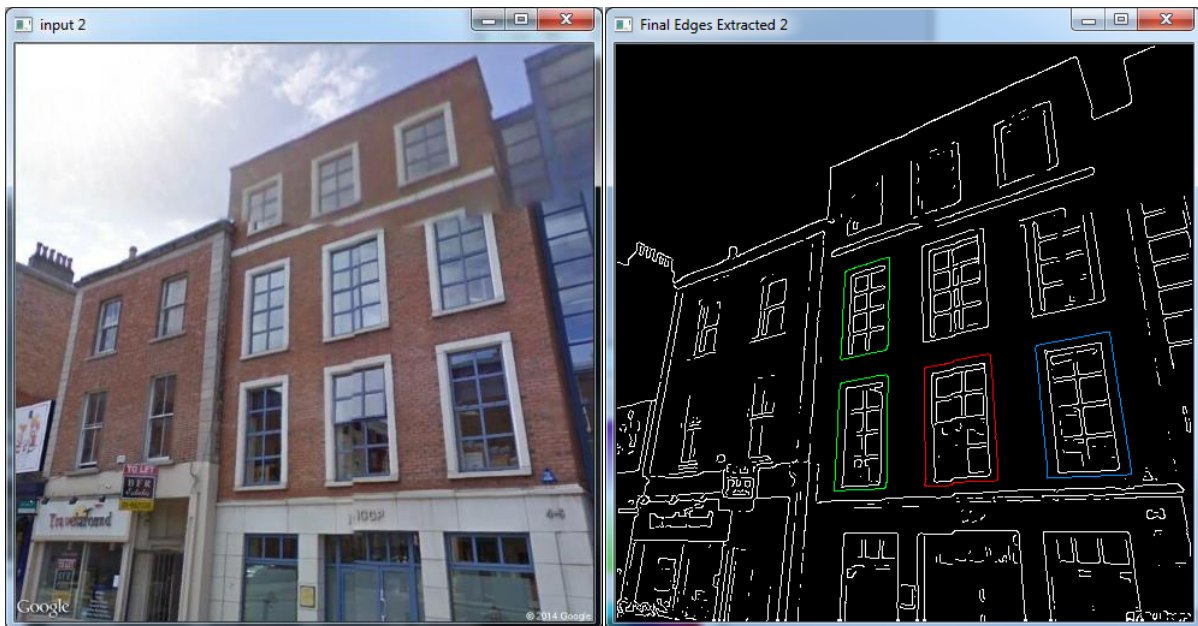


Figure 50 The Google Street View image, with extracted windows colour coded by column

The result of this test is shown below;

```
Column 0 in image 2 matches to column 0 in image 1
Column 1 in image 2 matches to column 2 in image 1
Final score: 88.8505
```

The application has matched the two leftmost columns and the two middle columns, but the overall score is very low.

5.4.3 Conclusion:

The above test illustrates that the matching algorithm is very successful in matching when the majority of windows are found. In this case it was able to handle effectively two windows not having a match in the second image. One of these was caused by a window not being extracted and the second scenario involved an inner outline of a window being classed as a window. This represents a false positive in the window extraction process. Thus overlapping windows need to be removed, but care needs to be taken in removing the right windows when this occurs.

The last test in test case two has shown the penalty for missing windows in one image is quite severe. The algorithm needs to strike a better balance between highlighting the missing windows and showing how strongly the extracted windows match. Due to time constraints, this is left as future work.

The biggest problem in this and every other section is the stitching of images by Google. It distorts the images by shifting windows positions, introducing new edges, blurring details and distorting window shapes.

5.5 Conclusions:

The three main stages in the building matching algorithm have been tested above. These are the corner detector, window extractor and façade matcher. There were a number of issues which caused problems at each stage and as each stage builds on the previous stage, errors propagate through the application and effect the end result.

The corner detector is very successful. It can reliably extract the corners, as defined, when presented with clean edges to work with. The errors associated with the corner detector arise from

two sources; perspective distortion and noisy inputs. The perspective distortion can stop the corner detector from finding pixels in a given direction. This then leads to T-junctions being interpreted as corners. The noisy input can have two effects, if the region is noisy and filled with random edges then the detector will find random corners. This is the expected behaviour. The issue arises when the window has more than one line defining its outline, with multiple lines running parallel. When these lines turn and join at the end, there is such a high concentration of edge pixels in the area, the corner detector can generate random results, or no results.

The window extraction process is limited by the quality of the corners detected and the edge pixels defining the outline of the window. This is an example of where the errors from an earlier part of the application can propagate through. If either the edge extraction or corner extraction processes have errors, they will affect the window extraction process. In the main, the window extraction process was very successful. It found 70%+ of windows in the regions it would be expected to succeed in. Although this number is not reliable as it only represents three tests, the results are encouraging.

The matching algorithm is not perfect, and it is again affected by errors propagating through from the previous stages. Overall it has proved quite successful in matching buildings where the majority of windows have successfully been extracted. The problems arise when there are very few windows extracted in one image compared to the other. The algorithm penalises missing windows quite severely. This penalty needs to be adjusted to place more emphasis on the windows actually present. The algorithm struggles to associate windows in matched columns if there are not very many of them. This is because of the lack of positional information available if there is only one. Thus it may be necessary to consider rows of windows as well as columns. This would provide extra positional information for use in score calculations.

Overall the corner detector and window extractor performed to a high standard. The matching algorithm is quite successful, but improvements could be made when handling missing windows. The testing has shown how errors occurring in the early stages of the application, have effects at all subsequent stages. The biggest problem across all three stages is the stitching methods employed by Google Street View. If this project was to be continued, it would be necessary to contact Google directly and see if access could be made available to the unstitched image data. This data would most likely only have views of buildings close to the camera, but removing the stitching issue would allow for better results to be garnered. While the above examples illustrate the successes and issues with the current implementation, more results are presented in Appendix B: Supplementary Results data for further examination.

5.6 Limitations:

The application, in its present state, makes a number of assumptions, which limits its performance. The description of a window is the main limitation in this regard. The description of a window is quite basic and does not consider information about the shape or interior layout of the window. The shape defined, restricts the type of windows which can be found, for example windows with arched tops will not be extracted. The interior layout of windows differ quite significantly, some windows contain a single pane of glass, while other contain complex patterns with bars running vertically and horizontally through the window. Using this information in the description of a window would allow for more detailed comparisons and improve the matching process.

The overall building description method employed, reduces the application effectiveness on some buildings and landmarks. For example glass fronted buildings and landmarks with no windows will not be successfully extracted.

The edge detection methods used can fail to extract the outline of the windows in certain scenarios. If the outlines are not correctly extracted the corner detection and window extraction

stages will fail. Some of the reasons for these failures are; illumination, colour and obstruction. Illumination can cause brighter and darker regions within a single window pane, this can create artificial edges within the window. Dark regions can cause more problems than bright, as this can lead to the colour of the window pane matching the colour of the surrounding walls. This means the contrast in colour does not exist between the window pane and the wall, thus the edge detector will not extract the edges along the outline of the window. Obstructions such as street lighting, ornamental hanging flower baskets and people in the images generate extra edges within the image. If these edges overlap or interfere with the outlines of the windows, this has the potential to interfere with the corner detector and window extraction processes.

The limitations presented above can be broadly grouped into two categories; assumptions and environment variance. The application assumes buildings will have windows and these will conform to the definition of a window provided. This definition could be expanded to include more shapes and to include data about the interior layout of the window. Variability introduced by the environment can also have a negative impact on the application's performance. Changes in illumination across a window pane can introduce spurious edge pixels. As can obstacles in the scene such as people, cars and ornaments. While the limitations of the application due to assumptions can be addressed by changing the application, the environmental issues will always be present.

6 Final Word:

6.1 Future Work:

The details presented above represent the stages of the application that have been implemented. Due to time constraints there are a few sections of the application, which are unimplemented. Examining the results above, some of the stages also need refinement before the application would be of a releasable standard.

Small refinements could be made to the corner extraction process. The process for identifying the exact corner location is not very accurate. The corner can be located on an edge near the corner, to give greater accuracy this should be reduced to locate the corner at the exact intersection point of the two lines.

The building matching algorithm needs refinement. The main issues seem to be the method of penalising missing windows and inaccuracies when a column has only a single window. There are a number of different approaches that could be made to improve these, but all require significant implementation and testing time. Currently the application scales the score of a column by the ratio of the number of windows matched to the total number of windows in the largest column. This means if a column with three windows matches a column with four windows, the maximum score possible is 75%. This penalty is too harsh and needs to be re-evaluated. To help with matching windows within columns with only one window, it may be helpful to examine the rows of windows as well. This approach gives the positional information that is missing, basing the positional information off windows in other columns. This would help to stop the case where a window near the top of the building in image one, matches a window near the bottom of the building in image two.

The application is not fully finished, the extraction of the address and the annotating of the input image for display to the user are as yet unfinished. These are the two final steps to be implemented in the application. Due to time constraints it was not possible to investigate these final steps. There are two main tasks involved in the final steps; identification of an address based database and filtering the data for display. A database or repository of information about buildings or landmarks based on addresses, needs to be identified. It is from such a repository that the information to be annotated onto the input image will be downloaded. Before the downloaded data is displayed, it will need to be parsed into a presentable format. This would include links to external sites and ranking the information in terms of importance.

6.2 Conclusions:

The task of building identification in computer vision is not a new one. There are many applications where identifying a building can be key to the applications performance. The majority of these applications involve a user navigating an urban environment. This means the application needs to be able to traverse the environment with the user. Most previous attempts have used back end applications to perform the actual image analysis and a mobile device to provide input and display output. This project examines the feasibility of performing such image process on the mobile device and removing the need for the back end system.

To identify the building in a previously unseen image, it needs to be compared with an image containing a known building. This requires a database of images with labelled buildings, which can be used to compare with the unseen image. In previous systems this database was specifically generated by the owners of the system. It was constructed in such a way as to be stored in the most convenient manner for comparing with the input images. This required the system operators to collect the data, design and construct the database and perform the pre-processing on these images.

This project aims to remove this overhead by using a publically available source of images, Google Street View.

To match the buildings in two images, a description of what the building looks like is constructed. The method used in this project is to describe the building façade by the windows present in the façade. The façade is described by the size, shape and relative positioning of the windows. To achieve this, the process was broken down into a number of different stages; corner detection, window extraction and façade description and matching.

To extract the windows, the application is provided with a definition of what constitutes a window. This description is a rectangular region defined by four corners, which are linked by edge pixels. There are four different types of corners and these must be in the correct order. A corner detector has been developed to extract these four types of corners. It works by searching for lines of pixels in the vertical and horizontal direction which pass through a given pixel. This yields four counter values, one for each direction; up, down, left and right. Based on these counters the corner type can be defined. The process of extracting windows from an image after these corners are identified, is based on searching for a pattern of four corners (one of each type) linked by edge pixels, which outline the window.

To describe the building façade, the size, shape and positioning of the windows is used. The windows are grouped into columns. The windows are described by their aspect ratio and a distance ratio indicating how far away they are from other windows in the column. The columns are related by the distance ratio between them. To compare two buildings in different images, the above information is used. The matching process is divided into two stages, the first is column matching and the second is window matching. The first stage involves, searching for the layout of columns present in the first image, in the second image. This means finding groups of columns that are similarly positioned relative to one another. Once this layout has been extracted, the layout of the windows within the matched columns is examined. This is based on the aspect ratios of the windows and the distance ratio between them.

Through testing the different elements of the application a number of conclusions can be drawn about the suitability of the application. The corner detector has performed very strongly, but is depended on clean edges being detected. Noisy edges cause problems and if they are located near a window corner or outline, the corner detector is liable to miss the corner. The window extraction method proved quite successful in suitable conditions. The method requires single lines of edge pixels to join the corners together, if this is the case the performance is good. The matching of buildings between images was not as successful as the other sections.

The biggest problem seen was the distortion of the images by Google Street View. Google Street View stitch a number of images together to create Street View. During experimentation it was discovered that the algorithm used for this is imperfect. The image above the stitch can be blurred compared to the image below and the stitch line can shift upper section relative to the lower. The end result is that windows become distorted beyond recognition and are impossible to extract. This is the biggest issue facing this application into the future.

The results show that the corner detector and window detector are suitable for this application. The matching algorithm needs refinement but shows promise. The next step in taking this application further is to port the implementation to a mobile device, download address based information and display it, overlaid on the input image.

7 Bibliography

- Chen, D. M., Baatz, G., Koser, K., Tsai, S. S., Vedantham, R., Pylvanainen, T., . . . Grzeszczuk, R. (2011). City-Scale Landmark Identification on Mobile Devices. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Providence, RI.
- Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *Conference on Computer Vision and Pattern Recognition*, (pp. 886-893).
- Dawson-Howe, D. K. (2012, September). *Edge Notes*. Trinity COLlege Dublin. Retrieved from mymodule.tcd.ie.
- Galvez-Lopez, D., & Tardos, J. D. (2011). Real-time detection with bags of binary words. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 51-58).
- Google. (2014, March 28). *About Street View*. Retrieved from Google Maps: <https://www.google.ie/maps/about/behind-the-scenes/streetview/>
- Harris, C., & Stephens, M. (1988). A Combined Corner and Edge Detector. *Alvey Vision Conference*, (pp. 147-152).
- Heaston, P. (2014, March 29). *2-Point Perspective: Understanding SPace*. Retrieved from [craftsy: http://www.craftsy.com/blog/2013/06/2-point-perspective/](http://www.craftsy.com/blog/2013/06/2-point-perspective/)
- Hough, P. V. (1962). *United States of America Patent No. US3069654 A*.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Vol 2.*, (pp. 1150-1157).
- Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), 91-110.
- Matas, J., Galambos, C., & Kittler, J. (2000). Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding*, 119-137.
- McClean, E., & McDonald, J. (2013). *An Augmented Reality System for Urban Environments using a Planar Building Facade Model*. Maynooth: NUI Maynooth.
- Robertson, D., & Cipolla, R. (2004). An Image-Based System for Urban Navigation. In *IN BMVC* (pp. 819-828).
- Rosten, E., & Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In *Computer Vision – ECCV 2006* (pp. 430-443). Austria.
- Rosten, E., Porter, R., & Drummond, T. (2010). Faster and better: a machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, (pp. 105-119).

8 Appendix:

8.1 Appendix A: Alternative Approaches

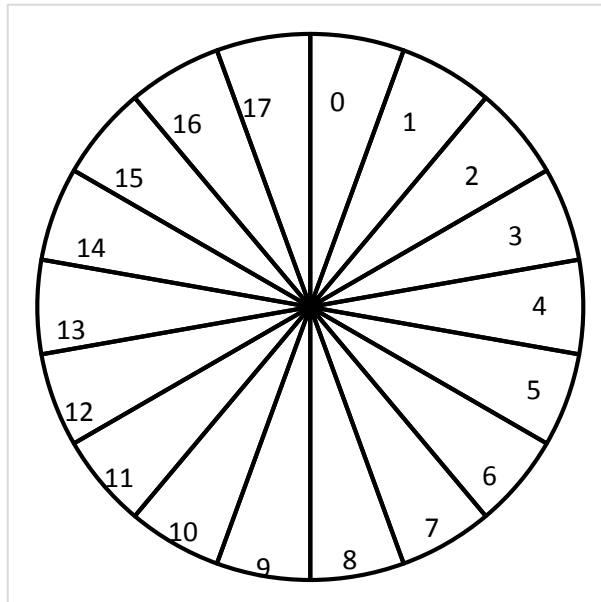
8.1.1 Vanishing Points:

As the majority of the research presented relies on vanishing point detection as a starting point, this approach was the first explored. The advantage of this approach is the plane of the façade is identified and the façade outline can be extracted. To successfully extract the vanishing points associated with the façade planes requires accurate line detection and classification. The lines must be based on the regular linear features on building façade, such as roof line, window edges and door frames.

8.1.2 Line extraction:

To extract the linear features within the image, as line detector is applied to the edge image. Two line detectors were trialled, the Hough Line detector (United States of America Patent No. US3069654 A, 1962) and the probabilistic Hough Line detector (Matas, Galambos, & Kittler, 2000). Both of these line detectors are implemented in the OpenCV library.

These lines are then grouped based on their orientation. Divide the range $0 \rightarrow 2\pi$ ($0 \rightarrow 360$ degrees) into N bins, $N = 18$ for this experiment. This means each bin covers a range of 20 degrees.



Extract the vertical lines as lines with orientations in bins 0, 8, 9 and 17. Extract the horizontal lines as points in bins either 4 and 13 (strict policy) or 3, 4, 5, 12, 13 and 14 (relaxed policy), experiment to see which is better

8.1.2.1 Vanishing Point Estimation:

To find a reasonable approximation to the most prominent vanishing point (VP) in an image the following method was trialled. This estimation was run on the horizontal and vertical line separately to extract two vanishing points, one for each group. The method takes the list of lines in either the horizontal or vertical direction and is described by the following steps;

1. Pick two lines at random and calculate their point of intersection. This point of intersection is chosen as a candidate vanishing point.
2. Calculate the score of this vanishing point with respect to all other lines.
 - a. Firstly a score for each line (excluding the two chosen in 1, above) is calculated as:

$$Line_Score = (Length\ of\ line) * \frac{1}{(perpendicular\ distance\ from\ line\ to\ VP)}$$

- b. This score is subject to a threshold to determine if the line is an inlier or outlier for this vanishing point.
 - c. All scores of lines classified as inliers are summed.
3. The estimated vanishing point is then given a score as;

$$VP\ Score = \left(\frac{No.\ Inliers}{Total\ No.\ Lines} \right) * \sum Inlier\ Line_Scores$$

4. Steps 1 to 3 are repeated until the VP Score goes above a threshold, or the maximum number of iterations have occurred, at which point the VP with the best score is returned along with the set of lines classified as inliers.

8.1.2.2 Vertical Vanishing Point Extraction:

Using the vertical lines:

- Threshold lines by length, disregard very short lines as noise.
- Divide image into 13 horizontal segments
- For each segment determine which line are in the segment (count lines that are both partially and fully within the segment)
- For each segment find the most prominent vanishing point.
- Combine these 13 vanishing points to extract the most prominent overall vanishing point.

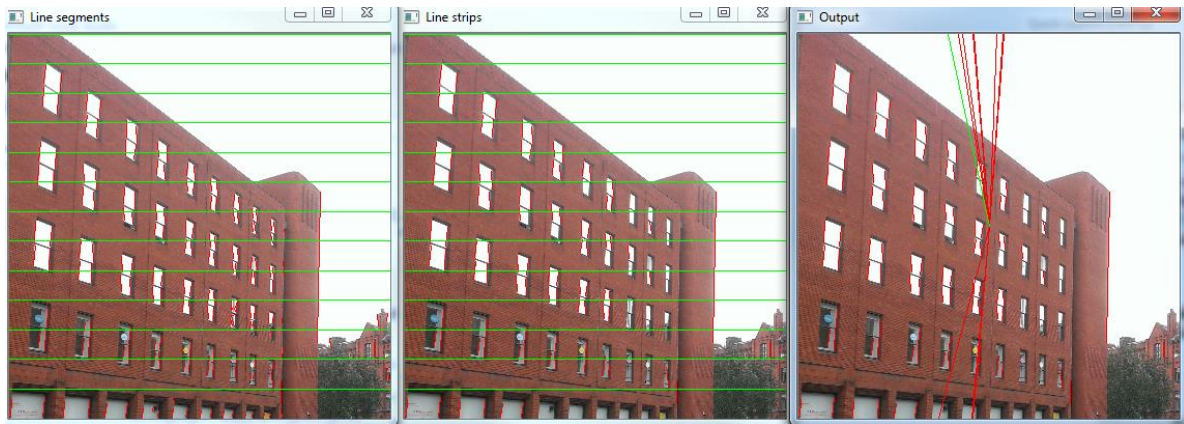
8.1.2.3 Horizontal Vanishing Point Extraction:

Using the Horizontal lines:

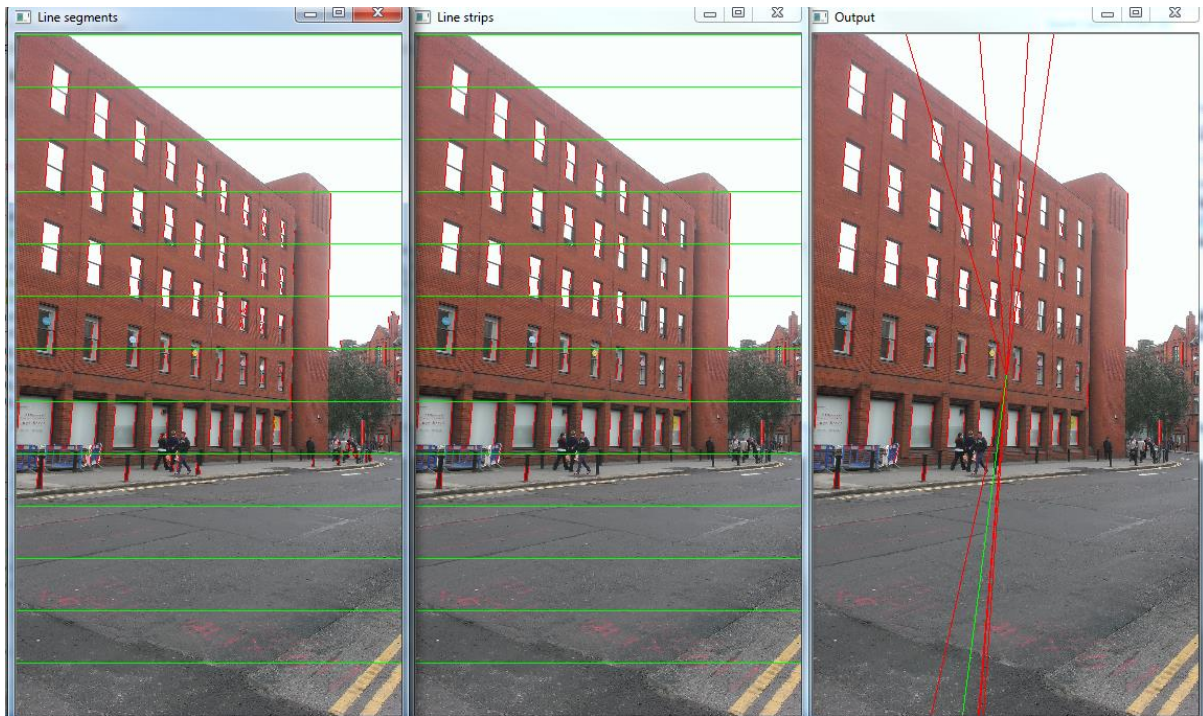
- Threshold lines by length, disregard very short lines as noise.
- Divide image into 13 vertical segments
- For each segment determine which line are in the segment (count lines that are both partially and fully within the segment)
- For each segment find the most prominent vanishing point.
- Combine these 13 vanishing points to extract the most prominent overall vanishing point.

8.1.2.4 Results:

Below two sets of results are presented. In both cases the segmentation of the image into thirteen equal strips is shown. The output shows the vanishing points via a line from the centre of the image to the extracted vanishing point. The coordinates of all vanishing points and the overall vanishing point are shown in the console. The numbers printed at the top are the segment number, followed by the number of lines in that segment, if a segment has no lines, then no vanishing point can be extracted for that segment.



```
C:\Windows\system32\cmd.exe
0 0
1 6
2 10
3 11
4 18
5 22
6 16
7 23
8 24
9 18
10 20
11 11
12 13
Vanishing point for segment 1, found at: (-306, 6153)
Vanishing point for segment 2, found at: (113, -336)
Vanishing point for segment 3, found at: (88, -419)
Vanishing point for segment 4, found at: (-221, 4848)
Vanishing point for segment 5, found at: (-228, 5228)
Vanishing point for segment 6, found at: (493, -9183)
Vanishing point for segment 7, found at: (-148, -3681)
Vanishing point for segment 8, found at: (-259, -5314)
Vanishing point for segment 9, found at: (346, -2056)
Vanishing point for segment 10, found at: (352, -2291)
Vanishing point for segment 11, found at: (212, -244)
Vanishing point for segment 12, found at: (-38, 1037)
Overall vanishing point (averaging) at: (33, -521)
```



```

C:\Windows\system32\cmd.exe
0 3
1 16
2 32
3 31
4 37
5 29
6 21
7 39
8 9
9 0
10 0
11 0
12 0
Vanishing point for segment 1, found at: <-403, 7709>
Vanishing point for segment 2, found at: <57, -100>
Vanishing point for segment 3, found at: <317, -647>
Vanishing point for segment 4, found at: <-270, -5417>
Vanishing point for segment 5, found at: <350, -2277>
Vanishing point for segment 6, found at: <-9, 1207>
Vanishing point for segment 7, found at: <-370, 8844>
Vanishing point for segment 8, found at: <-1066, 15623>
Overall vanishing point (averaging) at: <-174, 3116>

```

8.1.2.5 Conclusions:

The vanishing points for the different segments are very unreliable, there is no consistency between the directions shown. Some segments find the vanishing point above the image and some below. The results shown above are simply not consistent or reliable enough to proceed with this approach.

8.1.3 Histogram of oriented gradients:

The histogram of oriented gradients (Dalal & Triggs, 2005) approach uses the orientation of the pixels around a pixel to calculate a histogram for the current pixel. In this case a circular region around the current pixel is examined and the orientation of each pixel in the region is classified into one of eighteen bins. This is similar to the approach described above of classify lines into bins based on orientation. Once these histograms are calculated the total number of pixels in bins 0, 3, 4, 5, 8, 9, 12, 13, 14 and 17 to summed the number of pixels in the vertical and horizontal directions. The idea was this would highlight region corresponding to the corners of the windows. The results are shown below, with the orientation, gradient, raw data and non-maxima suppressed images shown;

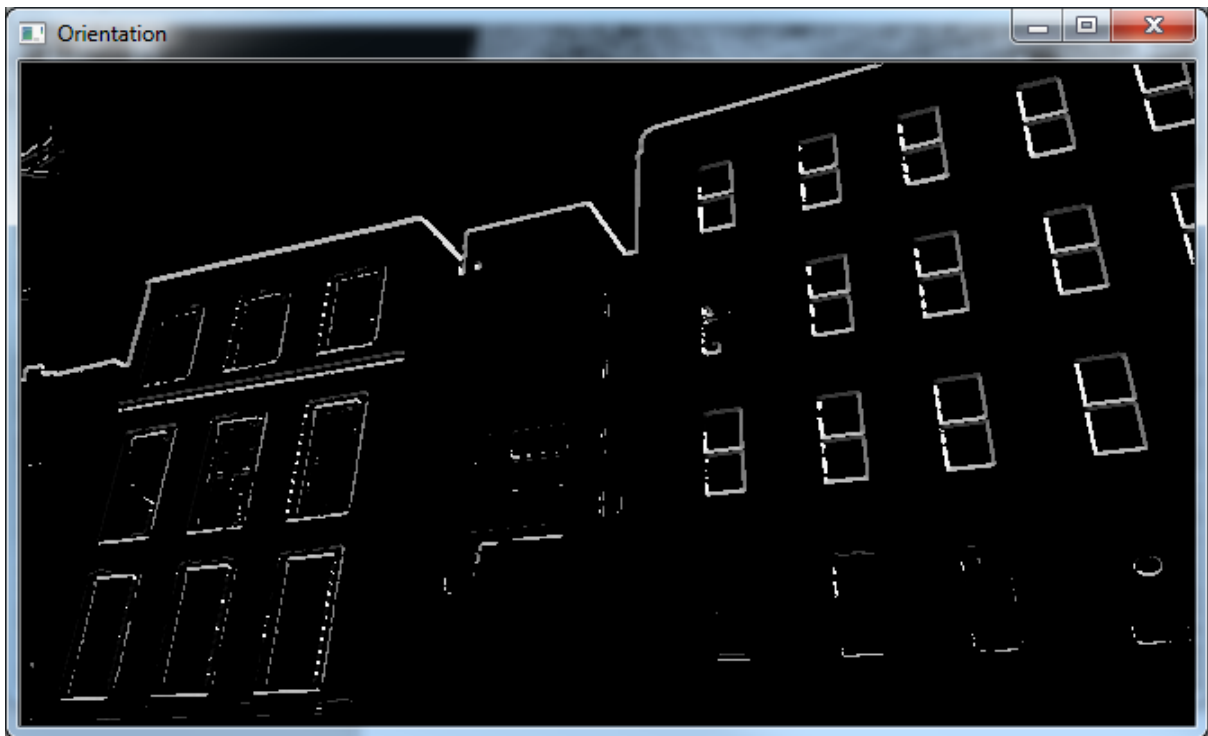


Figure 51 Orientation Image

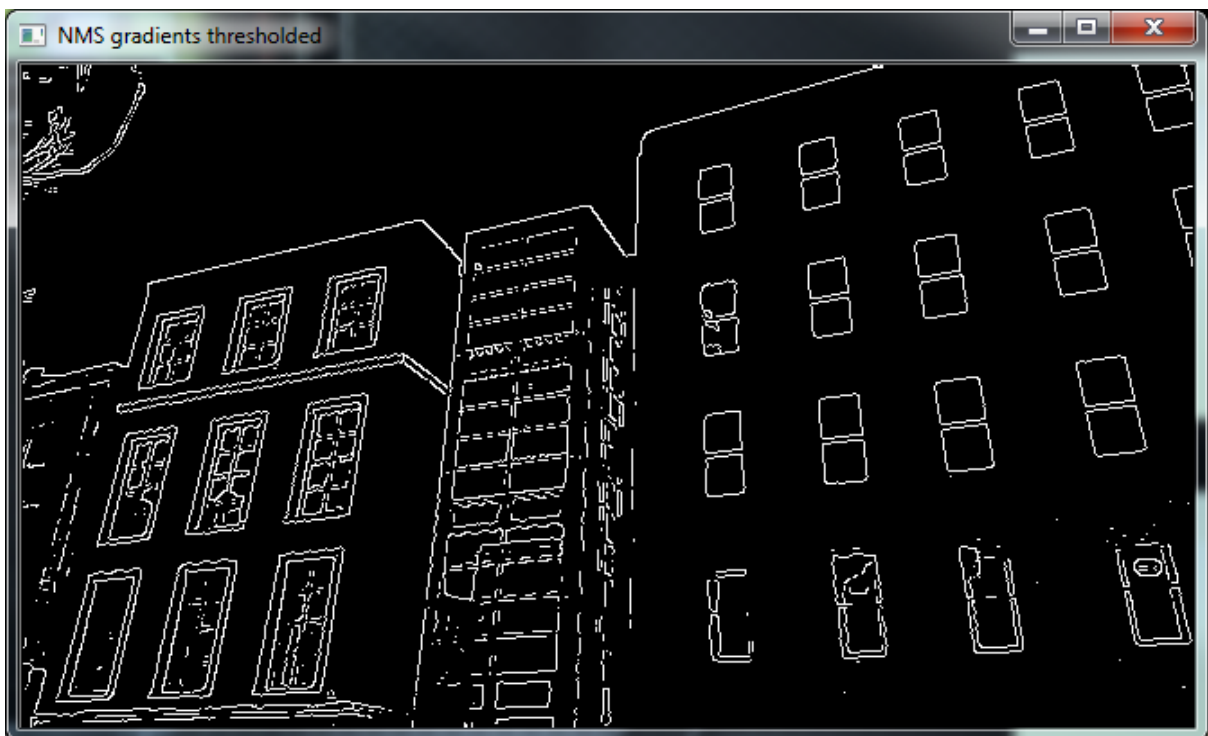


Figure 52 Non maxima suppressed edge pixels

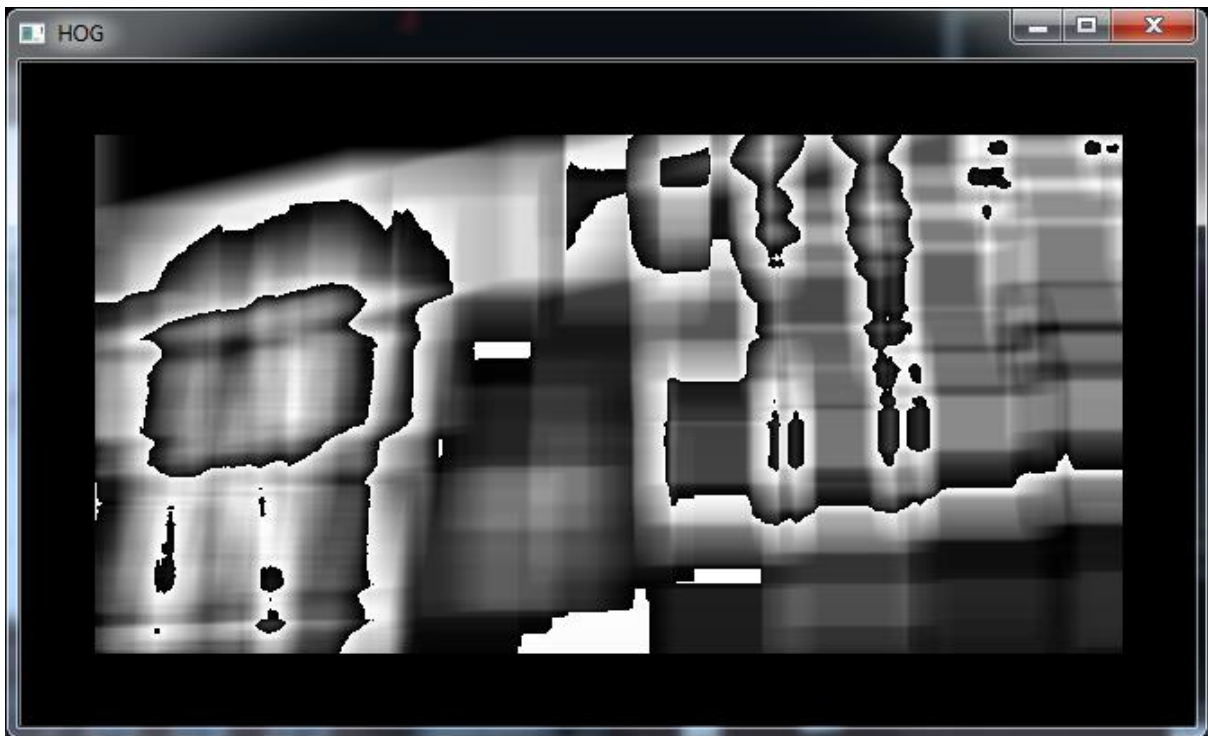


Figure 53 HOG image, simply the number of pixels that are vertical or horizontal within a 80x80 pixel area around the pixel in question

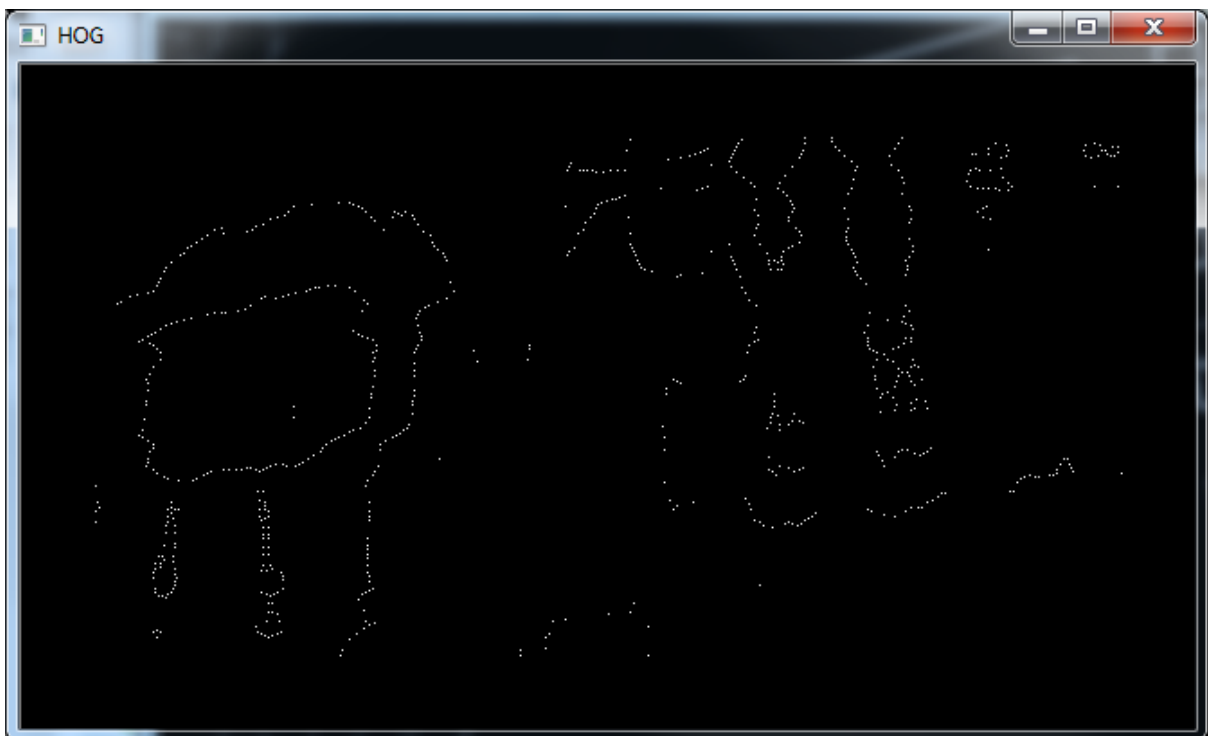


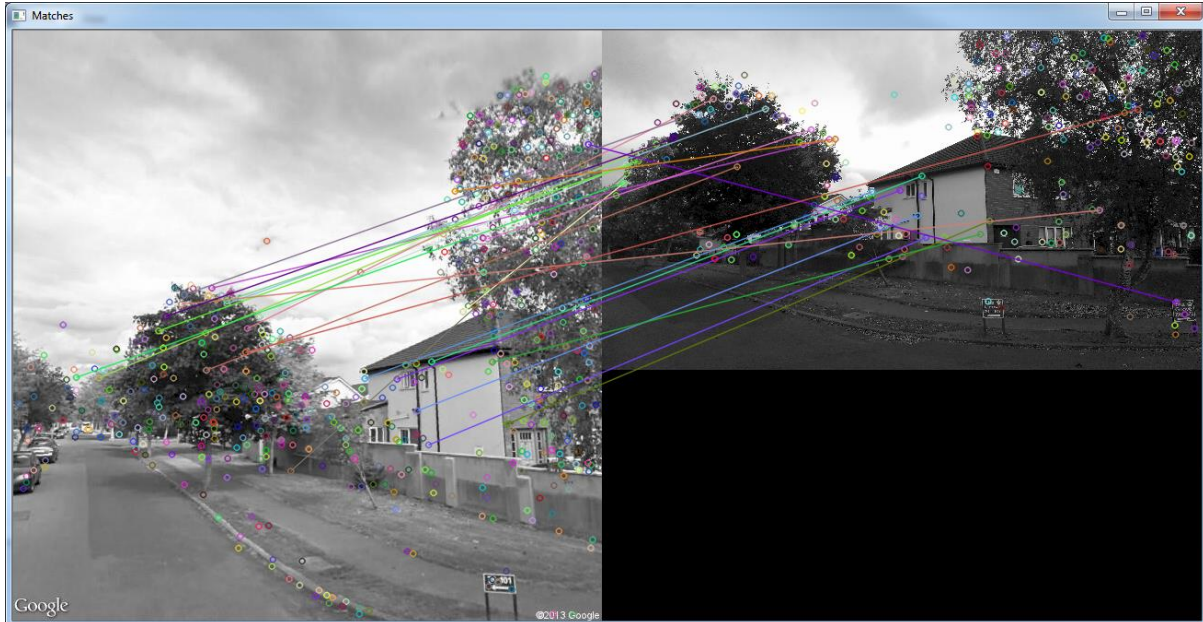
Figure 54 Non maxima suppressed version

8.1.3.1 Conclusions:

The output data is not representative of the window layout. Ideally the non-maxima suppressed response would occur at corner locations, but this is not the case. Thus this approach was abandoned.

8.1.4 SIFT:

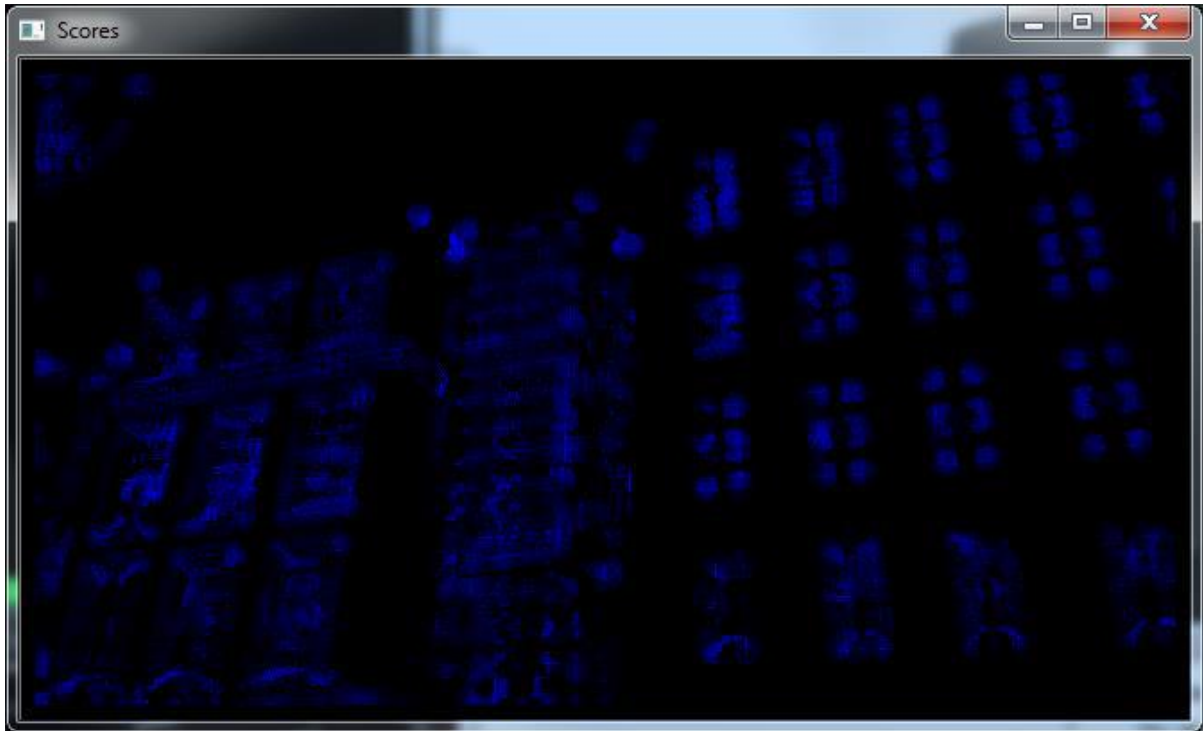
The SIFT technique was presented in 2004 by (Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004). This approach is the standard feature detector and was one of the first reliable detectors introduced. A number of improvements have been made over the years since it was first presented.



The results are presented above. There are a lot of points, circles, discovered. The matched points are generally quite good, but the issue is most are being generated in the scenery, trees, and not on the building. Thus this would require extra work to extract the building façade. Thus this approach may have been very successful if the vanishing point approach had proved successful in extracting the building facades from the larger image.

8.1.5 FAST Features:

The FAST corner detector (Rosten & Drummond, Machine Learning for High-Speed Corner Detection, 2006) (Rosten, Porter, & Drummond, Faster and better: a machine learning approach to corner detection, 2010) considers a circle of sixteen pixels around the current pixel. If twelve contiguous pixels are above or below a threshold it is considered a corner. An alternative and faster approach is to examine four of these pixels, the 12, 3, 6 and 9 o'clock pixels. If three of these are less than or above the threshold, then a corner exists.



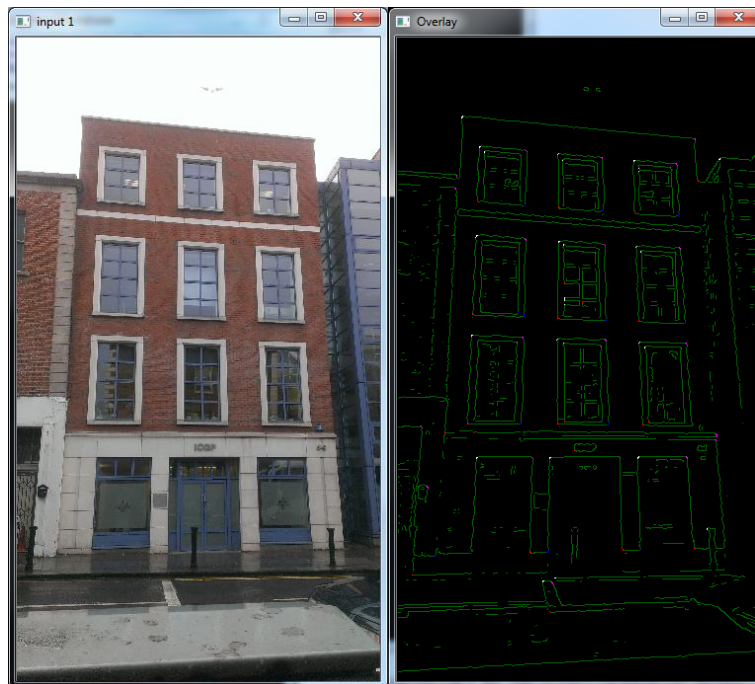
The results of this testing is shown above and the results are quite promising. The only issue being the high corner response in the middle of the window where the pane is split in two. Ideally this would be removed. It was the result of this test which led to the approach presented in the main body of the report being pursued. The application presented in the main report examines the four directions, as FAST does, but tried to exclude the central corners detected.

8.2 Appendix B: Supplementary Results data

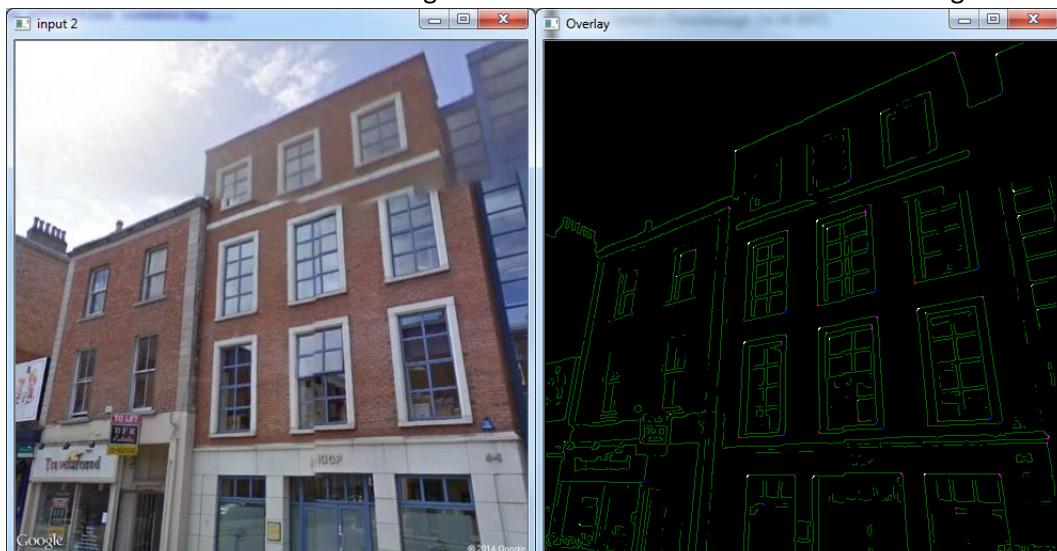
8.2.1 Corner Detector and Window Extractor:

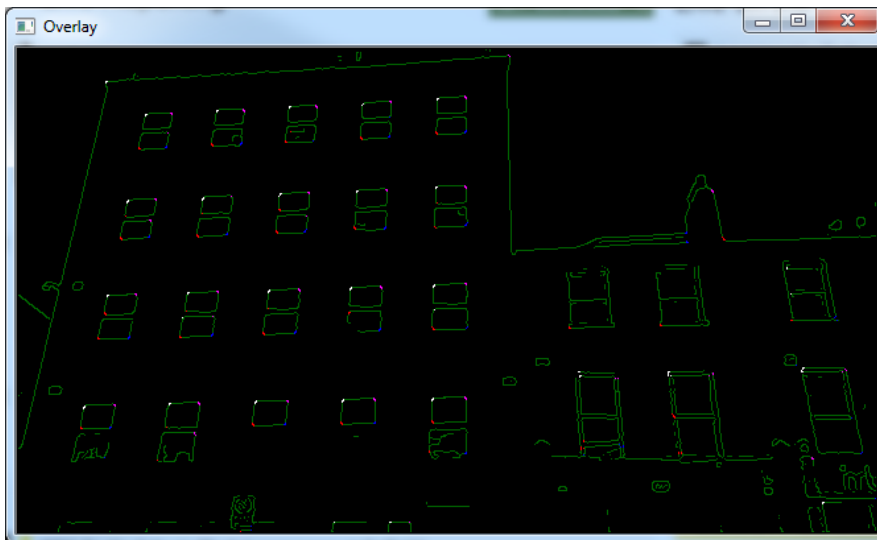
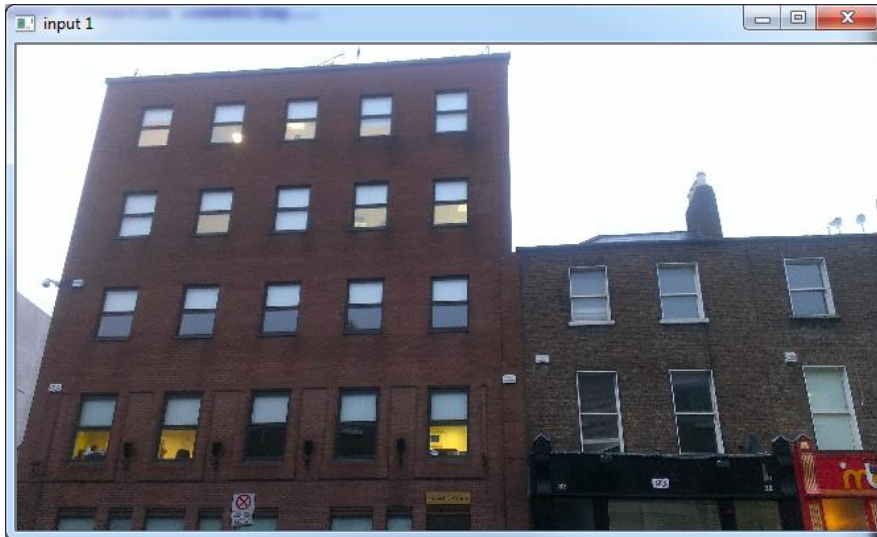
Below follows a number of other examples of input and overlay image pairs. The corners extracted are shown and are colour coded;

- Red: bottom left corner
- White: top left corner
- Purple: top right corner
- Blue: bottom right corner

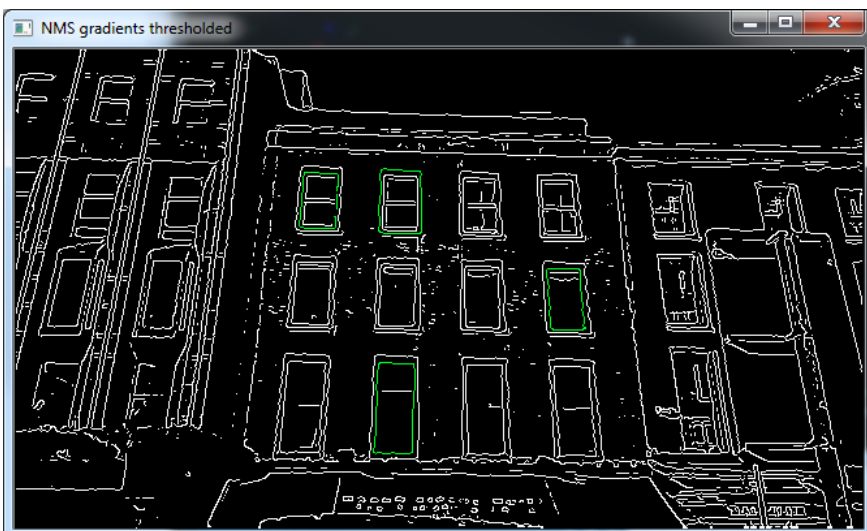
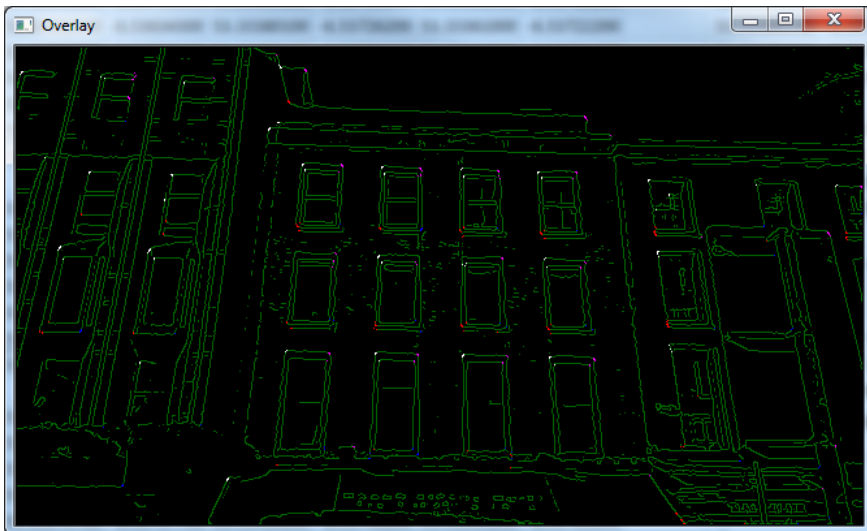


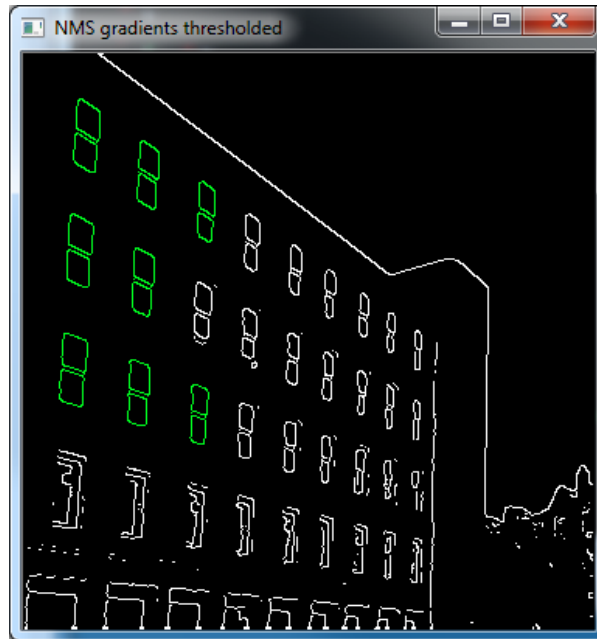
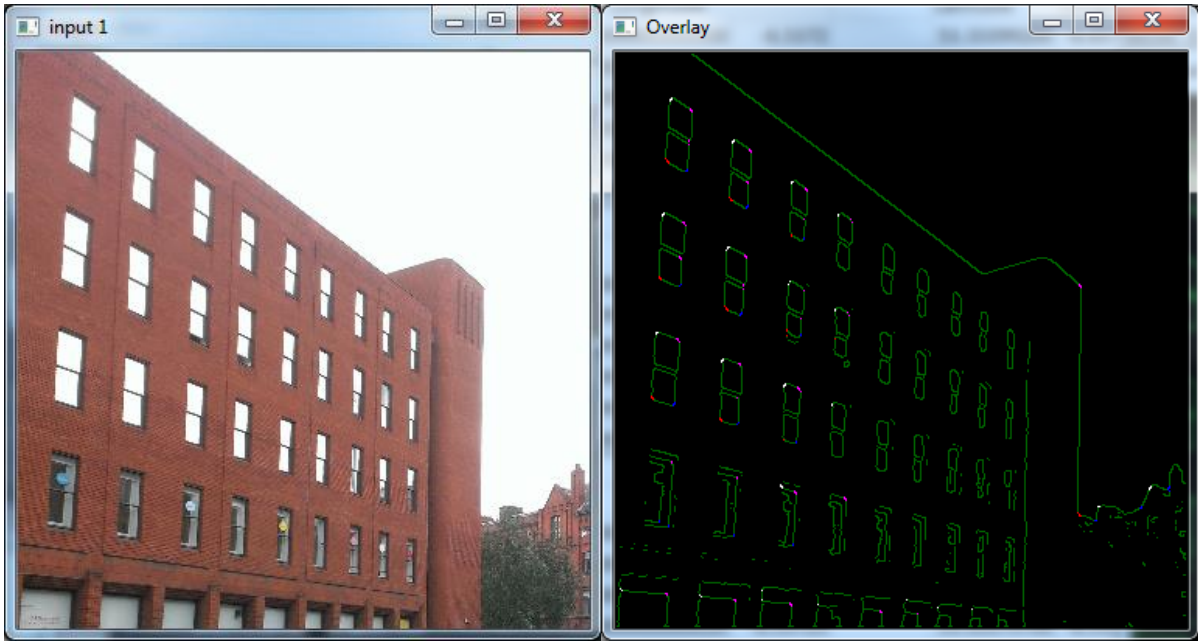
The extracted windows for the image sets above and below are shown in testing section.

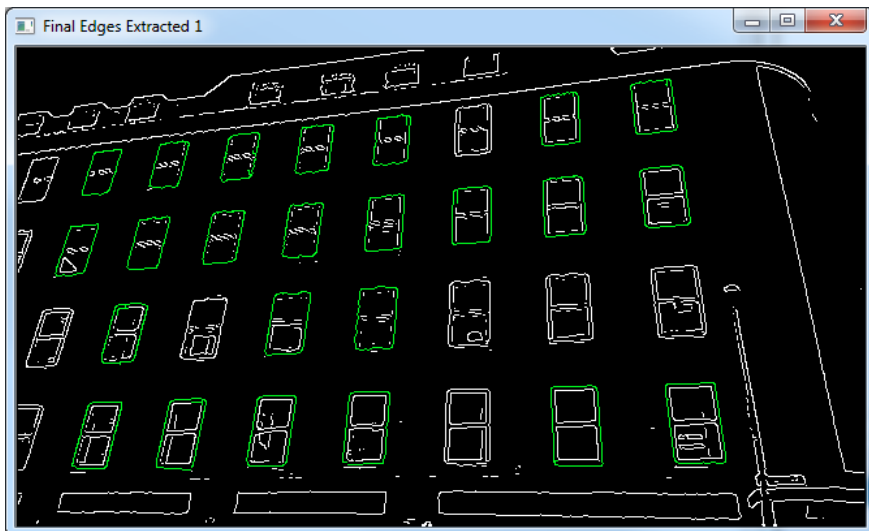
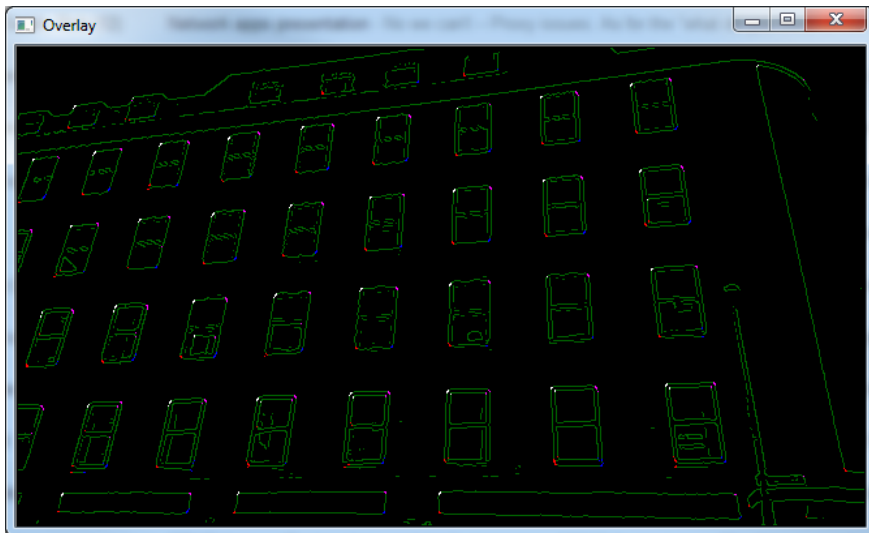












8.2.2 GPS Data

The data below is related to test case two in the GPS testing section above.

Sony Xperia Tipo				
Time step(sec)	Location 1		Location 2	
	Lat	Long	Lat	Long
0	53.333823	-6.537433	53.33403611	-6.537473611
10	53.33381944	-6.537424167	53.33403611	-6.537473611
20	53.33381944	-6.537424167	53.33403611	-6.537473611
30	53.33381944	-6.537424167	53.33403611	-6.537473611
40	53.33381944	-6.537424167	53.33403611	-6.537473611
50	53.33381944	-6.537424167	53.33403611	-6.537473611
60	53.33381944	-6.537424167	53.33403611	-6.537473611
70	53.33381944	-6.537424167	53.33403611	-6.537473611
80	53.33397	-6.536223333	53.33403611	-6.537473611
90	53.33397	-6.536223333	53.33403611	-6.537473611
100	53.33397	-6.536223333	53.33403611	-6.537473611
110	53.33397	-6.536223333	53.33403611	-6.537473611
120	53.33397	-6.536223333	53.33403611	-6.537473611
130	53.33375583	-6.536681944	53.33403611	-6.537473611
140	53.33375583	-6.536681944	53.33403611	-6.537473611
150	53.33375583	-6.536681944	53.33403611	-6.537473611
160	53.33375583	-6.536681944	53.33403611	-6.537473611
170	53.33375583	-6.536681944	53.33403611	-6.537473611
180	53.33375583	-6.536681944	53.33403611	-6.537473611
190	53.3337475	-6.537285556	53.33403611	-6.537473611
200	53.3337475	-6.537285556	53.33403611	-6.537473611
210	53.3337475	-6.537285556	53.33403611	-6.537473611
220	53.3337475	-6.537285556	53.33403611	-6.537473611
230	53.3337475	-6.537285556	53.33403611	-6.537473611
240	53.3337475	-6.537285556	53.33403611	-6.537473611
250	53.3337475	-6.537285556	53.33403611	-6.537473611
260	53.33383833	-6.537470278	53.33403611	-6.537473611
270	53.33383833	-6.537470278	53.33403611	-6.537473611
280	53.33383833	-6.537470278	53.33403611	-6.537473611
290	53.33383833	-6.537470278	53.33403611	-6.537473611
300	53.33383833	-6.537470278	53.33403611	-6.537473611

Average:	53.33381833	-6.537063251	53.33403611	-6.537473611
Median:	53.33381944	-6.537285556	53.33403611	-6.537473611
Std Dev:	7.52637E-05	0.0004622	1.42109E-14	5.32907E-15
Min	53.3337475	-6.537470278	53.33403611	-6.537473611
Max	53.33397	-6.536223333	53.33403611	-6.537473611

	HTC One S			
Time step(sec)	Location 1		Location 2	
	Lat	Long	Lat	Long
0	53.33383558	-6.53730678	53.33383558	-6.53732775
10	53.33383178	-6.53730822	53.33384322	-6.537331111
20	53.33383178	-6.53730772	53.33384703	-6.537328194
30	53.33383558	-6.53730722	53.33404922	-6.537517528
40	53.33383558	-6.53730722	53.33412933	-6.537613861
50	53.33383558	-6.53730722	53.33409881	-6.537476972
60	53.33383558	-6.53730678	53.33402633	-6.537367806
70	53.33383558	-6.53730633	53.33396528	-6.537472278
80	53.33383558	-6.53730633	53.33408356	-6.537484611
90	53.33383558	-6.53730633	53.33407972	-6.537514667
100	53.33383558	-6.53730633	53.33406828	-6.537572333
110	53.33383558	-6.53730633	53.33405303	-6.537483722
120	53.33383558	-6.53730633	53.33407211	-6.537617222
130	53.33383558	-6.53730722	53.33410261	-6.537599028
140	53.33383558	-6.53730772	53.334095	-6.537579972
150	53.33383558	-6.53730867	53.33408736	-6.537570944
160	53.33383558	-6.53730867	53.33408736	-6.537571444
170	53.33383558	-6.53730911	53.33408736	-6.537573806
180	53.33383558	-6.53730911	53.33408736	-6.537573806
190	53.33383558	-6.53730911	53.33409117	-6.537581889
200	53.33383558	-6.53730867	53.33408736	-6.537580972
210	53.33383558	-6.53730772	53.33409117	-6.537579528

220	53.33383558	-6.53730722	53.33408356	-6.537532806
230	53.33383558	-6.53730722	53.33407972	-6.537509917
240	53.33383178	-6.53730822	53.33407972	-6.537517028
250	53.33383178	-6.53730867	53.33407972	-6.537517528
260	53.33383178	-6.53730964	53.33407592	-6.5375085
270	53.33383178	-6.53731014	53.33407592	-6.537509389
280	53.33383178	-6.53731058	53.33407592	-6.537509917
290	53.33383178	-6.53731153	53.33407592	-6.537511806
300	53.33383178	-6.53731206	53.33407592	-6.537511806

Average:	53.33383448	-6.53730808	53.33405389	-6.537514456
Median:	53.33383558	-6.53730772	53.33407972	-6.537517028
Std Dev:	1.72738E-06	1.52114E-06	7.42734E-05	7.87192E-05
Min	53.33383178	-6.53731206	53.33383558	-6.537617222
Max	53.33383558	-6.53730633	53.33412933	-6.53732775