



DUBLIN INSTITUTE
of TECHNOLOGY

Institiúid Teicneolaíochta Bhaile Átha Cliath

***FindMyImage* –Estimating the location of an image.**

Ciaran Corson

C10317489

Dissertation submitted in fulfilment of examination requirements for:

Degree in Computer Science (B.Sc.)
Dublin Institute of Technology

Supervisor: Mark Foley
Second Reader: Luca Longo



Abstract

This project investigates the process involved in estimating the location of a photo, implemented with the help of a few APIs (Flickr, OpenCV, ScyPy and Numpy), through the use of image searching, feature detection and image comparison. The main focus of this project is to estimate the location of a photo where geographical data for the photo is not present. The focus is on the possibilities of this project, as not all photos have geographical data with them, while others do. To be able to leverage the geographical data from a similar photo to estimate the location is an exciting process.

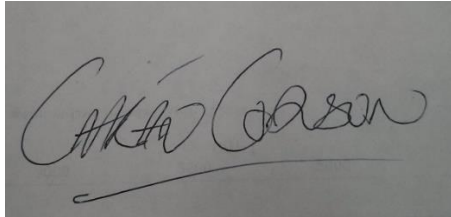
Images in today world are hugely important. The information that can be gained from them is astounding. Being able to detect what is in a picture is in high demand, and as a result, there are a number of viable ways this particular project could be used. The aim of this project is to simply create an application that estimates the location of the given photo, by comparing it against similar photos that have geographical data associated with them. Users of this application will be able to seek out locations and new information from photos the use in the application. It is hoped that the application enables users to come to new conclusions about their photo, was also providing an entertainment or educational value to it.

For instance, where was a photo taken of some church in Dublin? What church is it? A tourist could take a photo of it, and may not remember what the building was.

Declaration

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

Signed:

A handwritten signature in black ink, appearing to read 'Ciaran Corson', is written on a light-colored surface. The signature is fluid and cursive, with a long horizontal stroke extending from the bottom of the name.

Ciaran Corson
April 3rd, 2014

Contents

Chapter 1

Ciaran Corson	i
Abstract	ii
Introduction	vii
Background	vii
Objectives	viii
Summary of Chapters	ix
Introduction	ix
Estimating the location	ix
Research	ix
Design Methodology	ix
Development	ix
Testing	ix
Evaluation and Conclusion	ix
Estimating the location of an Image	x
Introduction	x
Process of Estimating a Location	xi
Why do it?	xi
Research	xii
Literature Review	xii
Technologies Researched	xiv
Search APIs	xvi
Computer Vision	xvii
Related Systems and Work	xviii
Design Methodology	xix
Introduction	xix
Choice of Design Methodology Used	xx
Agile Development Model	xx
Project Plan (Software)	xxi
Relevant Prototyping	xxiii
Conclusion	xxviii
Development	xxviii
Introduction	xxviii
Flickr API	xxix

OpenCV	xxx
Matplotlib	xxxi
Setuptools and Docutils	xxxi
Shapely	xxxi
Numpy	xxxii
System Architecture	xxxii
Component Architecture	xxxii
Controller	xxxiii
Client Page and Uploading Image	xxxiv
Crude Search for Suitable Images	xxxvi
Comparison of suitable images	xxxvii
Returning a result	xl
Development and Implementation (of main components)	xlii
Image Query	xlii
Image Comparison – Feature Matching	xliv
Image Comparison – Histogram comparison	xlvi
System Environment and Technologies Used	l
Web Server	l
Operating System and Hardware	li
File Storage	lii
Testing	liii
Introduction	liii
Performance Testing	liii
Usability Testing	lv
Test Cases	lv
Conclusion	lvi
Evaluation of Project	lvii
Introduction	lvii
Challenges Faced	lvii
Changes in Project Plan	lx
SWOT Analysis	lxi
Conclusion	lxiii
Conclusion	lxiii
Introduction	lxiii
Main Points Learned	lxiii
Future Work	lxiv

Final Words.....	lxvii
Appendix A– Results	lxviii
Appendix B- Summary of Files Submitted.....	lxix
References.....	lxx
Figure 1 Advantages	vii
Figure 2 Image Location	x
Figure 3 Agile Development Process	xxi
Figure 4 Gantt Project Plan.....	xxii
Figure 5 Use Case Diagram	xxiii
Figure 6 Prototype Process	xxiv
Figure 7 Client Screen 1	xxv
Figure 8 Prototype Screen 1.....	xxvi
Figure 9 Prototype Screen 2.....	xxvii
Figure 10 Architecture Diagram	xxxiii
Figure 11 Controller Code	xxxiv
Figure 12 Upload File Code.....	xxxv
Figure 13 Upload Image Page code	xxxv
Figure 14 GPO Original.....	xxxvii
Figure 15 GPO Result 1	xxxviii
Figure 16 GPO Result 2.....	xxxviii
Figure 17 GPO Result3.....	xxxix
Figure 18 GPO Result4.....	xxxix
Figure 19 Client Page code	xli
Figure 20 Flickr Example Response	xliii
Figure 21 Flickr walk method.....	xliv
Figure 22 determine season	xliv
Figure 23 Output log.....	xlvi
Figure 24 feature matching part1	xlvi
Figure 25 feature matching part2	xlvi
Figure 26 feature matching part 3	xlvi
Figure 27 Draw Keypoint Matches.....	xlvi
Figure 28 Histogram Part1	xlvi
Figure 29 Histogram results.....	xlvi
Figure 30 Histogram part 2	xlvi
Figure 31 Display method code	l
Figure 32 Xampp Control Panel	li
Figure 33 Compare Folder	lii
Figure 34 Performance Tests Result Screen 1	liv
Figure 35 Test Cases 1	lv
Figure 36 Managing Project Plan Changes.....	lx
Figure 37 Gui Start Screen.....	lxvi
Figure 38 Gui Progress Screen	lxvi
Figure 39 Gui Results Screen	lxvii
Figure 40 Results Map.....	lxviii

Introduction

Background

This project is based on the desire and interest in the geographical data associated with an image. Images are a large part of social media and many use it to document an event or describe something that happened, as it gives a perfect visual illustration. Studies have set out to acquire, maintain and analyse this kind of information for many different purposes (scientific, geographical and academical). *The location of an image is a highly valued asset of data, and there are many reasons why acquiring that information is desirable.* Therefore being able to estimate the location of a photo accurately, where there is no geographical information present (GPS address) is something worth researching.

Many studies that have set out to achieve this goal have been able to use the information available from these photos to their advantage, as illustrated in the table below.

	Description
Tourism	Taking a picture on holidays of a famous building, but forgetting where the photo was taken.
Scientific	Ascertaining what kind of climate or season the image was taken in.
Geographical	Enable studies of related pictures and geographical data through similar images, from the same geographical location.

Figure 1 Advantages

The interest in this endeavour commenced during a research process in the summer. It became apparent that such a system would have huge benefits to society as many are interested in the geographical data of photos. It became clear that a system like this could be a huge help in estimating geographical information for a photo. This is a very large research area in computer science that is still ongoing. It is a relatively new topic which has yet to be completely mastered. There are many approaches to collecting the images needed to start an estimation process, and there are numerous ways to compare the images collected. No current method has proved to be conclusively more advantageous than another, but there have been enough technological advances in the field for it to be possible to create such a project as ambitious as this.

Images without geographical information, namely GPS data is a thoroughly interesting and exciting topic, and there is definitely a need and niche for such a system. Security, geographical analysis, entertainment and so forth are some of the many examples of why there are reasons as to how this project could be considered a huge success. As mentioned above, it is a new topic which has yet to reach its full potential.

Finding a photo without GPS tags is the premise, and the benefits to this project are without a doubt very impressive. People have always been intrigued by a photo's origin, and to be able to estimate, or come to a very close conclusion of where the photo was taken, and even an exact match. The amount of geographically tagged data is extremely high, and yet there are still images that do not contain gps tags. To be able to successfully leverage those geographically photos to estimate the location of a photo without a confirmed location really is an exciting endeavour. [1]

The applications for such a project are numerous. The amount of geo-tagged photos available to the public eye is quite a big figure, and this could be leveraged to find more photos like it. Being able to geographically identify photos has huge benefits. These include being able to find out where your friend took that embarrassing photo of you, aiding natural disaster teams (identifying objects and so forth) and even weather predictions. Seeing a photo in the same place from many different angle on different days can give a deduction of the climate, and help predict the weather. Overall, the use of such an application would be well received. [1]

Objectives

The objective of this project was to research the possible implementation of a solution for the use in estimating the location of an image, without geographical information. This objective meant that the project was considered a success if it returns a reasonably accurate estimated location for the photo, along with similar pictures, under normal conditions. Normal conditions are defined as use of acceptable input from the user for a valid, comprehensive result that in some way meets their expectations.

The objectives can be broken down into:

1. Create a fully functioning crude search query that can return suitable images for comparison. The images returned in the query must have gps data associated with them.
2. Create a set of techniques to successfully analyse each image that is returned from the search query, in a suitable comparison process.
3. Create a set of techniques to compare those images, through methods such as the SIFT algorithm, template matching and colour histograms.
4. To return a valid response to the user, with an estimation result of where the photo was taken, along with similar photos.

The objectives in this project are inter related, if the first objective is not met to a satisfactory level, the next objectives will be limited in their success. The system needs suitable images to compare, and can't return an estimation without a corpus of images to compare from.

When this project was started out, the objectives were to try and make the search query as dynamic as possible, in that an estimated location or even search tags were not a major requirement. This has since changed, as user data on photos, and the overall complexity of image searching mean that it simply is not possible to create a valid search query. The comparison process as a result has a higher chance of succeeding, as the images it is receiving are more valid.

Summary of Chapters

This section provides a short summary of each of the chapters that will follow in this report.

Introduction

The introduction chapter acts as a preview to the report, providing details on what will be discussed in the report. It also provides a high level overview and background of the project.

Estimating the location

This chapter deals with the overall aim of the project. It seeks out to explain the overall desire to find the location of an image. It deals with how it would theoretically be possible, and sets up the scope for how the project should be done.

Research

This chapter covers all the research and information gathered in relation to this project. It covers any research that was part of this project, be it hardware research to host the application, computer vision techniques and suitable APIs.

Design Methodology

This chapter deals with all the design methodologies that were considered for this project along with the chosen design methodology, agile in detail.

Development

This chapter deals with the development and implementation of the system. It covers topics such as computer vision, API programming, file IO and so on. It discusses how the system was designed, developed and implemented.

Testing

The testing chapter covers the different testing methods that were used in this project, including performance testing, usability testing and results testing.

Evaluation and Conclusion

The last two chapters deal with the evaluation of the project and any conclusions made. The evaluation deals with the results returned from the system and how they matched expectations, while the conclusion deals with how the project fared with respect to those results. The conclusion deals with what was learned in the project.

Estimating the location of an Image



Figure 2 Image Location

[2] (Original photo credited, photo changed to include question marks).

Introduction

Estimating the location of an image comes from the desire to know more about an image. Where was the photo taken? When was it taken? In what climate and season was the photo taken. There is a huge movement within computer science to estimate geographical information, be it from a picture or another piece of data.

Remember that photo you took of the lovely building in a town outside Barcelona in Spain? Imagine driving through the town, seeing it and taking a photo on your camera, but have the GPS off. [3]

Recognition systems that recognise the location of a photo are used to map the world's photos, and the use for them is in high demand. Being able to estimate, or even accurately guess where the photo was taken is therefore an extremely useful system. "The emergence of vast amounts of geographically-calibrated image data is a great reason for computer vision to start looking globally – on the scale of the entire planet!" [1]

Process of Estimating a Location

The process of estimating a location of an image varies, as there are many approaches to this, with varying levels of success. The process used in this application is one of several steps, which aims to return an estimated result.

It does this by taking the image, constructing a search query from the image and user input. This includes season, time of day, some search tags, and some sort of estimated location provided by the user (city). Doing this allows for the query to find similar pictures that can be compared in an image comparison process to return a feasible result. Any image found contains a GPS address which allows it to be compared against the original without a gps address. If the images match, the gps address is used to open up a map to the point, where the user can see where they took the photo. The process of estimating an image is to find suitable images, compare them and return a suitable result, and therefore any approach taken must do this effectively. It can be said that the process relies on several factors and processes being accurate, as if they are all accurate, the results are returned are usually more valid.

Other approaches involving downloading a very large corpus of images (20 or even 30 million images), storing then and building a search query locally, which isn't practical in the case of this project. This was the IM2GPS approach, which while being successful came with a large overhead of data processing and storage. These approaches feel it is better to have the images at hand rather than acquire them as needed, which wasn't suited to the goals of a system like the one developed.

Therefore, other approaches were considered unusable. The process is meant to estimate the location and return some viable results which means the process should collect possible matches and compare them quickly. It was desired that the system would work quickly under pressure, only needing the images for that particular request. The emergence of geographically calibrated data also brings about a pique in interest with regards to images. This process therefore has a great advantage in that there are so many geographically calibrated images to work with, that the process potentially becoming easier.

In the words of the main research paper that influenced this project, "The emergence of vast amounts of geographically-calibrated image data is a great reason for computer vision to start looking globally." [1] IM2GPS has a very good google tech talk video which is worth a watch in relation to this project. It can be found at this address. [Tech Talk Link](#)

Why do it?

While many photos available on the internet do contain a valid address, there are many that don't. Being able to complete such a feat really is an exciting prospect. It all boils down to the curiosity of human nature. Desiring to know more about a topic is something that drives innovation and scientific discoveries by the human race. This project is no different. Being able to locate a photo's location is all about deducing where the photo was taken, and this is enough to entice the average user of such an application.

There not only is a desire present, but also the ability. Technology needed for such an aspiring project is at a sufficient level where this project could be carried out successfully. API's like Flickr have reached the point where constructing specific queries for Geotagged images is very doable. Computer Vision libraries like OpenCV, SimpleCV (lighter implementation of OpenCV for Python) and Mahotas (computer

vision library for Python) all have very powerful functionality that works well in a large number of situations. To correctly ascertain the location from an ambiguous photo is the biggest challenge to a project like this.

Being able to accurately pin point the location of a photo is without a doubt a very challenging thing to do. This is more than enough motivation to attempt a project as ambitious as this. To construct a project that could feasibly be used to ascertain the location of a photo is an exciting prospect which was undoubtedly the main source of motivation during the course of this project. Having a system with the functionality needed was the aim, and if the functionality was successful in achieving the goals, then the potential for this project could be very high. In summary, the potential is there to make an application, and the functionality was there to make it work well.

Research

Literature Review

For this literature review, I sought out to research and evaluate potential systems that were similar in structure, design and had a similar end objective to my own. I thoroughly researched many topics, ranging from the google tech talks on image geo-estimation, to Flickr API tutorials and so forth. Considering how ambitious and complex my project is, I realised quickly that I needed to research successful implementations of APIs for Flickr like PhpFlickr [4]. Python Flickr [5] too, as finding suitable images to compare was a priority when I first started. In the end, I chose Sybren A. Stüvel's implementation of Flickr in Python instead [5]. This implementation creates a sufficient wrapper for the API, which implements all the methods successfully, and can be easily managed.

When starting out, researching image feature extraction techniques was another big priority. I used examples such as the canny edge detector example to research how an edge detector might be created. I also looked at the SIFT algorithm library. SIFT stands for Scale Invariant Feature Transform, and is an image algorithm capable of image correction, comparison, feature detection and other valuable image processing techniques. Later in the year, I made a decision to switch the entire functional end of the project to python, realising how efficient and powerful a language it was. Within Python, the research I did was highly conclusive. I was able to use OpenCV, which implements the SIFT algorithm in a class.

I sought to research anything that could benefit my project, and as a result, I have collected a wide range of sources and documents that have helped make this project possible. For instance, there are not many applications that can estimate image location effectively at the moment. It is still quite a new research topic, and the IM2GPS study describes it perfectly. "Estimating geographic information from an image is an excellent, difficult high-level computer vision problem whose time has come. The emergence of vast amounts of geographically-calibrated image data is a great reason for computer vision to start looking globally — on the scale of the entire planet". [1] Researching IM2GPS as a reference has allowed me to grasp the key concepts of this project quickly. It is a valuable resource which will be a great help during this project. IM2GPS seeks to collate results and create a system that can

estimate the location of a photo effectively and with a high accuracy rate. While the IM2GPS system and the proposed system differ in objectives, standards and techniques, they both seek a similar end goal.

The research of this project also led to the belief that a virtual machine may not be practical as first thought, as it became extra works, distracting from the actual design and implementation of the system. As the virtual machine was in a different setup and language than I've usually seen, it was a learning curve which I felt was best to scrap. It will still remain as a backup in the event of hardware failure, but the system was developed within a windows/Eclipse/PyDev [6] environment which made it far easier to simply concentrate on the developing of the application. PyDev is a Python IDE which can be implemented within Eclipse and can be used for Python, Jython and IronPython projects. It allows for standard eclipse features to be used, like: code completion, refactoring, code formatting etc.

While researching for how the system's components would integrate and communicate, I realised that it would be quite simple from that perspective. The server side script to collect images from the user would ideally be written in Python, or PHP. Once the user had uploaded the image, the script would call the query python script, starting the search process. The search process could then call the comparison process. When researching tier systems, it became clear the process could run seamlessly in the background, while the user receives updates.

While researching potential data storage methods, databases and file storage methods were compared. While databases are generally more efficient, powerful and better suited to storing large amounts of data, it became clear that a database would not have been as practical as originally thought. The data could be accessed quickly through a database, but not in a way that suited the system. The system works with one original image, and around 40-60 other images (for comparison), with a maximum of 80 for comparison images. Being able to iterate through each image in a folder makes it far simpler to just put the images in a folder, and get the comparison process to run through each image at a time, and move onto next image in the "queue" when done with an image. This approach I feel is better suited to the project.

The point of any research in a situation like this is to ensure that the project will be a success. The research done for this project was to look for a number of things:

- Firstly, any research done on similar systems was done to get an idea on how to approach this project. The IM2GPS study [1] for example had been a huge success as it followed a set of defined steps that were deduced from good research and a good development and testing strategy. They realised that leveraging a very large corpus of suitable images was the best way to go about starting a comparison process, which proved to be successful. Being able to see why it was a success was very important. Finding out how it was successfully implemented provided a great insight into how this project needed to be approached for it to be successful.

- Secondly, research was done on various tutorial and research documents which gave explanations on how to achieve each task needed to be completed for the project. This involved looking at python tutorials and blogs which explained various terms and approaches to completing certain tasks. The research done was used to my advantage, as it allowed me to gain an understanding and appreciation for each of task very quickly. I was able to develop functionality quickly as a result. [7]

In conclusion all the research done here proved to be a huge success as it helped the project succeed. The project wouldn't have been successful, if it were not for these studies, research papers and tutorials. The purpose of the literature review was to illustrate how the research done would benefit the project, and it is clear that it has achieved that goal.

Technologies Researched

For this project, I researched numerous technologies I felt were valid for each of the different components. The technologies I have researched range from web servers like LAMP, operating system environments like Ubuntu, image comparison libraries in Java and Python, multiple APIs for image searching and so forth. I've looked at many technologies, having researched their benefits, disadvantages and overall simplicity. While researching, I looked for technologies that would interact with other technologies smoothly, could manage the amount of data they were getting effectively, and basically looked for a near perfect fit for each requirement. As a result, the technologies chosen have been implemented successfully, and at first I approach)

Approach 1: Included LAMP (a full server stack with Linux, Apache, PHP and MySQL. It involved using a virtual machine, and accessing the machine using port forwarding. The solution was provided by Linux Turnkey, as a full solution package. There were alternatives to this, like WAMP. WAMP being an AMP stack for Windows, but it wasn't as strong or as efficient as LAMP.

A suitable java environment was also considered to run the comparison process. This would involve having an IDE (ideally Eclipse) to run the comparison processes, like SIFT and so forth. The comparisons performed are very demanding on the underlying hardware, and would be quite complex, so I was had to choose suitable libraries that met these requirements.

OpenCV was researched for this approach, as was SIFT, and the Canny Edge Detector. I had realised that SIFT and OpenCV were the most comprehensive, which comfortably beat all of the remaining competition. While Canny Edge Detector was a good fit, it didn't have nearly as much functionality as OpenCV, or SIFT. (This also meant that I didn't consider it for my second approach using Python, which I just used OpenCV, and SIFT). These libraries should be simple to use and manage, allowing for quick changes and manipulations to be made, as image comparison is a hard process and takes many tries to find the right balance. Therefore, these libraries will be edited quite a lot, and must be easy to change, implement and workable again in a few minutes

Later in the year, just before the start of development, I came to the conclusion that I would use Python instead. As I had a good understanding of the language, I felt that it wasn't a big step to swap, despite the fact that I was very experienced in java. I had also reached the conclusion that I would eliminate the need for LAMP, and even the need for a virtual machine, unless completely necessary. It would become necessary in the event of a hardware failure on my machine.

A virtual machine, accessible by port forwarding wasn't practical when the system needs to return results at an acceptable rate, hence why it was scrapped. Instead, the system operates as a python application with a server linked to it, also created in python. The server is managed as an apache server, and interacts with the python application in the background, manually. There were issues trying to connect the server to python to start the process normally, with it being full of errors and not working correctly, the image is uploaded and then the process is started manually. This will be explained in more detail in the development and implementation chapter. Not only that, but it became apparent very quickly that my other machine at home simply wasn't capable of running the application as the performance demands were too high.

Python was chosen for a very simple reason. It's efficient. I estimated, during the process of developing the crude search component that the equivalent amount of code in Java would have been almost double that in Python. In 140 lines or so, I had the a of the functionality completed, while I estimated I'd need close to 350 lines in Java for that functionality alone, probably spread through multiple classes. Python, like Java allows for numerous imports to be downloaded and implemented into the interpreter, and was therefore at an equal footing with Java. There was no need to worry about missing out on functionality in Java, as Python had access to these libraries and API's too, most likely a lot more for what was needed in the project. [8]

The technologies I am using are vital to the success of the project, and they form the very core of what my project is about, therefore the research into this area was of utmost importance. The process from getting a user's image, searching for suitable images, comparing images and displaying the estimated location of images must run as smoothly as possible, therefore these technologies must be integrated successfully. I set out to refresh my knowledge in Python, while also learning as I went. The functionality available to me in Python allowed me to progress through my goals for the project quickly, as I found it easy to develop the functionality needed. The learning curve for the language is gradual and easy, and it puts an emphasis on readability.

Search APIs

For this project, it was imperative that a corpus of suitable images could be collected for each query. It was a big concern when this project started out whether any API in question had the functionality needed to create the specific searches needed in this application. Research for this area meant acknowledging that what the task of searching for suitable images was a complicated one.

The APIs researched for this included – Flickr API, Instagram API, Twitter API, Facebook and so forth. The APIs were researched with a few things in mind. It was imperative that the API allowed me to find geographical data associated with photos found in a search. As most of these APIs are based on websites that have photos being uploaded on a very regular basis by smartphone, it is easy to assume that the photos uploaded contain a gps co-ordinate associated with the picture. These photos are therefore ideal to use in a comparison process, if they can be downloaded efficiently, while having a similarity to the original image in question.

Facebook and Instagram were put to one side in favour of Flickr at the start (for purely research purposes, in the hope that if testing was successful with Flickr, the other APIs could be implemented into the system at a later stage. As of now, the only API implemented in this project is Flickr (a Python wrapper of it), but it is hoped that the other APIs could still be implemented at a later stage. The search functionality was always to find similar photos, through any sort of similarity. However, the search needed to be efficient and return a result that was a decent match.

Naturally, other search APIs were researched heavily and tested to see how they performed.

- Facebook was too complicated for the functionality I needed.
- Google did not provide all the functionality I desired out of a search (searching for images on google can be complicated and tricky).
- Instagram while having access to millions of geotagged images, it still fails in comparison to Flickr. Flickr simply was superior in every way
- Twitter was considered an option, with the added fact that hashtags could be used to help with search terms. It however, wasn't up to scratch to search for images and in testing, it failed to return an acceptable corpus. It could of course be added to a future release of the software, where more time could be spent researching the API.

These were all the APIs I considered, but without a doubt there is room for improvement. Much more research could be done to find other image providers.

The purpose of an API for a website like Flickr or Twitter is that it allows the user to access and search for content on the website, in a very specific way. Flickr for example has a multitude of search methods that take a variety of parameters to make the search more effective. As a result, using an API is far more effective to find images than manually searching a website. Having a more specific search makes the process of finding new images quicker, and more accurate, which was the aim. The functionality researched was of a main concern, as finding suitable images to compare was always going to be the trickiest part of the process. Having a bunch of similar bunch images to compare is easier, as computer vision (explained below) is at a high

enough standard where feature extraction and keypoint matching works well. Finding those images is the tricky part because finding similar images is not as simple as searching google. The search needed to be inclusive of time of day, some way to describe the image (search tags, an estimated location, and so forth).

As described above, the purpose of the search query is to return suitable images that can be compared against the original image uploaded by the user. Therefore, the search query was designed to search for images by likeness. This is a complicated process that this application has only half mastered. If this project is to be evaluated correctly, it needs to be assumed that a definite improvement needs to be made to the query. If an improvement was made, the application would have even high success and satisfaction ratios.

Computer Vision

Computer vision was one of the main areas I researched for this project and comprised of research on anything related to image processing. Computer vision is the field of acquiring, analysing, processing and comprehending images. It is used to produce numerical or statistical information from a picture (i.e. make a decision if two photos are alike). Computer vision encapsulates things like: feature detection, homography, histogram calculation, image processing, video processing, camera calibration and picture reconstruction. [9]

For this project, computer vision was researched with the aim of finding a suitable match that could provide the in-depth functionality needed for such a project. Whatever was used needed to be highly powerful and fit the needs of the project. Once such approach was to research OpenCV, a popular open source library which specialises in computer vision. Researching this approach involved implementing OpenCV with a corpus of sample images of the front arch of Trinity. It became obvious that OpenCV was a perfect fit for the requirements. With OpenCV (and SIFT implemented within it), I had a perfect set of tools at my disposal. The image comparison process usually involves 40 or 50 images, and it is reassuring to know that OpenCV can handle these demands well.

Sift was also researched, as it is a way of finding the key points (and descriptors) within an image. As mentioned in the literature review, it is an algorithm for finding keypoints, invented by David Lowe. The algorithm is patented in the USA by the University of British Columbia. SIFT involves constructing a scale space (internal representation of the space on image), a LOG approximation, finding key points on the image, removing the bad points, assigning an orientation to the key point, and then generate the SIFT features. Overall, this allows the SIFT algorithm to identify the key features of the image. [9]Sift is clear example of how powerful a computer vision

algorithm or library can be, and as a result it was realised very clear early in the research process that it would be used.

Not many other alternatives were researched as the evidence provided by OpenCV (and SIFT) made it clear that they were undoubtedly the best options to use. For instance there were other keypoint matching algorithms like SURF (Speeded up Robust Features), but it isn't as nearly as efficient as SIFT. Sift works in a lot more situations that surf does and is more efficient. SURF is inspired by SIFT and it is a performant scale and rotation, and while being faster than SIFT, it isn't as accurate in tests I've performed. [10]

Computer Vision is important to this project as being able to detect similarities in images found is a huge part of this project, and the research done reflects the huge amount of need for it.

Related Systems and Work

When researching for a topic like this, it was important to research similar systems and ideas. Having other projects and systems to refer to is a huge benefit, especially with a topic like this. The area of interest for this project revolves around estimating a photo's location, and naturally this topic has been researched before. The systems that have been looked at include the IM2GPS study which is a study on estimating geolocation from a photo, several libraries of image feature extraction, Flickr search applications and so forth

The IM2GPS project is not a complete fit for the requirements identified, but it allowed me to get a good grasp of how to construct the image crude search. Researching this related system has been hugely beneficial, and has definitely been a major inspiration for how I created this project. What this system does is quite incredible in terms of accomplishments, but uses permanently stored corpus of images, which I feel isn't practical. While they use a corpus of over 6 million "geo-tagged" images, I plan to download images on the fly, for each search query. I plan to use around 30 images to compare against, all from Flickr. This is quite similar to the IM2GPS approach, but I plan to make my query a little more specific, as it will take the image data, and search against much more tags. In short, I'm taking some key aspects and points of this system, applying my own standards and logic, and creating a search system that works in a different way, while still relying on the theory of getting comparable images with geographical data to use in a comparison.

Another system that I looked at was the implementation of the SIFT algorithm. SIFT stands for Scale Invariant Feature Transform, and is the leading algorithm in image feature detection and extraction. It was created by David Lowe, in 1999 and since then it has grown into a serious piece of work. It can be implemented in multiple languages, and its capabilities are numerous. For instance, it has been use in image stitching, image recognition, video tracking and 3D modelling. It can therefore be can applied to image comparison for the purpose for comparing photos and estimating a location in the process. I have implement elements of the SIFT algorithm, using a jar library implementation [9] in Java and an OpenCV implementation of SIFT, within

Python. I had decided to use Python for a number of reasons, and this being one of them, as when SIFT is implemented in Python/OpenCV it is more effective compared to an implementation in java.

While the systems I've looked at are very impressive, they lacked the completeness and usability I needed to implement in this project. They are good learning references and contain many points of interest, and enabled me to create a concise plan to create the system. However, they weren't complete enough when put against the list of requirements I had created. In short, they acted as a great learning reference and helped me create each component effectively as inspiration, but never were going to be 100% similar to the requirements I was after.

Design Methodology

Introduction

As seen in previous chapters, the technologies researched will only perform at their best if the software development methodology used in or the project is successful. In previous chapters, the need for a fully operational efficient system is highlighted. Estimating an image is a complicated process and to do this effectively, the software design methodology must be able to meet these requirements. A successful design methodology will help ensure the success of this.

Any development of software requires the use of a design methodology, if it is going to be a success. As seen in every major commercial project, bugs appear when code is created. The aim of a methodology is make sure the software created meets the requirements created at the start of the process, and testing is done to ensure this, which helps to remove bugs. Before deciding any methodology, it was clear that the design requirements needed to be established, and how any methodology used would meet these requirements. It was clear, from previous experience, that software engineering projects are full of risk, and any methodology used must stand to limit or mitigate these risks.

In the next chapter, the choice of methodology is shown, whereby it is illustrated why it was chosen in respect to other alternatives.

Choice of Design Methodology Used

For this project a few approaches were considered, ranging from the agile methodology, waterfall model, incremental development (which is quite similar to agile) and rapid application development.

Right from the start, it was obvious that agile was the most likely option for a development of such a system. Agile follows a set of steps: design, develop, build and test. The steps are repeated recursively which allows for high quality code and quick implementation of functionality. This approach would obviously suit the needs of the project. The waterfall model is a development style in which development flows downwards (like a waterfall) through the phases of gathering requirements, designing the system, implementation and developing (coding), testing (validation), integration, and maintenance of the system. Incremental development is a combination of agile and waterfall, where the system is released and developed incrementally which allows for new functionality to be made at each “waterfall”.

Rapid application development is a style of iterative development, focusing on creating prototypes which allows for the development of a prototypal system which can be implemented easily. [11]

While all these options were good choices, agile was clearly the best fit as it allows for code to be developed quickly and to a high standard, increasing the functionality on each turn. Therefore it wasn't hard to choose it as the design approach, as agile was the clear favourite.

Agile Development Model

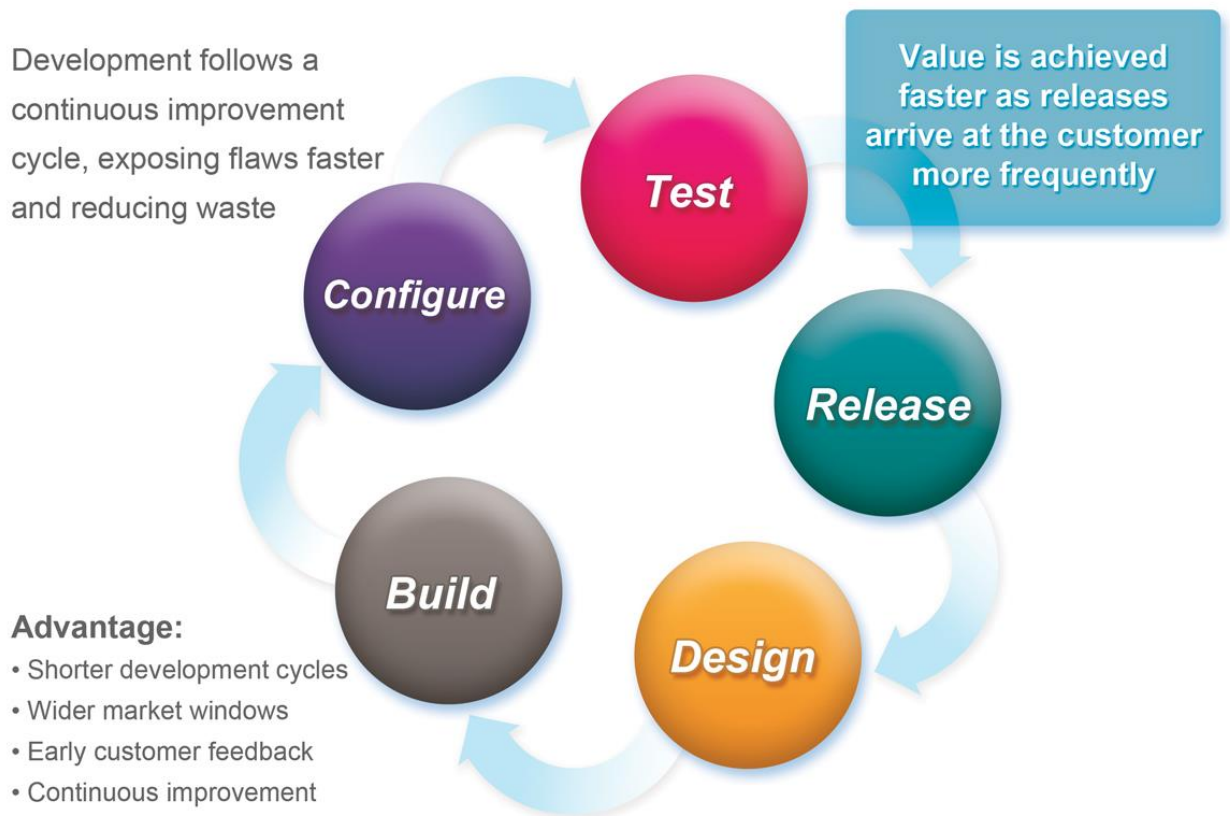
After some research, it became apparent that the agile methodology fit the requirements best. An agile methodology is simple, and its 4 stage development plan was extremely efficient. The development plan follows a 4 stage process to reach an end goal:

- Requirements Gathering.
- Design the system
- Develop and code the system
- Test the system and evaluate if design and development meet requirements.

The methodology follows this standard recursively, making improvements at every round or try. Agile development allows for usable code to be released at every stage, with new improvements coming later again, over and over. After careful consideration of my approach to the methodology of the system, it was clear that using an agile methodology enabled the system to be developed successfully and quickly. The benefits of agile development is that it increases productivity, reduces risk, it increases predictability in how the code is developed and helps with innovation (to create new code). Basically, it improves an application through repetition.

The diagram below illustrates the flow in an agile development process.

Agile Development Process



[12]

Figure 3 Agile Development Process

Project Plan (Software)

The plan for this project involved creating a plan that would enable such an ambitious project like this to be possible. The plan involved dividing the project into several smaller goals, and having a timeline to achieve each goal. There was also a minimum requirements specification defined, in the event of not reaching all the goals. The project could therefore still be a success, without having some of the features first defined in the proposal in September. The plan was created with an end goal in mind. The plan was made to ensure that a piece of deliverable software, that met the requirements of the proposal, was created. The plan involved several stages of research, followed by development and testing, and then implementation. The plan was as follows:

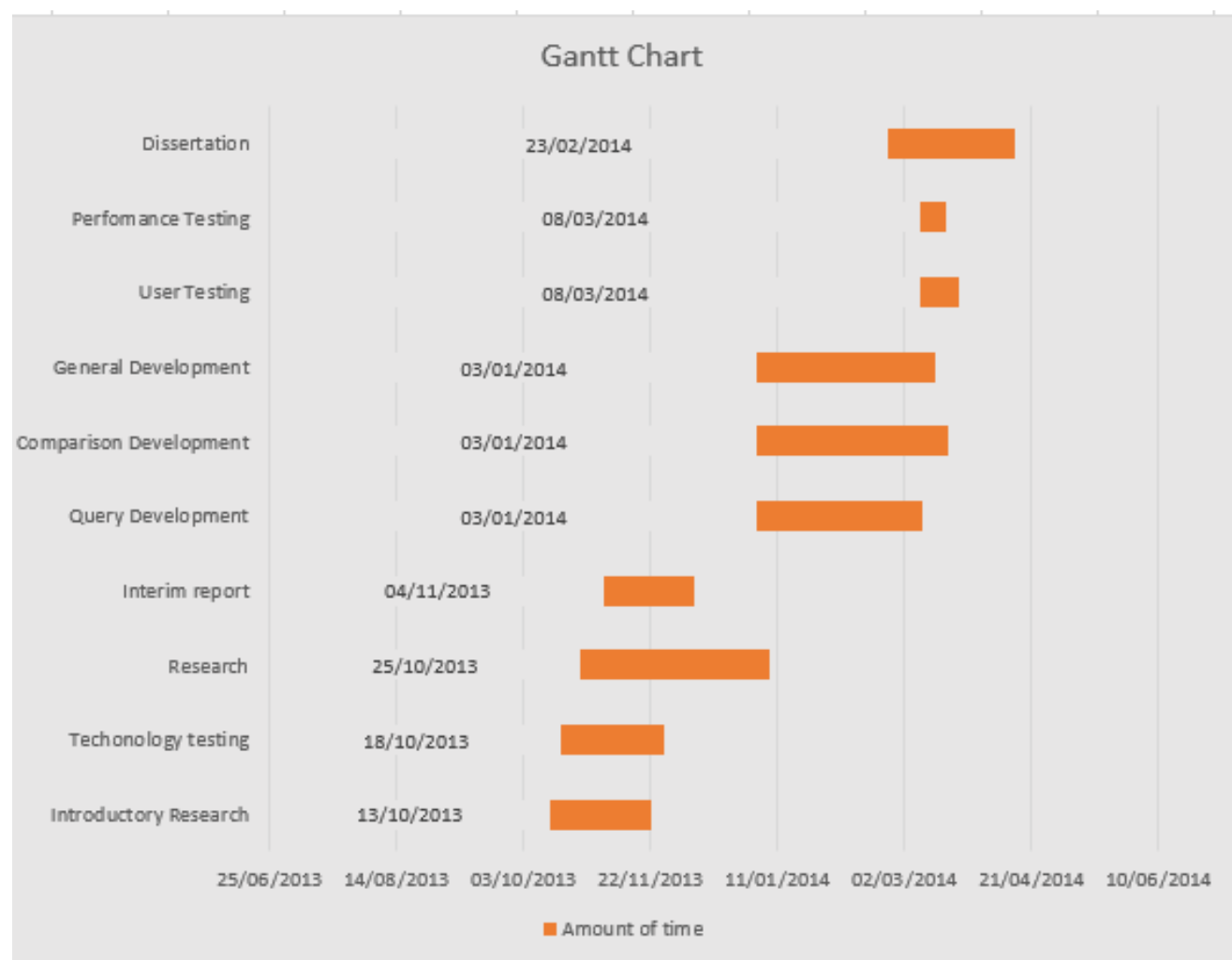


Figure 4 Gantt Project Plan

The project plan shown above is a Gannt chart which illustrated the work done on the project since the project was commence at the start of October.

Relevant Prototyping

Included in this section is a list of prototypes used in the development of the system. They include: Horizontal prototypes, vertical prototypes and mock ups of screens.

The first kind created was a horizontal prototype for the system, which represents the process a user follows while they use the system, from screen to screen. The prototype shows the system from the perspective of the user, representing how they would interact with it, and what the system would return at each stage. This prototype illustrates the flow of data from user to server and the result returned to the user.

After the success of the horizontal prototype, it was clear that a more functional prototype was needed. A mock-up was created of the two main components of the system: the image search query component and the image comparison component. This prototype illustrated the functional design and made it clear that the project was well on its way to being a success.

The prototype includes a simple web server to upload the image to a folder, where the name is changed to “origin.jpg” for simplicity when the search begins. The search begins with this image, by extracting exif data to get date and time and a few other key values. It searches for similar pictures, using this information along with an estimated location and relevant search tags the user provides. Once the images are found, the comparison process starts on these images, to come to a conclusion about what image is the most like the image provided by the user.

Also included was a use case diagram of how the user would interact with the system.

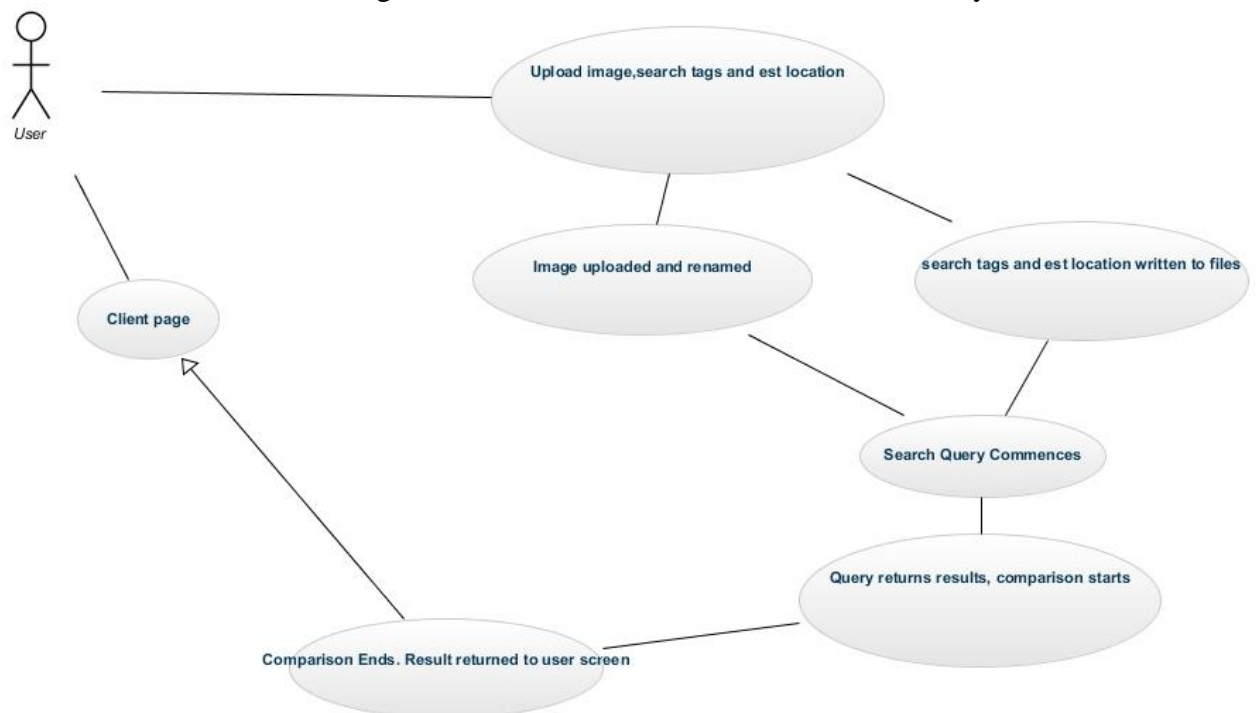


Figure 5 Use Case Diagram

Obviously, creating any prototype had to follow a set of steps too.

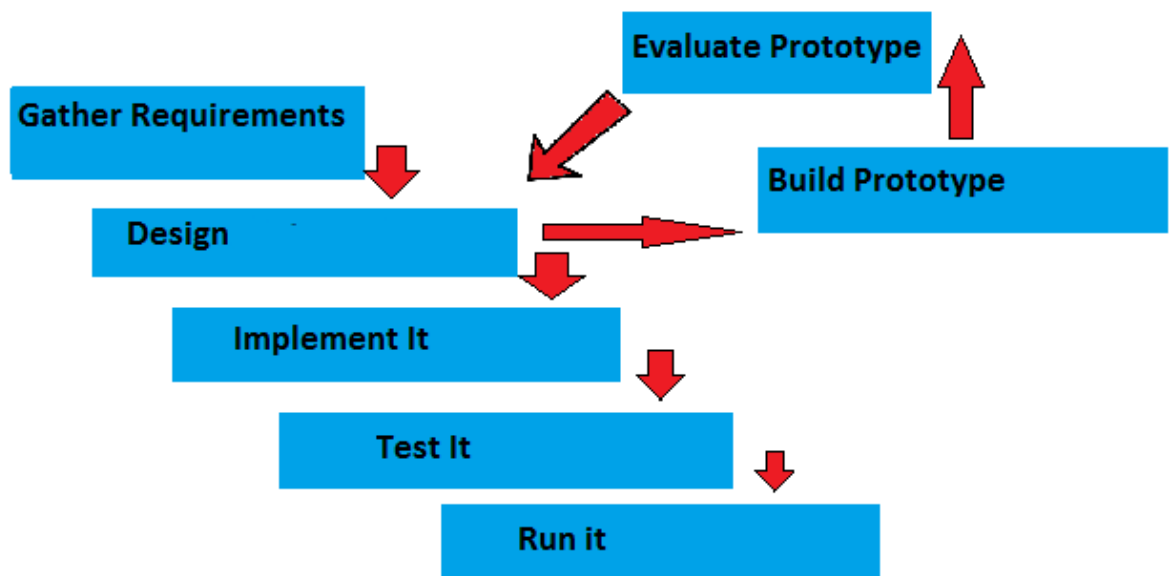


Figure 6 Prototype Process

The prototyping for this project involved prototyping it horizontally (how the user would react with system), a vertical prototype (with new functionality added to each new version), and so on. While a considerable effort was put into creating a horizontal prototype and even more effort was put into a vertical prototype, the systems functionality improved most efficiently when code was being developed for the actual project. Prototypes could illustrate and show the progress of the project, but could never match the true development of code for the project. There were a few prototype screens created for the project which reflected the initial design and hopes for the project, but eventually they weren't implemented as time was spent developing more important functionality. The current client page looks something like this (this version was a prototype):

The client screen is shown on the next page.

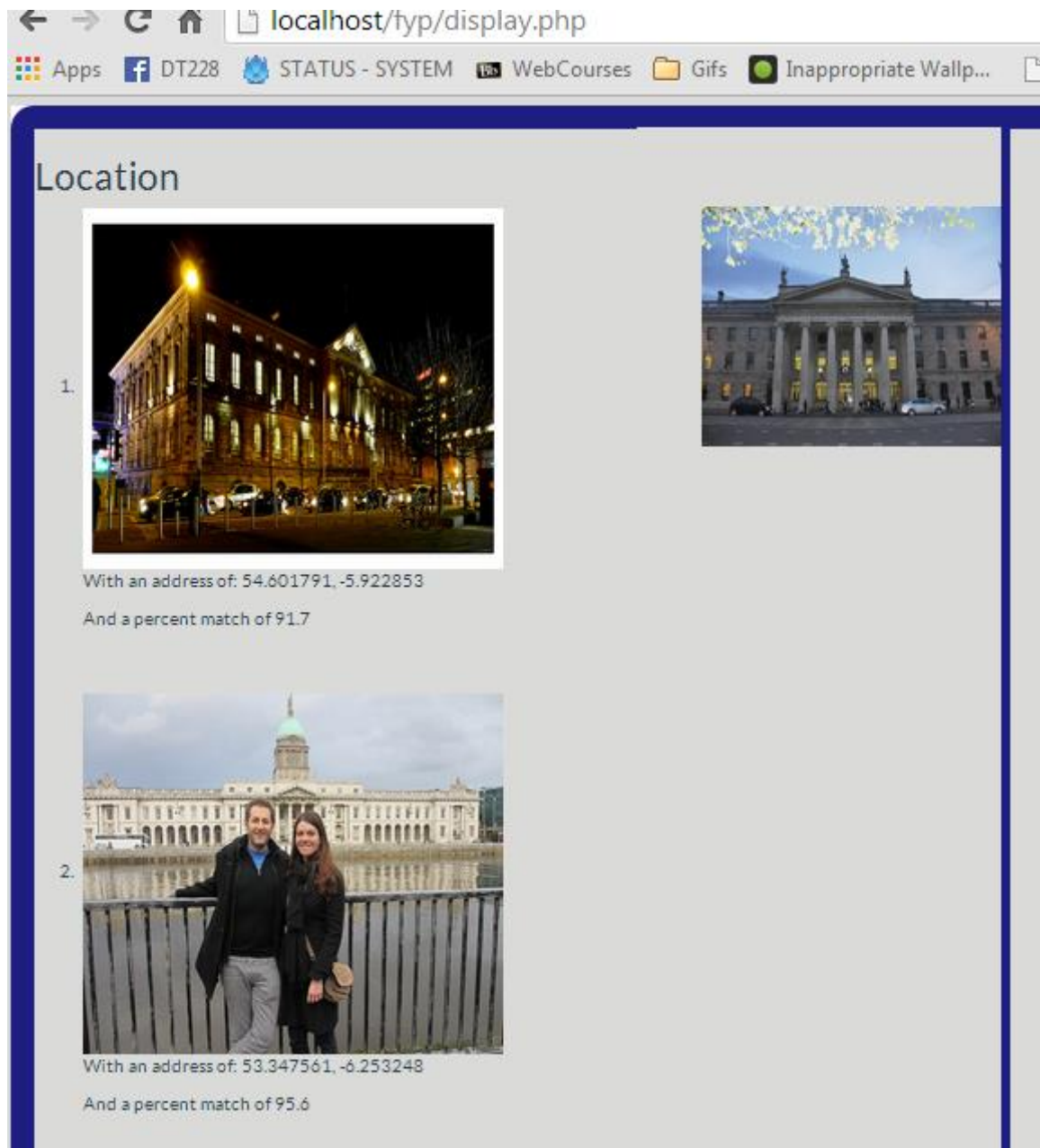


Figure 7 Client Screen 1

Therefore, the client page works by getting results send to a folder, and the page simply refreshes every 30 seconds, allowing new content to be seen.

The image on the right is the original image, and the image on the left is a potential match with the GPS address and percentage match given. The example used here is the Custom House on the quays of the river Liffey.

The client page has changed in specification quite a bit, and any relevant prototyping done this project had to be suitably changed, which proved to be time consuming. Horizontal prototypes used for the client page (before the client page was made), involved designs like so:

1. Beginning Screen (with progress screens in between it and the end)

Estimate Image

Please enter an a JPG image

Upload

Supply any relevant tags if you can, separated by comma

Figure 8 Prototype Screen 1

2. Result screen

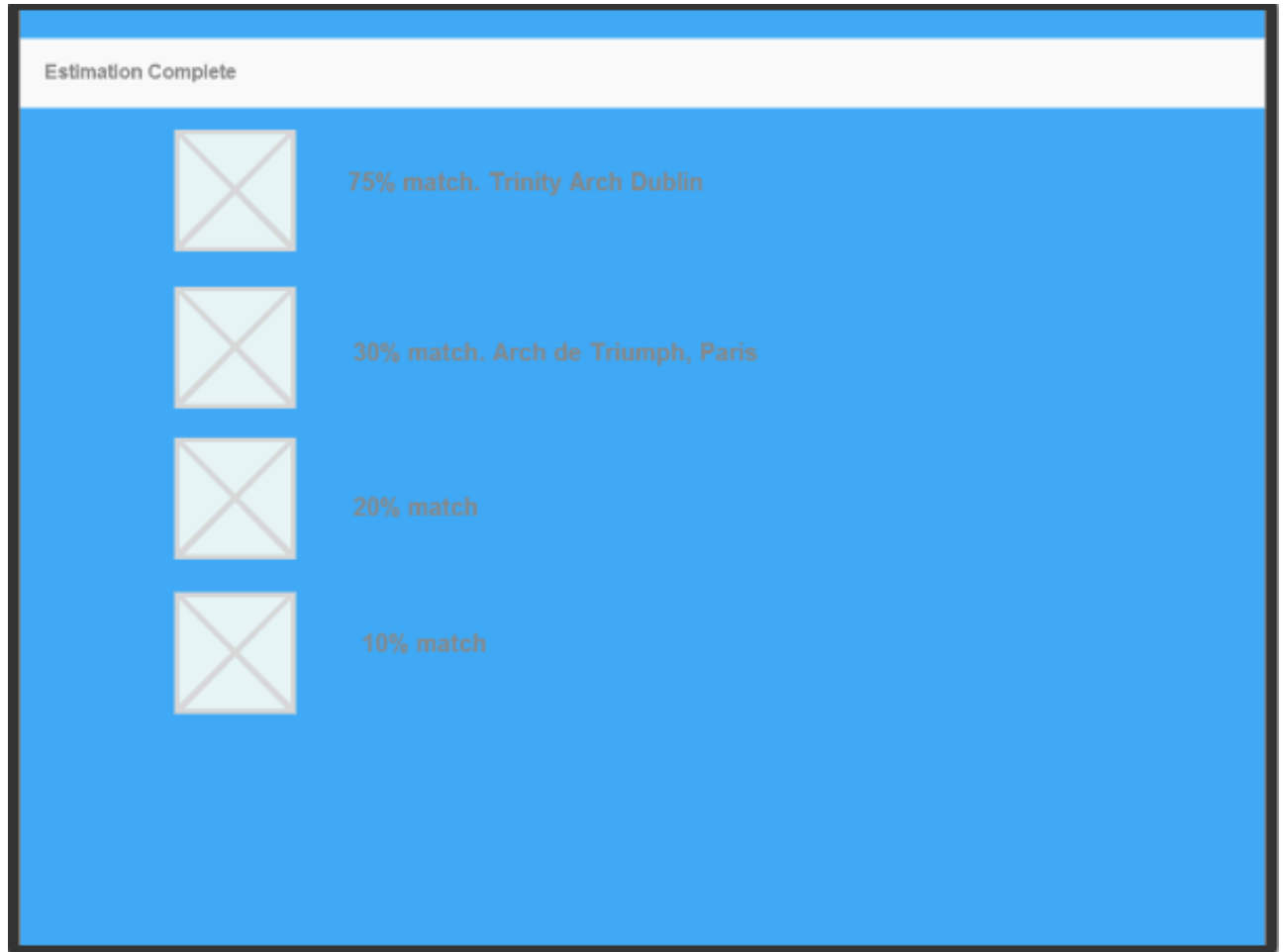


Figure 9 Prototype Screen 2

The prototype shown above took a while to develop and ultimately was reduced to something similar (with less need for Ajax web responses), the system was kept simple with a bootstrap CSS as proposed, but progress screens were kept a minimum. Progress screens were not just unfeasible, they were difficult to manage. Working with JSON proved to be more difficult than originally expected, so progress screens were quickly scrapped in favour of a simpler, less feature rich alternative. The prototypes here are made in FluidUi, which is an online prototyping tool, used by many companies including: Dell, Samsung, LinkedIn, Motorola and Oracle. It is a mobile prototyping tool that allows for screen prototypes to be created, through a design and test strategy. [13]

Conclusion

In conclusion, the methodology used in the project is key to its success. Without a decent methodology for the project to follow, it would undoubtedly fail. The design methodology keeps the project on course and allows for goals to be made, and progress to be measured. It is the guidelines which development and design should follow, and defines how the process of how the application is made should happen.

The methodology of this project was chosen to make sure the project was a success, the process involved making sure every stage in the process ran smoothly and delivered the goals required. It ensured the project continued to grow and allowed for the implementation to happen at an acceptable time. Therefore, the methodology chosen above was chosen to make sure the project was a success, and it is clear that it achieved its aim.

Development

Introduction

The purpose of this section is to provide the user or anyone interested in the system with knowledge of how the system was developed and implemented. This section is intended to make it clear how each component performs and how it was developed. The development of a project as ambitious as this was always going to be complicated set of goals, and to design a working system that could meet the requirements set out in the proposal was never going to be easy.

The project required a huge amount of development as the functionality proposed for this project is very ambitious. The development, as mentioned in the design methodology section followed an agile development approach. This approach involved a set of steps followed recursively, which allowed for improvements to be made on every “round” or “try”. The agile approach meant that a design for some part of a component was made (Flickr search for similar photos), and then the development was commenced. The development needed to match the expectations created during the design, and as a result the development followed the design process very closely. Once some functionality had been developed in a component, it would be built and tested. Usually the development allowed for small increases in functionality, so the testing was done to test each piece of functionality. The development for small increases in functionality allowed for progress to be made quickly which could be measured against the expectations throughout the process. Once testing was successful, the process would start again and the design of new functionality was started. This development style highlighted flaws in the design and development and improved the rate at which working, testing functionality was made, compared to a more traditional approach.

The development of this system therefore followed an incremental approach, and as a result, the system was successfully implemented with a majority of the goals achieved that were stated when this project was proposed.

Flickr API

Flickr is one of the most popular photo sharing and management websites on the internet. It is used by millions worldwide, and aims to provide a service whereby users can share their photos with whoever they like. Flickr desires to improve the connectivity between people who uploads and share photos. This has allowed them to create a website with billions of photos, including millions of geotagged photos.

The Flickr API is a powerful REST-based API used for searching Flickr. It can be used to search for, upload and even download photos from the service. Basically, the possibilities with this API are endless. API means Application Programming Interface. It enables a developer to use the methods in Flickr to work with photos and user data from the website. Flickr as methods such as GetInfo (which gets the information about a photo), GetSizes (which gets a list of all sizes of a photo, where the id of the photo is provided) and Search (which searches for photos, and takes parameters such as has a gps address, tags, views, upload date and so forth). The API has enabled the application to create a specific search for similar photos on Flickr that could match the photo the user uploaded.

The implementation of this API is done using a Python package, flickrapi. As mentioned earlier, it is maintained by Sybren Stuvel [5]. The package acts as a wrapper around the API and enables the use of the Flickr methods in Python. The package. The API is really advantageous in the fact that a really powerful search query can be made using it, returning a good amount of similar pictures in location, likeness and description to the upload image. There are many ways to specific the search, including only return photos with gps data, return a certain size of any image found, find images related to the image in time(getting the time and date of each object and comparing them(season and time of day)), and so forth. Overall, it's an efficient, powerful API which has access to a huge amount of images, Flickr estimates it to be 6 billion. In fact, the API has access to over 6 million "geo-tagged images".

[1]

With all APIs, there are conditions. For instance, Flickr limits the amount of calls to the API an hour to 3600. While this seems very high, reaching it is a possibility and I've reached the once during this project. For this reason, a second instance of the API has been set up. This meant creating another account on Flickr and creating another API key for use in testing if the limit was reached. From the perspective of someone searching for photos, there aren't many disadvantages to using Flickr. There are issues with finding suitable images from a search, as creating a specific search to match each query from a user is quite complicated. For example, where the tags used aren't related to any other tag, or where a date is entered wrong (2000-00-01 has appeared, and caused the search to crash). Searching by exif specifically was the aim when this project started, but not many APIs allow that now with certain legislation preventing it.

Therefore creating an exact search that could return a very accurate corpus of results is yet to be achieved. Flickr is no exception to this with its methods like search, getInfo and getSizes, the API has the ability to perform very powerful searches. Even with these methods, Flickr still has issues creating dynamic accurate searches that could return exact results to be compared. It does as well as it can in these circumstances given, and the results that come from this are astounding.

OpenCV

OpenCV as defined above is an open source computer vision library. It is developed by a community of developers to provide computer vision functionality that can be implemented in many languages (C++, Java, Android and Python etc.).

The two paragraphs below set the scope and show the huge potential in using OpenCV.

“OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.” [14]

“The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. The library is used extensively in companies, research groups and by governmental bodies. Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV’s deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan”. [14]

The library, while being open source is maintained and the development is led by ItSeez, the company which founded it.

OpenCV provides the image processing functionality needed this project, such as SIFT (Scale Invariant Feature Transform) used to find keypoints in an image, homography to detect if 2 images share a likeness (a bike shouldn't look like a building to the system), histogram calculation and comparison. OpenCV is therefore imperative to the success of this project. Its importance cannot be taken for granted, considering just how much it was used during the course of this project.

Matplotlib

“ [15]Matplotlib strives to produce publication quality 2D graphics for interactive graphing, scientific publishing, and user interface development and web application servers targeting multiple user interfaces and hardcopy output formats. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots and so forth, with just a few lines of code.” Matplotlib has allowed the project to easily create and represent data on suitable charts, like histograms. Matplotlib works excellently with OpenCV, and can display any images created during any process in OpenCV, such as histograms and plotting keypoint matches from one picture to another (using lines).

Setuptools and Docutils

These two packages are central to installing python packages and managing them easily. Setuptools allows python packages to be downloaded, installed and upgraded easily. “Docutils is a modular system for processing documentation into useful formats, such as HTML and XML. For input Docutils supports reStructuredText, an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax”. [16] Setuptools and Docutils are of extreme importance to the success of the project as they created an environment where new installations could be added to the project with ease, and processing the information into useful formats, like converting results found in xml to arrays.

Shapely

Shapely is a python package which specialises in analysis and manipulation of geographical data. It allows programmers to create geometrical operations like buffers and so forth. It is used in the system to create a simple buffer around a point. The purpose of shapely in this project is not a major one, as it is only needed to perform one particular task. It chosen to create a buffer around the estimated location which photos could be discarded when they were located outside the range of the buffer used.

Numpy

Numpy is a python package that allows for scientific computing to be done easily in Python. It is a key part of this application, as OpenCV works with Numpy to perform its calculations. It contains things like powerful n-dimensional array objects (perfect for working with histograms), sophisticated functionality for a wide variety of scientific tasks and so forth.

“Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows Numpy to seamlessly and speedily integrate with a wide variety of databases.” [17]. Numpy works well with OpenCV as it allows all the OpenCV array structures to be converted to and from Numpy arrays, which can improve calculations.

System Architecture

This chapter describes the function of each component in the system. It goes into detail on how each component performs.

Component Architecture

The technical architecture of this system is a 3 tier application, with a web application client, a business logic layer (image search and comparison), and an underlying data layer, a file system. The client has quite simple functionality and is only responsible for presenting the data it gets. This allows the client to focus on presenting and showing data to the user, using an html page. The client, is basically a web page, with an html form which enables the user to enter suitable information about the image easily. The form is supplied with 3 input areas: one input for the image, 1 for the relevant search tags and one for the estimated location. Therefore, the logic and “heavy lifting” is left to the business layer and data layer.

The business logic layer is responsible for the management and processing of data. Included in this layer is the image retrieval component, the image searching component, and the image comparison component, and the hosting component, which hosts the server (an apache server, used simply to upload the image and data about it). Finally, there is the data layer, which stores the data collected from the user, and from API searches. It interacts with the business logic layer, but cannot interact with the client layer. The client layer works the same, only interacting with the business logic layer. [18]. Shown below is a diagram of the system, highlighting the technical architecture.

The architecture diagram is shown below.

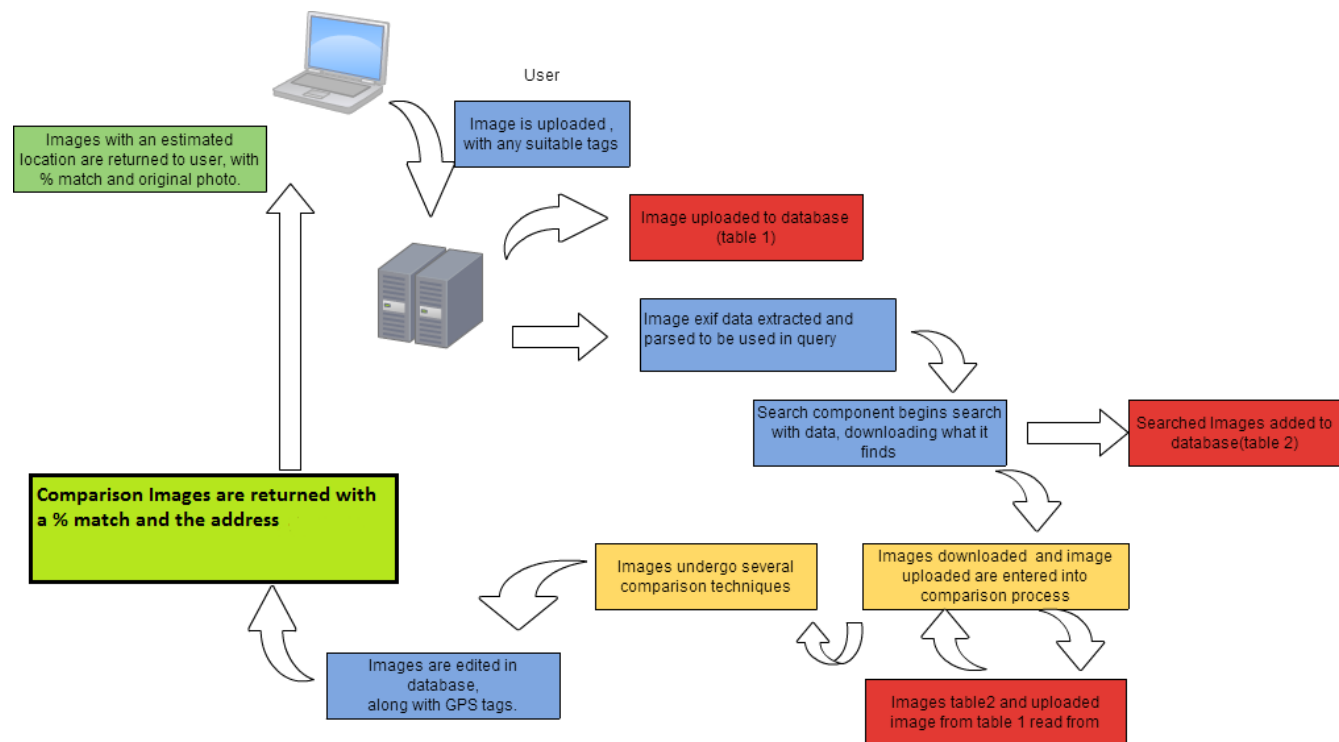


Figure 10 Architecture Diagram

Controller

The controller is responsible for interacting with all the classes and it starts the process of finding suitable images, while also returns any information found from each step in the process to the user. Each class interacts with the controller, and can send it suitable updates if desired. These updates can then be sent to the user, using a suitable display method like Ajax. The controller also takes the image uploaded and renames it to origin.jpg. This is done for the sake of simplicity, and allows for the query to begin quickly. This class can also clear folders of images, so when the controller class is called, the folders are empty and do not contain old data.

The controller code is shown on the next page, as a code snippet in a screenshot.

```

8
9 import shutil
10 import glob
11 import os, cv2
12
13 for image in glob.glob('original/*.jpg'): #Get any images uploaded.
14     #(maximum one in this folder usually.
15     os.rename(image, 'original/origin.jpg') #rename it to origin.jpg
16 files = glob.glob('compare/*')
17 for f in files:
18     os.remove(f)
19 files2 = glob.glob('compare2/*')
20 for f2 in files2:
21     os.remove(f2)
22 files3 = glob.glob('compare3/*')
23 for f3 in files3:
24     os.remove(f3)
25 execfile('query.py') #Start the next part in process.
26 def disp():
27     for images in glob.iglob("compare3/*.jpg"):
28         image = images[:-5]
29         image = image[-12:]
30         lat = image[:5]
31         long = image[-5:]
32         print(lat)
33
34

```

Figure 11 Controller Code

Client Page and Uploading Image

This component is a simple web form where the user can upload their picture to begin the search query. It is a basic form and html page, linked to a script which uploads the image to the server. Using a script like this means that the image can be uploaded securely and quickly. The script takes parameters of the image (as a file), some search tags (text input) and an estimated location (text input).

This kind of template allows for simple html to be generated and changed easily, with a form used to “post” the data a user uploads to the correct folder. The process starts off with the rendering of an html page, with a form and some div areas for returned results.

The page uses a bootstrap CSS to generate a style sheet for the page. The form on the page has 3 inputs: a file input for the image, a text input for search tags, and a text input for the estimated location. These 3 inputs are managed by html, and given suitable names for the PHP script to recognise.

Before the upload, the folder containing the image and text files is deleted of all content. The image is upload to this folder, where the query can access it, while the search tags and estimated location are written to text files which can be read from in the query process. Search tags are written to a file called tags.txt.

The estimated location is written to another text file called estLocation.txt. Once this has been successfully completed, the user is alerted that the uploading was a success, and the PHP script call the controller class.

Upload file code:

```
<?php
//Code reproduced with permission of w3schools, and edited by Ciaran Corson
$files = glob('C:/Users/Ciaran/workspacePython/ImageQuery/search/original/*'); // get all file names
foreach($files as $file){ // iterate through files
    if(is_file($file))
        unlink($file); // delete file. (deleting all files)
}
$allowedExts = array("jpeg", "jpg");
$temp = explode(".", $_FILES["file"]["name"]);
$extension = end($temp);
if (($FILES["file"]["type"] == "image/jpeg")
|| ($FILES["file"]["type"] == "image/jpg"))
{
    if ($FILES["file"]["size"] < 2000000)
    {
        if (in_array($extension, $allowedExts))
        {
            if ($FILES["file"]["error"] > 0)
            {
                echo "Return Code: " . $_FILES["file"]["error"] . "<br>";
            }
            else
            {
                echo "Upload: " . $_FILES["file"]["name"] . "<br>";
                echo "Type: " . $_FILES["file"]["type"] . "<br>";
                move_uploaded_file($_FILES["file"]["tmp_name"], 'C:/Users/Ciaran/workspacePython/ImageQuery/search/original/' . $_FILES["file"]["name"]);
            }
        }
    }
}
else
{
    echo "Invalid file";
}
if(isset($_POST['tags'])){
    $tags = $_POST['tags'];
    print("tags included: ");
    print($tags);
    print("<br>");
    $tagfile = 'C:/Users/Ciaran/workspacePython/ImageQuery/search/original/tags.txt';
    $ret = file_put_contents($tagfile, $tags);
}
```

Figure 12 Upload File Code

Upload image html page (contains bootstrap css header):

```
uploadfile.php | display.php | upload.html
1 <html><head><title>File Upload</title>
2 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
3 <link href="css/bootstrap.css" rel="stylesheet">
4 <div class="display">
5 <div class="form">
6 </head><body><h1>File Upload</h1>
7 <form action="uploadfile.php" method="POST" enctype="multipart/form-data">
8 Image name: <input name="file" type="file"><br>
9 Search tags: <input name="tags" type="text"><br>
0 Estimated Location: <input name="location" type="text"><br>
1 <input name="submit" type="submit">
2 <p><h5>For estimated location, city and country will do </h5></p>
3 </form>
4 </div>
5 </div>
6 </body>
```

Figure 13 Upload Image Page code

Once the image has been uploaded and renamed successfully, the process of searching for suitable images can be started. When this was first designed, it was proposed that the user would upload an image, and the process would be started automatically by a batch process. It became clear very quickly that would not be feasible, and was scrapped. Instead the image is uploaded, and then the process is started manually by pressing start. While this may not be ideal, it is the best possible solution to this issue.

Crude Search for Suitable Images

The architecture for the crude image search relies on being able to search for images that are similar to the uploaded image, in most ways as possible. Images relating to the sample image need to be downloaded quickly, and still have a strong match. The images that are found in the search must contain GPS data, as they are going to be compared against the uploaded photo. This is an absolute necessity, and if a potential image does not contain gps data, it is skipped in the search process. If the images returned from the search do not contain any GPS data, it makes it near impossible to estimate a location from the photo uploaded. Naturally, finding the right approach took a few attempts when the component was being created. Eventually, I decided to use a combination of the Flickr API and python to create a search. This approach was chosen as python is efficient, simple to use and is a very powerful interpretive language. Java was considered for this approach, as there are a few very powerful implementations of it available, like FlickrJ [19]. The search process involves calling the Flickr API until some sort of result is returned, either a set of images to use in the comparison, or invalid search (where the search tags and estimated location, along with the picture, fail to return a valid result). For instance, the IM2GPS uses a Python script to search the Flickr API for images that have geotags, and returns them. [1] This component retrieves the images it finds, so it is important that rather than it downloading the first 30 it finds, that it searches so effectively that it returns viable matches. Basically, it is important to go for quality over quantity. The search can allow for multiple searches to be made (if nothing is found, the system can “dumb” down the query a little bit, and return images that way. If this method fails to return anything substantial, the user is alerted that the search was unsuccessful. The search therefore needs as much information as possible to return a good set for comparison.

The architecture for this component is important to the success of the project, and is designed to collect images as quick as possible. Therefore was designed to work at a very high processing rate. In short, it takes an image, extract the exif data and stores it in suitable data type to use in a Flickr API call. The data extracted from the exif data includes date, time, light, focal length etc. It is then used to search for suitable images along with some suitable search tags and an estimated location from the user, and the search then commences. Once the search is successful, it will then download the images, and store in a specific folder.

The process of how this component is explained in the development and implementation section of this chapter.

Comparison of suitable images

In this component, the images returned from the search query are compared against the uploaded image. This component will process the uploaded image and start a comparison process, and will process the images found on queries. Once this starts, the feature detection and extraction can start.

The comparison process is broken down into several parts.

1. Feature extraction to find the key points of images.
2. Homography is then used to see how the images relate to each other in homographic way. (i.e. a building of the GPO in one picture should look like the GPO in another picture). Homography can be defined as “a 2d projective transformation that maps points in one plane to another.” [9]
3. If the homography returns a good match, the keypoints are compared, to see if both images have a set of matching keypoints to compare against. The minimal amount of keypoints needed can be changed which allows for adaptability in the comparison.
4. If the keypoints match to a satisfactory level, and the images are a decent match, the comparison image is moved to the next folder, where the histogram comparison can begin.
5. The histogram calculates a histogram for the original and comparison picture, and the histograms are compared. If they have a decent match, the images are said to be a decent match, and the image (with the GPS information and % match on the file name) is added to the web server which page can display.

The purpose of this component (both the feature matching and histogram comparison processes) is to take the original image along with any images found and compare them. Therefore, the success depends on the component working under pressure. When designing such a component, it is of vital importance that the component can work well and quickly. That therefore was the main aim during development of these components.

As an example, a search for the GPO:



Figure 14 GPO Original

Which returned results like:



Figure 15 GPO Result 1

With a 62% match.



Figure 16 GPO Result 2

With an 84% match.



Figure 17 GPO Result3

With a 61% match.

This one is a tricky match, as it matches on location, but the angle of the shot is a confusing one, coming from the top of the building. Being able to accurately determine if this photo matches the original is therefore extremely hard to do.



Figure 18 GPO Result4

With a 91% match.

The results returned can give an indication on how the process works, as the feature matching process and histogram comparison when combined, return suitable results. The comparison process works in two stages.

The feature matching and homography is to remove all results that do not look similar to the original image in shape and features and likeness. I.e., it does its best to remove pictures of people, insides of buildings and cars when it is looking for a picture of a building, like the GPO, shown above.

The second part is to remove images that look like the original in features, but aren't a close match. Colour histograms, as explained above try to match photos on numerical colour values, and therefore it can be said that it is used to clean up the search and remove any remaining values that might not be a match.

This whole process is extremely resource heavy, and relies on the underlying system and software used to ensure that it can perform the calculations it needs. While the process works on feature matching and homography, it iterates through the images in a loop, using glob. There could be up to 50 image present when it starts this process. To calculate the keypoints and homography detections, along with nearest neighbours is a very resource intensive task, which can sometimes need a lot of memory and processing power. Having to do this multiple times in a loop can be even more resource intensive, so the software and hardware that this process relies on must be up to standard. Next in the process is the histogram calculation and comparison, and like the feature matching process it is also resource intensive, as it iterates through images in a loop as well. This process needs to have the hardware suitable to work well. Therefore the process of comparing the images has the most priority in terms of resources and hardware needed.

Returning a result

At this point, some results have been found. The system moves these images, along with a % match and the gps address (both on the file name) to a folder that the webserver has access to. The display page uses a refresh Ajax call to refresh the page every 10 seconds, showing new images as they are displayed.

These images are displayed to the user, along with the % match, the geographical location, and a link to google maps where the user can verify that the location is correct.

Screenshot of results page code below.


```

<html> <!-- Used to implement Angular-->
<head>
<title>Location</title>
<link rel="stylesheet" href="css/bootstrap.css">
<link href="css/bootstrap.css" rel="stylesheet">
<script type="text/javascript">
    setTimeout('window.location.href=window.location.href;', 10000);
</script>
</head>
<body>
<div id="images" class="display1">
<h2>Estimated Locations</h2>
<?php
$dirname = "images/";
$origindir = "origin/";
$images = glob($dirname."*.jpg");
$origin = glob($origindir."*.jpg");
foreach($origin as $img){

    echo '';
}
echo '<br>';
foreach($images as $image) {
    $gps = str_replace('.jpg', '', $image);
    $temp = substr($gps, -26);
    $addr = substr($temp, 1, 20);
    $percent = substr($temp, -4);
    echo '<li>
        
        <p>With an address of: ' . $addr. '</p><p>And a percent match of ' . $percent. '</p><br>
    </li>';
    print("<br/>");
}
echo '</li>';
?>
</div>

```

Figure 19 Client Page code

It uses a JavaScript time out function called `setTimeout` to refresh the page every 10 seconds. This allows for the page to be refreshed, and new content can arrive. It albeit is a little lazy compared to dynamically refreshing when new content arrives, but with time limits like this, it was more important to make some working functionality for the project. This keeps loading the images simple and easy.

A for loop is used to iterate through the folder of images that might be a match to the original. Glob works by finding all the .jpg images and the for loop loops through them, displaying them in a list.

Development and Implementation (of main components)

The development of the system was done in an incremental way. Both the image query and image comparison components at first, where a piece of functionality would be developed and then tested. It followed this procedure as the system improved in size and features. The development of each component included developing the code which followed a simple list of steps on a piece of paper, and tests were run at each stage to see how the code and results fared in respect to the expectations at that stage.

Image Query

The image query component comprises of:

The script works as follows:

1. Read in the image, with some suitable search tags and estimated location, found in the text files provided from user input.
2. The estimated location is changed to a gps address. The location, when entered by the user does not need to be specific (city and country will do), but a country must be entered to round down the gps.
 - a. This is done using Python's Geocoder package to geocode the address into a latitude and longitude.
E.g. `centerLocation = Geocoder.Geocode("Dublin, Ireland")`
3. Collect exif data from the image, using the Python Image Library package.
 - a. This is done using a function: `def get_field()`, which gets certain fields like date and time, and focal length.
4. Extract Month and Hour from datetime, to create efficient search parameters. This is done using the Python `Strptime()` method.
 - a. It allows for determining what time of day and what season the picture was taken in (18:00 is evening, March is spring). This is done using a conditional statement to detect what season and time of day.
5. After season and time of day are calculated, the Flickr query is started. The application works by using Stuvell's [5] implementation of the Flickr photos search method, which is called "walk". [5] It searches for images, using certain parameters.
 - a. Gps data is present, tags associated with image match relevant search tags provided by user, accuracy level of 12 (higher the number, the less accurate. 11 being city level), public photos only (removing private photos), and only searching for actual images, no screenshots etc.
 - b. The search returns the photo id for each photo found, which enables the search to be narrowed down further on.
6. The location of each photo found in search is collected and turned into a GPS point. A buffer is created around the estimated location point created before this, and only points which are within a certain distance of the point are needed. [20]
 - a. The photos outside the buffer are discarded, and the photos inside are kept, for the next process.
7. Next, the date and time are collected from the photos left in step 6a. Strip time is used here again to ascertain time of day and season. If the season or time matches, the image can be downloaded.

8. The images remaining after this can be downloaded once they are prepared sufficiently.
9. The URL is stripped to leave the last 10 characters and .jpg at the end. The search returns a _s.jpg version of each image, which is a thumbnail version. This is removed so the urllib retrieve method can work efficiently. The _s and full name of link are removed and a 10 character long name is left, which is downloaded into a suitable folder, where the comparison can begin.

The Flickr photos.search method returns a result similar to something like this: With the photos being found page by page along with a photo id, owner id, and several other properties of the particular image.

Example Response

This method returns the standard photo list xml:

```
<photos page="2" pages="89" perpage="10" total="881">
  <photo id="2636" owner="47058503995@N01"
    secret="a123456" server="2" title="test_04"
    ispublic="1" isfriend="0" isfamily="0" />
  <photo id="2635" owner="47058503995@N01"
    secret="b123456" server="2" title="test_03"
    ispublic="0" isfriend="1" isfamily="1" />
  <photo id="2633" owner="47058503995@N01"
    secret="c123456" server="2" title="test_01"
    ispublic="1" isfriend="0" isfamily="0" />
  <photo id="2610" owner="12037949754@N01"
    secret="d123456" server="2" title="00_tall"
    ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

Figure 20 Flickr Response Example

The development of this component took many things into consideration, and built up an effective search query.

- A search buffer limits the area photos can be found in. If the estimated location is Dublin, the buffer must find photos in Dublin, and only Dublin. There were problems when developing where the search found photos in Colchester, England. The buffer needed to be accurate, but also accommodate a good search error. I.e., if the buffer was too small, it'd only find matches very near where the Dublin gps point is centred. This was a challenge that needed to be overcome
- Searching for images on Flickr raises many challenges, including search tags. Flickr's main search method (photos.search) allows the method to be supplied with lots of parameters, and some don't work as well as they should (geocontext allows search to be limited to outdoors or indoor, but fails to work), and so forth(as seen in code below, in the Flickr walk method(python name for the Flickr method))

```

7 try: #Try to search, (with an except fro FlickrErrors) (works like a try catch)
8     flickr = flickrapi.FlickrAPI(api_key, secret) #Search for photos
9     for photo in flickr.walk(flickrKey = api_key,
10                             ispublic="1", #Returns only public photos
11                             media="photos",
12                             ext = relSearchTags, #Photos with title, descripti:
13                             sgeo = "1", #Hasgeo 1 = has gps tags, 0 = does no
14                             gs = relSearchTags, #Tags included (coming from
15                             geo_context = "0", #0 undefined, 1 indoors, 2 out
16                             #Problem with GeoContext, as it terminates search.
17                             accuracy = "12", #Accuracy level, 10 being city
18                             content_type = "1",
19                             extras = "date_upload, views",
20                             ):
21         photoid = photo.get("id") #Get the photo ID

```

Figure 21 Flickr walk method

- For geotagged photos, the accuracy level can be limited (with 10 being city level, and 16 being city, higher the number the more accurate) etc. This shows that the search can be made very specific and powerful, but it also meant that sometimes it could limit results. This search functionality needed to be managed well if it was going to succeed.
- Limiting the search goes further than just search parameters. The search method gets the photoID, and the photoID is used to get a geographical location for the image, using the Flickr method: geo. Getlocation. This method works by taking the photoID, and getting the geolocation associated with the image on Flickr. When this is complete, it converts the string address into a geographical point, which can be checked against the buffer. If the picture is inside the buffer, the code moves on to stripping the image details (like date and time) to see if the image was taken in same season or at same time of day.
- The search then prepares the link for download. The actual link to the file is gotten using photos. GetSizes, along with the url for the photo. This method returns a thumbnail version of the image i.e. imag122121_s.jpg (_s means small), so the image is stripped of that _s to return the full size image. The full url is stripped to leave the photo id (7891326_526566.jpg), which can be downloaded, along with the gps information. Getting the GPS information is tricky as Flickr removes exif data from any image downloaded. Therefore, the gps address is appended onto the end of the filename, like so:

```

138 ..... gseason= spring
139 ..... if "May" in gdate or "June" in gdate or "July" in gdate:
140 .....     gseason="Summer"
141 .....     if "August" in gdate or "September" in gdate or "October" in gdate:
142 .....         gseason="Autumn"
143 .....     if(gseason is dseason):
144 .....         path, dirs, files = os.walk("compare/").next()
145 .....         file_count = len(files)#Current amount of files in folder.
146 .....         limit = 30 #Amount of files in folder(limiting how many to download)
147 .....         if file_count < limit: #if amount of files is less than limit, download next file(in loop)
148 .....             addr = str(addr)#Convert address to string.
149 .....             link = link[:-4]#remove the jpg,
150 .....             link += addr #Append the address

```

Figure 22 determine season

This component then progresses on to download the file, to a folder named “compare/” as seen above, using a suitable created beforehand. The console log below shows how this process is dealt with.

```
'picture not within area', (31.243434, 121.48644))
'picture not within area', (51.524863, -0.077081))
'picture not within area', (51.524997, -0.076734))
'picture not within area', (-37.801789, 144.951736))
'picture not within area', (53.382394, -1.472157))
'picture not within area', (51.562347, -0.23772))
'picture not within area', (22.572061, 88.347877))
ownloading picture: 12706465273 4f679fa663(53.349392, -6.260566).jpg
```

Figure 23 Output log

Image Comparison – Feature Matching

The next step in the process is to begin the image comparison, comparing each image in the compare folder to the original image. The original image has keypoints on it (edges of buildings and so forth), and they form the basis of the comparison process. The minimal amount of keypoints is set at the start. The original image is read into the program, and a for loop starts iterating through the images in a folder. Everything in the comparison happens with this for loop. Sift is commenced on both images (original and image in compare loop). And the keypoints and descriptors are found (detected and computed). In this component OpenCV and Numpy is used to perform the operations (including loading SIFT).

OpenCV can be implemented in Python, and allows the use of its computer vision functionality. The open source library is discussed in detail in its own chapter later in this report.

The feature matching (or feature detection) process involves computing values of image information, whereby a decision is made at every “keypoint” where another image could share that keypoint. The choice of feature matcher used is Flann - Fast Library for Approximate Nearest Neighbours. “It contains a collection of algorithms optimized for fast nearest neighbour search in large datasets and for high dimensional features”. [7] The other choice is a brute force matcher which brute forces every point against every point in the other picture, until matches are found, while Flann is a little more intricate and is better at working with larger data sets(it is quicker). For FLANN based matcher, we need to pass two dictionaries which specifies the algorithm to be used, its related parameters (index and search). [14].

SIFT is an algorithm which computes the descriptors for an image, and extracts its keypoints. It does this by following a set of defined steps:

1. Scale-space extrema detection: This is where the keypoints for the image are detected. Gaussian filters are used different scales to create a blurring which allows the edges to be detected, and the keypoints founds.
2. Keypoint Localisation: “Once potential keypoints locations are found, they have are refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected.” [7]

3. Orientation Assignment: “Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching.” [7]
4. Keypoint Descriptor: Next a keypoint descriptor is created for the image, a 16x16 neighbourhood around the keypoint is created. This is divided into 16 sub blocks of 4x4. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc. [7]
5. Keypoint Matching: “Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per David Lowe’s research paper.” [7]

The screenshot below highlights the process of setting the feature matching process up, with SIFT creating the first steps needed. This code has been reproduced with the permission of OpenCV tutorials. The tutorials are provided by OpenCV [7], but the link is here. [Tutorials link](#)

```
MIN_MATCH_COUNT = 40 #Min amount of keypoints matched, for image considered to be alike.

img1 = cv2.imread('original/origin.jpg',0) # queryImage
for image in glob.iglob("compare/*.jpg"):
    #print(image)
    img2 = cv2.imread(image) # comparison images

    # Initiate SIFT detector
    sift = cv2.SIFT()

    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)
```

Figure 24 feature matching part1

Next the Flann process is started. Flann is a nearest neighbour matcher that matches descriptors in an image. The index and search parameters are needed for the descriptor matcher, and the KNN match works by finding the k nearest neighbours, where k currently = 2. It then stores all the good matches found in the Flann KNN matcher.

```
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 10)
search_params = dict(checks = 200) #higher the check, better the precision

#Flann (Fast Library for Approximate Nearest Neighbors)
flann = cv2.FlannBasedMatcher(index_params, search_params)

matches = flann.knnMatch(des1,des2,k=2)

# store all the good matches with regards to Lowe's ratio test. (David Lowe being the creator of SIFT)
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
```

Figure 25 feature matching part2

If the amount of matches found is higher than the min match count, the image is kept. If it's lower, it is discarded. Source and destination points are created, and homography is performed on the image. The keypoints matches are drawn together, using polylines. (It isn't shown to the user at the display page, but can be used if needed).

```
#If the amount of matches is greater than the minimal match count(accepted amount of matches)
if len(good)>MIN_MATCH_COUNT:
    src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape((-1,1,2))
    dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape((-1,1,2))

    M, mask = cv2.findHomography(src_pts,dst_pts,0,5.0)
    matchesMask = mask.ravel().tolist()

    h,w = img1.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape((-1,1,2))
    dst = cv2.perspectiveTransform(pts,M)

    img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)
```

Figure 26 feature matching part 3

Next, the process reaches the end of the “if loop”, and the “except” is either ignored, or it prints that there are not enough keypoints. If there are not enough, the image is skipped, and the process moves onto the next image. If this next image meets the minimum amount of keypoints required, the draw parameters are created, and the images are matched. Images with a sufficient number of matches are moved to the next folder for comparison, “compare2/” The python method, `execfile()` calls the next step in the process, the histogram comparison.

The comparison process naturally has room for improvement. While it reaches a very high standard and works well under pressure, it still has room for improvement, as

there are many more feature detections methods to implement into the system if desired. Shown below is a keypoint matching process in OpenCV.

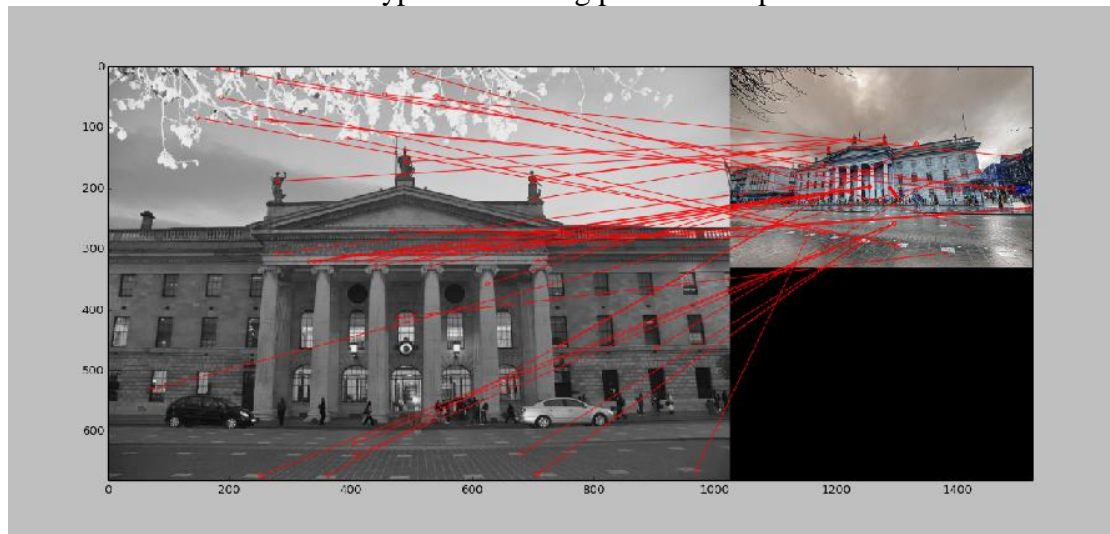


Figure 27 Draw Keypoint Matches

Image Comparison – Histogram comparison

Next, the system compares the histogram of each picture in the comparison folder against the original. The comparison begins with reading in the original image, and a split is performed on the image. A split involves splitting the image into 3 channels, blue green and red. A multidimensional array is created to store the values of each channel, and then the histogram calculation for the original image is started. Histograms are counts of data from the image predefined into a set of bins.

This code has been reproduced with permission of OpenCV tutorials, found here

http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html [7]

```

7 import numpy as np
8 import time
9 import os, math
10 import glob, gc, shutil
11 from PIL import Image
12 from matplotlib import pyplot as plt
13 import display
14 rootdir = 'compare2/' #Files end up in this folder after clearing the feature matching test. Only matches get this far.
15 im1 = cv2.imread("original/origin.jpg")
16 b1,g1,r1 = cv2.split(im1)
17 h1 = np.zeros((300,256,3))
18 bins1 = np.arange(256).reshape(256,1)
19 color1 = [ (255,0,0), (0,255,0), (0,0,255)]
20 for item1,col1 in zip([b1,g1,r1],color1):
21     hist_item1 = cv2.calcHist([item1],[0],None,[256],[0,255]) #calculate histogram for each image.
22     cv2.normalize(hist_item1,hist_item1,0,255,cv2.NORM_MINMAX)
23     hist1=np.int32(np.around(hist_item1))
24     pts = np.column_stack((bins1,hist1))
25     cv2.polylines(h1,[pts],False,col1)

```

Figure 28 Histogram PartI

Next, the histogram calculation is performed on the comparison images using a “for loop”. The process is the same as the other histogram calculation, but iterates through the process right up until the histograms are compared. The process repeated for each image in the compare2 folder.

A histogram comparison seeks to see how similar or different two image histograms are. An image histogram is a histogram which represents the tonal distribution of an image in a graphical way. A histogram can keep count not only of colour intensities, but of whatever image features that we want to measure (i.e. gradients, directions and

so on.) It does this by plotting the pixels for each tonal value, and thereby allows the tonal distribution for an image to be judged. This can be used to create a set of numerical values which can be compared against one another. It is used to remove all the incorrect values left.

A sample histogram calculating colour values would look like this:

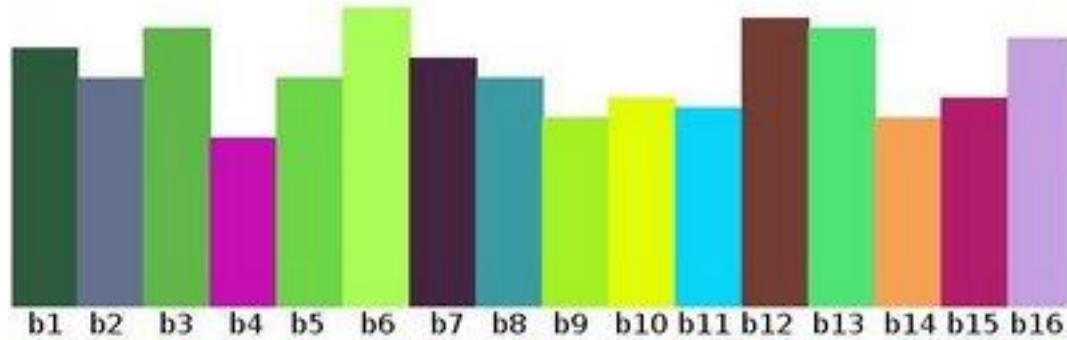


Figure 29 Histogram results

This histogram, while not being an accurate representation of any images found in testing still gives a clear indication of how image similarities such as colour can be shown as a numerical value.

The histogram code is shown below.

```
for images in glob.glob("compare2/*.jpg"):
    im2=cv2.imread(images)
    b,g,r = cv2.split(im2)
    h = np.zeros((300,256,3))
    bins = np.arange(256).reshape(256,1) # np.arange returns evenly spaced values within a given interval,
    color = [ (255,0,0),(0,255,0),(0,0,255) ] #Color array for each image
    for item,col in zip([b,g,r],color):
        hist_item = cv2.calcHist([item],[0],None,[256],[0,255]) #calculate histogram for each image.
        cv2.normalize(hist_item,hist_item,0,255,cv2.NORM_MINMAX)
        hist=np.int32(np.around(hist_item))
        pts = np.column_stack((bins,hist))
        cv2.polylines(h,[pts],False,col)
    base_test1 = cv2.compareHist(hist_item,hist_item1,0)#4 methods (0-3)(Need to test all 4 performance wise)
    #print(base_test1)
    if 0.4 > base_test1: #0.4 allows for a 40% difference, which after testing is the most suitable range.
        pc = round(base_test1,3)
        pc = pc * 100 #Convert PC to a percent
        pc = pc / 1
        pc= str(pc)
        pc = "[" + pc + "%" + "]" #Percentage match
        res = pc+".jpg"
        image = images[:-4] + res
```

Figure 30 Histogram part 2

If it matches, it is copied into another folder, called compare3. The results of compare 3 are then copied to a folder that xampp can access (a folder in C:\xampp\htdocs). In the event that images are compared further, the compare3 folder could be used as the results would be significantly lower than the first compare folder, which could be beneficial in the future.

It then copies the files to compare3/ and called a display method in a display class which copied the files again to xampp.

```
def disp():
    for images in glob.iglob("compare3/*.jpg"):
        dest = "C:/xampp/htdocs/fyp/images"
        shutil.copy(images, dest)
```

Figure 31 Display method code

System Environment and Technologies Used

The developing environment to create such a system is a complex one. Many components interact with each other, and perform operations on the data too. The environment of the system is critical to the success of the project. This project is admittedly quite an ambitious endeavour. To fully meet the requirements of such a system, the development languages used in this project had to be well researched, and the environment that it works on must be able to handle the pressure. The system is comprised of many different architectures all working together.

It is primarily a web application for image location estimation, so it relies heavily on interaction with APIs, and user interaction. Overall, the environment used is all about performance and usability. The environment has been chosen after a lot of in depth research, and it has been chosen as it meets a required standard. The environment for such a project was always subject to change, but having a defined starting point to start from made the environment simple to maintain.

The system is developed in a number of environments, ranging from Eclipse (an IDE) to notepad++ and wamp for testing the web server. The environment used is divided into separate areas and each are specific for the requirement they are used.

Unfortunately, there was never going to be “one solution fits all” solution for developing the system. Each sub section of the environment is described below.

Web Server

When planning for a system to host everything, it was important that the environment that hosted all the components could work well under pressure. When deciding what to use, many different approaches were considered. These approaches included:

1. Hosting everything on a virtual machine, accessible by port forwarding.
2. Having an apache server for uploading the image to a folder, and letting python work in the background on a normal system once the image was upload.

Eventually, the approach taken was just an apache server and python running in the background. It was chosen for its simplicity. All the components need some sort of system to work off, and which meant the solution used needed to be efficient, powerful and good under pressure. The data that each component manages will range in size, but there is a potential for a huge amount of data transfer (in the event of the search query finding over 50 images), therefore the system must be able to run comfortably in the upper limits. For this reason, it became apparent that any potential system would need to be tested before any work could be done on it. The potential environments included LAMP, WAMP, XAMPP, simple apache servers and so forth. In the end, a simple apache server was chosen as it fits the requirements best. Having apache, php and MySQL supported by a windows background allowed for simplicity. The hosting server only needs to host, and upload images to a folder.

LAMP (Linux, Apache, MySQL and PHP) was considered too, whereby Python could run within the LAMP virtual machine. LAMP works well because it's simple, and operates well with other applications, but wasn't as practical to set up and keep maintained (new code and so forth), when setting the system up as a simple apache server, and developing on windows allowed for the system to be used simply, and tested by others. The XAMPP (apache) control windows like this.

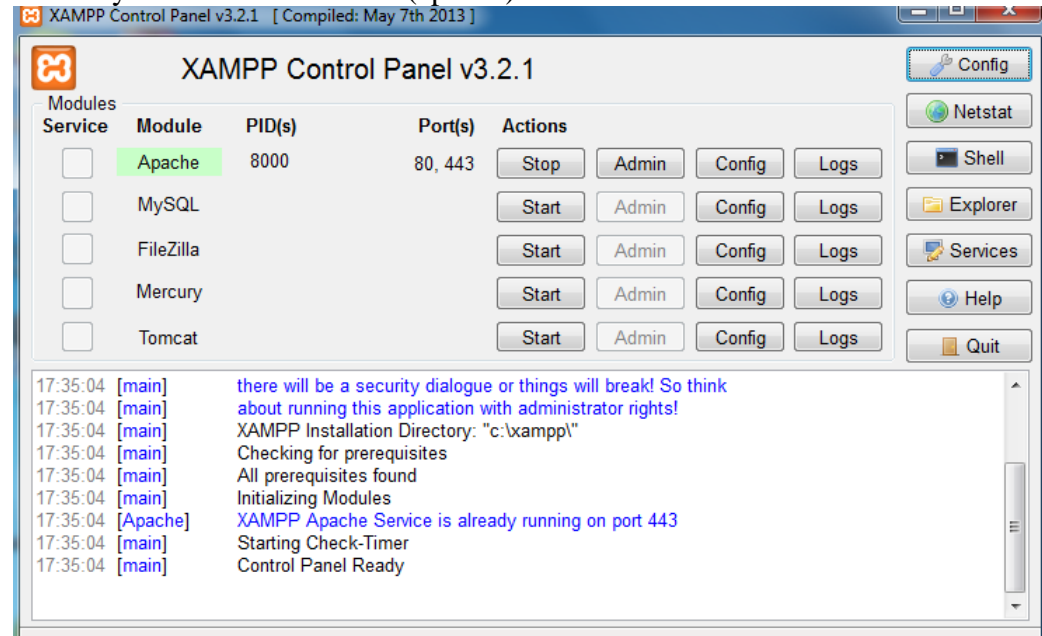


Figure 32 Xampp Control Panel

Operating System and Hardware

When this project was started, the potential operating systems that could be used in the project were Windows and Linux. Both were considered equally, and a lot of testing was done on each system to see how the matched up against the requirements. In fact, some development happened on both before a decision was made regarding the OS. The major advantage of UNIX is that has an advantage in that it is very suitable for Python development. However, it was set up as a version of Ubuntu on an old PC which wasn't being used as much as my own laptop, which had Windows 7. It quickly became an inconvenience to use it, as the machine it was working off had no internet connection without an Ethernet cable or a new wireless card, so from there it became obvious that it would be best suited to work as a backup in the event that my own machine failed.

Windows was used to develop the project, as eclipse and PyDev could be loaded onto it with ease, and I could concentrate on development. The main advantage of using windows is that I had a great deal experience in it from using it most often. It holds an advantage from my perspective, with some bias in that I feel it is better than the alternatives.

The system is being ran on a 64 bit version of Windows 7, with 8gb of ram and an i5 processor(with a speed of 2.5ghz), while the alternative option of a Ubuntu machine (used primarily as a backup) is a 32 bit version running on an x86 dual core Pentium processor with 2gb of ram. It can be easily seen just how demanding the application is

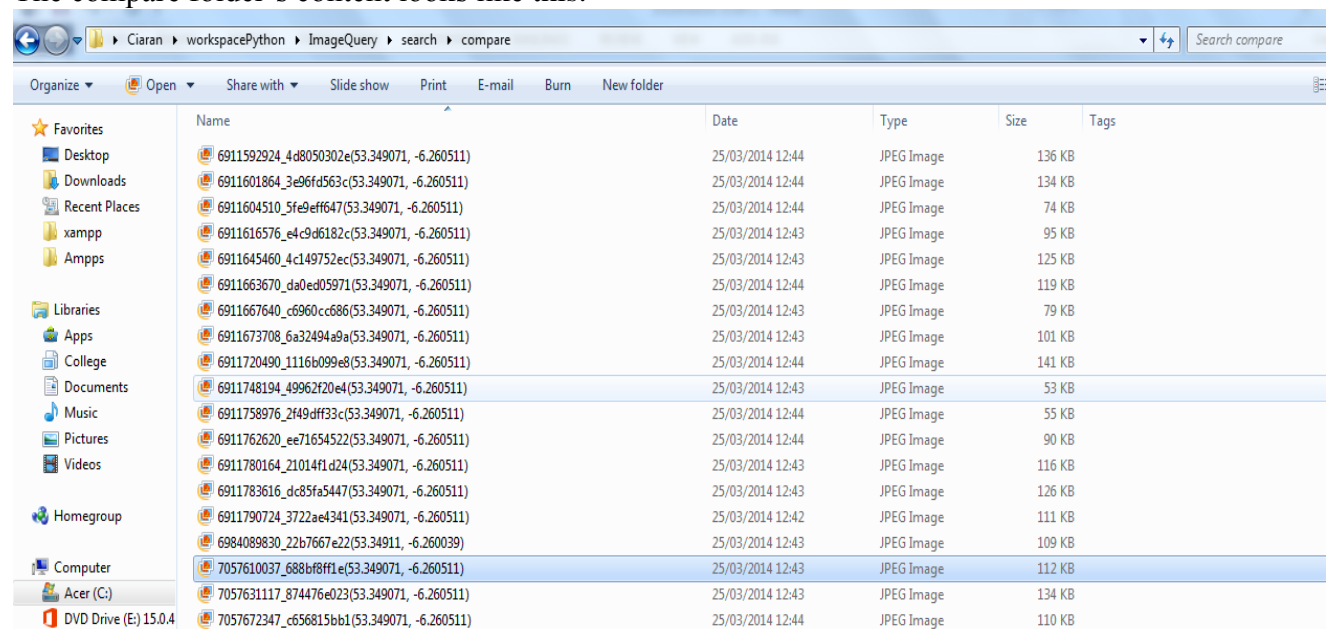
and how the Ubuntu machine simply does not have the processing capability to run this application well. The windows machine is far more superior to it and was always going to be superior, even if the other machine implemented a virtual machine.

File Storage

File storage for this project was always going to be an interesting topic. When research began, there were many options to choose from: ordinary file and folder storage, relational databases and so forth. Choosing the file storage was important, but it was not main priority. At first, a MySQL database was researched to see how viable it would be the store the uploaded image and images found in the search. Eventually, it became quite hard to access them, and store them. A simple file storage system consisting of two folders worked out better. Iteration was done through the compare folder to access the photos found in the Flickr search. It was chosen for its simplicity and ease of use.

The folders are named “original” and “compare”, where original stores the original photo from user (renamed to origin.jpg when uploaded) and compare stores any images found in the search.

Therefore, the data storage used in this project could be solved quickly, as the solution available was a simple one. The file system solution solved my file storage issues quickly and was easily implemented, therefore I could concentrate on the harder aspects of the project, knowing that the file storage solution had been created easily. The compare folder’s content looks like this.



Name	Date	Type	Size	Tags
6911592924_4d8050302e(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	136 KB	
6911601864_3e96fd563c(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	134 KB	
6911604510_5fe9eff647(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	74 KB	
6911616576_e4c9d6182c(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	95 KB	
6911645460_4c149752ec(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	125 KB	
6911663670_da0ed05971(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	119 KB	
6911667640_c6960cc686(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	79 KB	
6911673708_6a3249a9a(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	101 KB	
6911720490_1116b099e8(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	141 KB	
6911748194_49962f20e4(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	53 KB	
6911758976_2f49dff33c(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	55 KB	
6911762620_ee71654522(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	90 KB	
6911780164_21014f1d24(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	116 KB	
6911783616_dc85fa5447(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	126 KB	
6911790724_3722ae4341(53.349071, -6.260511)	25/03/2014 12:42	JPEG Image	111 KB	
6984089830_22b7667e22(53.34911, -6.260039)	25/03/2014 12:43	JPEG Image	109 KB	
7057610037_688bf8ff1e(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	112 KB	
7057631117_874476e023(53.349071, -6.260511)	25/03/2014 12:43	JPEG Image	134 KB	
7057672347_c656815bb1(53.349071, -6.260511)	25/03/2014 12:44	JPEG Image	110 KB	

Figure 33 Compare Folder

Testing

Introduction

The testing of this project, or any project is a very important area of concern when a piece of software is being implemented. The testing is done to ensure that the functionality created meets requirements set, and also does not contain bugs. It also ensures the code created is of a certain standard. Testing this project meant that it had to be tested at every step in the development process, from when the project was commenced, right up to when it was finished. The testing for this project was imperative to the success of this project, and as a result the testing approach chosen for this project was to ensure that the project was a success.

The testing for this project at first was going to be a mixture of performance testing (by users), and unit tests. This quickly became redundant and it was clear that it would not be possible. I had proposed to release a web server to allow multiple users test it, but it became clear that the system cannot work concurrently, and the software if it were to become a viable product would need to be installed as an application on the client's machine. This made it clear that a new approach was needed. Performance testing (done by myself), and usability testing (done by users on my machine performing test cases) was needed to test and evaluate the success of the project.

Performance Testing

Performance testing for this project was meant to establish just how successful the project was at estimating images, and how good the results were. It was used to see how the system fared and how the system performed under different conditions. This meant testing the system over and over, changing values in the process to see the change in results.

This involved changing values like the buffer zone, minimal keypoints matches needed, checks through a picture and so forth. These values needed to be set to the optimum for the results to be successful, and the only way to do this was to simply change values and record the numerical results (percentage matches), while also looking at what visual results were returned (images).

It quickly became clear that the system was functioning well, but not to the potential standard it could be. The results found were a little misleading at first, so while the results illustrated some values should be set to something to ensure a decent match, the numerical results didn't always correspond to a correct result. An example being setting keypoints = 75 where it returned a good % match in most situations, it didn't give accurate matches in terms of a visual picture. The performance testing allowed me to make assumptions and estimates in results, but most of all it allowed me to see a trend of where the results would be most accurate.

The performance testing looked like this:

Buffer	limit 50 unless specified	Feature Matching				Histogram Comparison		Results				NB
Amount	Pictures	Min Match	checks	KNN trees	homograph	CompareHist		Time taken	% Match Average		1-10 how accurate?	
2		50	25	50	2	0	0	10 mins	78.00%	25\50		
2		50	40	50	2	0	0	10 mins	82.00%	25\50		
2		50	50	50	2	0	0	9 mins	68.00%	4\50		
2		50	75	50	2	0	0	8 mins	59.00%	1\50		
2		50	100	50	2	0	0	11 mins	NO MATCHES	0\50		
2		50	150	50	2	0	0	10 mins	NO MATCHES	0\50		
2		50	200	50	2	0	0	9 mins	NO MATCHES	0\50		
2		50	250	50	2	0	0	10 mins	NO MATCHES	0\50		It is clear that minmatch must be set to 75 or lower to return good results
2		50	50	100	2	0	0	9 mins	76.00%	23\50		
2		50	50	150	2	0	0	10 mins	81.00%	16\50		
2		50	50	200	2	0	0	9 mins	84.00%	11\50		
2		50	50	250	2	0	0	10 mins	73.00%	8\50		
2		50	50	300	2	0	0	8 mins	71.00%	10\50		
2		50	50	350	2	0	0	9 mins	73%	11\50		
2		50	50	200	2	0	0	9 mins	84.00%	11\50		
2		50	50	200	4	0	0	9 mins	85.00%	31\50		
2		50	50	200	6	0	0	10 mins	86.00%	36\50		
2		50	50	200	8	0	0	9 mins	83.00%	34\50		
2		50	50	200	10	0	0	9 mins	77.00%	37\50		
2		50	50	200	2	0	0	9 mins	84%	11\50		
2		50	50	200	2 cv2.ransac	0	0	9 mins	0\50			ransac doesn't work in loops
2		50	50	200	2 cv2.lmeds	0	0	9 mins	0\50			lmeds doesn't work in loops

Figure 34 Performance Tests Result Screen 1

Therefore, while performance testing in this project is beneficial, it isn't always correct. Computer vision hasn't reached the stage where it can always give an accurate estimate of likeness, and the numerical results compared against the actual pictures prove this statement.

The testing was carried out with multiple examples, to get a better understanding of how the system performed and see how it reacted to a variety of test cases. These examples included the GPO, The Spire on O'Connell Street, Christchurch Cathedral, the Eiffel Tower in Paris, the Taj Mahal in India and a few others. The purpose of choosing different places and famous sites to compare was to get a rounded result, where the system could be tested well.

The results are shown in the appendix (as a snapshot of the spreadsheets), but the full list of results are in the external excel files known by the search name (gpo.xls, tajmahal.xls, eiffeltower.xls).

The results found in these tests made it clear that while the system is a success, the percentage match is not always a correct one, and it doesn't replace the user's ability to distinguish the correct location from an incorrect location. In essence, it adds to the user's decision making process enabling them to make an informed decision.

With average results for one test case, the GPO being around an 80% match, it is clear that the system does indeed return valid results. What it struggles with is determining which within the set of results returned match the query the best.

With results from the GPO, Custom House (all in Dublin), the Eiffel Tower and the Taj Mahal all returning *very* similar results, it was clear how alike the results were from test to test, and that the results were a clear indication of user's expectations and how the system enables them to make a decision, through additional information, and not replacing their existing decision making process.

That does not mean this application fails to meet its requirements. It makes a huge step forward in computer vision and image querying, and while it the numerical results may seem to be inconclusive and even misleading, the application still remains a success, and this cannot be made clearer. Numerical results add to the measurement of satisfaction in the program, but they are by no means a necessity.

Usability Testing

Usability testing was also needed in this system, the system had to be usable and work well from the perspective of someone who just wanted to use the application (and had no previous computing experience). It is important that they find it easy to use and feel as if the application is of use to them. A confusing application that is hard to use would disinterest the majority of users. Therefore, the upload page and display pages must be simple and only display what is needed. Simplicity is key. Not only that, but the application also needed to meet some sort of expectation on what it could do, from the perspective of any user. The application should deliver on their expectations on what results are returned. Testing done to make sure of this should take a huge variety of user tests, and see how happy they are that it's a match. Their satisfaction level is what really counts, along with how usable they feel it is.

Test Cases

For test cases, I used 3 testers to all use the application with their own choices to see how satisfactory it is, with a suitable survey, with the results shown below.

User	How usable?(1-10)	Functionality (1-10)	Expectations(1-10)	Does it return valid results(1-10)
User1	7	6	6	6
User2	8	7	6	6
User3	7	7	6	7
User4	8	7	7	7
User5	7	6	7	7

Figure 35 Test Cases 1

As seen in the picture above, the general consensus among sample users is that the application is a resounding success. It can be seen to be a usable application that interacts with the user well and gives the impression that it is easy to use. It is kept simple allowing the user to input their query, and await the result.

In terms of functionality, the average user is happy with the functionality provided in the system. The system sets out to achieve a certain task and the functionality is designed to ensure this.

The application is rich in functionality and works well to give a meaningful result. The functionality is of an acceptable standard which conforms to the requirements of the user. The expectations of the average user are high too, they expect the system to perform in a certain way and deliver results to a certain standard, and it does just that. The expectations however never reach brilliant heights, as it is clear that the expectations of the average user (and myself) are higher than what is implemented in the current system.

The expectations are for it to return a valid result that can help the user ascertain the location of the photo, and it does that. It does however have some accuracy issues, which are made clear in other sections of this report. It's a commendable project, but has yet to meet the full expectations of the average user.

The results returned can be measured too, in terms of how successful the user feels the system is: is it returning valid results, are they like what they expected. It is believed so. As usability testing is a centred on the user, the tests are created to accommodate and entice the user when using the application. The testing is done with users to prove that the application works well from an ordinary perspective and users can use it with minimum difficulty.

Conclusion

In conclusion the testing has been a success. It has produced evidence that the system performs to an acceptable level and returns a list of results that match a user's expectations. When trying achieve a goal as ambitious as this, there is no "one solution fits all" route to success. The results of these tests could be integrated into the development, which allowed for better code, which led to better test results.

The purpose of testing software is to uncover failures and bugs within the application. Testing comes in many forms like unit testing, performance testing, integration testing, and blackbox testing, white box testing and so on. These forms of testing are used in different situations and scenarios to improve an application. The choice of testing used here was a mixture of usability and performance testing as shown above. This testing allowed for the project to be tested successfully in a complete and well-rounded manner.

There isn't a defined set of testing requirements for this type of application which the project could follow completely. While testing approaches such as performance testing and results testing were indeed somewhat successful, the best way to test such an application is to sit down and use it over and over. Only then could the true potential of the project be unveiled. Naturally, this approach should be accompanied with a well-structured testing strategy and a comprehensive testing approach that could deal with the demands of the project

Evaluation of Project

Introduction

The evaluation for such a project was to ensure that the project's success or failure could be measured and critiqued. The evaluation allows the project to be critically examined, and thereby the progress and completeness of the project can be measured against the starting goal well.

The project is considered to be a huge success, considering how much of the goals it achieves defined in the proposal when this project was started. The success of this project can be seen to be high, while not reaching the full expected heights. Naturally there were problems involved in such a project, and ambition and desire were sometimes diminished in return for some working functionality. Features were scrapped, and the scope was changed. This is not an indication that the project was a failure, but the exact opposite. While it wasn't possible to make the application function as a web application that could interact with multiple users, it allowed me to realise that it could be deployed as a standalone application in the future.

The application's success is based solely on how feasible the goal was, and for that reason is considered to be a success.

Challenges Faced

Naturally, as this is a very ambitious project, there have been many challenges to overcome. The challenges in creating such a project involved many different aspects of the project, as it has a multitude of components. There were numerous challenges in each of the components. These challenges included development challenges, developing environment issues, problems with release versions of source libraries, inconsistency in results and so forth. There were many challenges that were successfully overcome, and as a result, I have been able to apply the knowledge learned from those challenges.

Challenges that have been encountered so far can be split into many different areas.

1. Development problems.
 - a. These are the most standard problems that I've ran into, with regard to developing code, and are also the most common. These problems include problems with returning search results efficiently, downloading images and so forth. One such problem was the development of a solution to find a potential match to the original image, and a corpus of around 30 images is needed to get a decent match. This naturally create a big challenge, with respect to how the 30 images are found.

- b. The development was subject to challenges of design, whereby a concept or piece of logic was quite complicated, and implementing it in the application proved to be quite challenging. Being able to successfully use open source methods too was a challenge that had an effect on the quality of the code written, as sometimes the old code written might not interact well with a library. Challenges arose like: trying to return the correct data type, type formatting, image manipulation etc.
- c. Another issue in the development of this project was that of how the process could be started. It was proposed that the user would upload their image and a call to batch file would be included in the upload script. This was tried in a variety of ways, and none of the options used proved to be successful. The main issue is that running the window in the background failed to run successfully, and if ran with the window visible, Windows would run it as an Interactive Services Detection which is a “mitigation for legacy applications that detects if a service is trying to interact with the desktop.” [21] This became an issue as Interactive Services Detection refused to work with all the Python packages I had installed. Numerous ways had been tried to rectify this problem and make the application run smoothly, but eventually it became clear that the best possible course of action was to run the process manually, once the image was uploaded. This challenge would need to be rectified during any future work, and ties back into why this application would be changed to work as a standalone python application, rather than a web application.
- d. Challenges also arose in how the Flickr query parsed information. The buffer for instance had to be of a certain size if the query was going to be successful. If the buffer was set to less than 1, the application was known to crash. Therefore, the buffer used was always set to a figure above 1.5, to make sure that the application didn't crash. This meant that understanding how the buffer worked was key. Naturally, it created a buffer zone around a point, but how does the number represent a geographical area or size? This was a challenge to solve, as according to the documentation associated with Shapely, the python package that provide the buffer functionality, the buffer was done by writing code like `buff = shapely.buffer(10)`. This according to the documentation that the buffer was set to 10km and testing this proved to be inconclusive as when it the buffer was set to 10, some of the images returned were more than 100km away. This became a challenge, as the documentation clearly stated that the buffer size was in km, unless specified.

2. Developing Environment issues

- a. This included being unable to use certain functionality of external libraries and APIs I was using, and problems involving setting up my environment for developing. A very good example of this was trying to access certain OpenCV methods not currently available in the official

release. To avail of this functionality, OpenCV needed to be updated to 3.0, a beta/testing version of the product. Once I did that, I still ran into problems, and couldn't access those methods. It eventually became clear that while my Python interpreter had detected that OpenCV had been updated, my Eclipse IDE hadn't. Refreshing and resetting some Eclipse preferences sorted this out, which makes it clear that environment challenges in this kind of project are unavoidable, and a solution must be made for each one.

- b. The environment that the project was developed in was subject to external libraries working together. The libraries used in this project included OpenCV, flickrapi, Numpy and ScyPy. Getting these libraries to work together wasn't that hard, but it did raise concern when developing, and not having the correct versions. FlickrAPI is best suited to Python 2.6 and 2.7 so I couldn't use Python3 for the project, while Numpy works best on Python 2.7. This created challenges in knowing what versions of Python packages I should use.

Like any user based application, there are usability challenges too. The system, which will be deployed as a web application needs to be accessible. For instance, some web apps are quite easy to make accessible, as there is limited functionality provided by the web app.

This project falls into that category, it only needs to be accessible from browsers – be it a mobile browser, or a browser like Chrome or Firefox. Each allows for an uploading of data, so accessibility should be easy to provide.

Another challenge is setting proper expectations for the user. Upon visiting the site, users have an expectation of what the application should do. While it may not be possible to fit their needs completely, it is important to take their interests and needs into account when designing the system. Doing so will allow the system to fit user expectations to a higher satisfactory level.

And finally, with respect to the user, the flow of the system should flow well, and remain appealing and interesting to the user. The movement from one screen to the next should be simple, so the user isn't irritated, but at the same time it should also keep the user's interests peaked. They should want to wait for the next screen, but not have to see everything that is happening. The main challenge in this project is about how to find the images quickly and efficiently, using an effective search algorithm or design that minimises processing power and returns a result as quickly as efficiently as it can. The need for such a system led me to believe that this project, if implemented successfully, could be a great project in a relatively new area of image processing which could be very successful. The challenges I've faced in making such a system have highlighted exactly what works in the system, and what doesn't. It allows for clarity in the system with respect to what tasks are going to be relatively straightforward, and what tasks are going to be long winded and complicated, thus allowing me to manage my time efficiently.

The challenges appeared as the project goes on, and even though it is possible to predict what will be the most challenging part to work on, it is not possible at this time to give a full prediction of what challenges will appear to be most demanding, so it is important that each challenge faced needs to be measured against an expectation, only then will it be apparent what components involve the most amount of challenge. The challenges in this project were a testament to the ambition of this project. The challenges of this project made it clear just how complicated this project was and how complex the functionality was.

Changes in Project Plan

Managing those changes is important as the project plan is imperative to the success of the project. Change in any project is expected and there can be major consequences to not managing those changes correctly. This can cause problems such as unwanted changes in the scope of the project, delays in development time, poor quality in the work that is done for the project and even project failure. The problem therefore with change is managing it, not the actual change itself. Technical changes for example, like the scope of some key functionality have to be managed correctly, and the change in scope must not affect how the system performs.

Naturally, with any project specification, there are changes. The project hasn't met the full ambitions of the proposal created when this project was commenced, and this was to be expected. The development of a project like this is fraught with risks, and to be able to manage these risks means that sometimes the ambition and desire for a project needs to be scaled back.

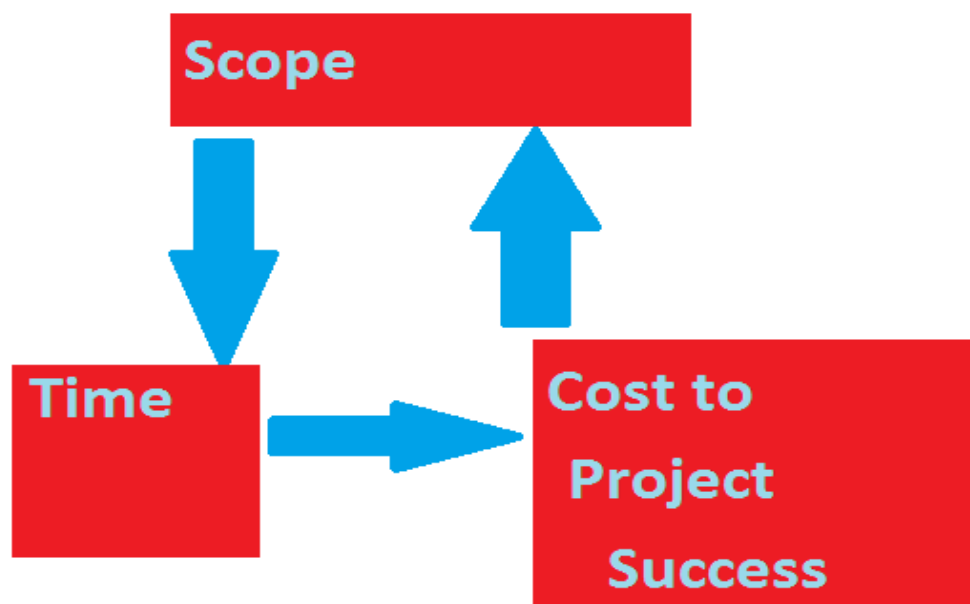


Figure 36 Managing Project Plan Change

The plan, when set out involved an agile approach of developing the two main components (with a huge amount of functionality) first. While the plan followed this approach, the amount of functionality actually provide in the system doesn't need the ambition of the proposal. At first it was planned to create a system that could search by exif, but that quickly became redundant as privacy laws prevent it.

The desire was to create a web application, whereby the system could be hosted on a server and used by multiple users. It became apparent that this wasn't possible, as the system doesn't operate well concurrently (which was hoped when the plan was created that it would). Instead, it became apparent that the system is best suited to a standalone application ran on the client's machine.

These two examples give a clear indication of how the project has changed since its inception, and how the project plan has changed to accommodate this. The diagram below shows the importance of a project plan.

SWOT Analysis

Here, the project is critically evaluated, and analysed to come to a conclusion on how successful it has been. For it be to be considered a success, it should more strengths to it than weaknesses and more opportunities than threats (SWOT stands for Strength, Weaknesses, Opportunities and Threads) and can be carried out on a product(including a piece of software). It allows for the objectives of the project to be critically evaluated, while also considering external factors (like access to a web API, and competition from systems like it).

Within the project, strengths and weaknesses can be seen as internal factors which are controllable, and can be acted upon, such as a piece of functionality. The Opportunities and Threats are external, uncontrollable factors. These form the external environment within which the software performs, like a Flickr server. The major objective of the SWOT analysis is to identify threats to the success of a project or piece of software or business. The Swot analysis allows for the project to be critically evaluated in such a way that any criticism found is constructive, and the project's success is not measured from a biased perspective.

Strengths:

Clearly, the project attempt to meet an ambitious goal. It comes very close to the original hopes of the project and meets the goal of estimating the location for an image. While it may not be as effective as first desired, it still reaches a commendable height in terms of functionality. The project works well to find suitable images, and successfully manages to compare them. While it may not always match photos correctly, it still has a decent success rate, as the underlying software does a good job at what it was created for. The strengths of this system are:

- Efficient
- High amount of functionality
- Decent success ratio
- Meets a lot of user expectations

Weaknesses:

There was always going to be risks in this project, and in turn those risks could create weaknesses. For instance, there was a risk that finding suitable images wouldn't be an easy task, and it wasn't. It created a weakness whereby finding suitable images depends on suitable search tags and an estimated location (city and country), the images returned from this search aren't always completely accurate. This has a negative effect on the end result, as finding similar images to compare is a key part of the process. The weaknesses of this project are:

- Images returned by the search aren't always completely accurate.
- The process from start to end is long process, and can lead to complications if it lasts a while (server time outs from server and so forth).
- The comparison process is depended on previous results, which leads to complications if the process doesn't work. If one part of the process fails to work, or doesn't return a decent amount of results it can lead to complications throughout the process.

Opportunities

With a project like this, being able to capitalise on the opportunities was always important. The aim of this project was to provide a system that could estimate the location of a photo. Ultimately it was designed to act as tool in the decision making process of a user. It aids in their decision process, it is not meant to replace it. Therefore it was obvious that there were many opportunities to this project.

Opportunities:

- Aid in the decision making process for someone searching for a location, giving it the opportunity to meets a user's wish to find out information.
- A chance to provide a service which could genuinely be seen as an advancement in computer vision and image recognition.

Threats:

Threats to the success of this project included:

- Reliance on an external provider, Flickr.
- Limited amount of queries can be made (due to Flickr limit per hour).
- Images found are subject to term. Other searches are more successful than others, and this is due to the external factor of how images are uploaded and described on Flickr

Conclusion

In conclusion, evaluating the system enabled it to be critically analysed in such a way that the success of the project could be measured accurately. It gave a clear conclusion on how successful the project was. A SWOT analysis gives a clear indication of how successful a project is, and this analysis does just that, as it measures the success of the project in a clear, structured way that can be easily understood. It has been successful in estimating the location of an image, and aids in the user's decision making process to establish the location of the photo.

Success of this project is measured not only by accuracy of the system, but also by the satisfaction level of those who use it. Combined they give a clear picture on how the system meets those needs, and is successful as a result.

Conclusion

Introduction

This chapter is the concluding chapter in this report and reviews the key learning points of the project, suggestions of future work to improve this system and includes a final conclusion. The conclusion is meant to ascertain whether the project was a success, future work that could be carried out, and what was learned during the course of this project

Main Points Learned

The project involved a huge amount of functionality, in several different areas of interest. I feel the experience over the course of the project has given me a great opportunity to learn and grow. It included areas such as:

- Interpersonal and Technical skills
- Development skills
- GIS
- Image Processing
- API programming.

The interpersonal skills learned during the time spent on this project are commendable, as they were imperative to the success of the project. Skills such as planning, self-motivation and time management were improved massively, in way

previously thought impossible. These skills allowed me to progress at a satisfactory rate which meant the project was finished in an acceptable time scale.

Development skills in this project were a highlight of how I learned new skills and improved existing ones. My understanding of the technical development involved in making a project has improved, and my knowledge of development styles and coding techniques in Python has improved massively. My technical skills and personal skills, such as time management have been honed the project has also increased the confidence in my abilities as a developer. The time spent on this project has also enabled me to see a new perspective on developing and creating a viable application. Sometimes spending large amounts of time working on a feature can be detrimental to my productivity during the day. Taking some time off to reflect and get a breather can allow for a solution to be met. Constantly looking at a screen to find a solution can be very frustrating, and I learned how to manage that.

GIS and API programming were also major parts of this project, ranging from using Python geospatial libraries like Shapely to working with web APIs like Flickr. My knowledge of image processing has also improved. I've learned to work with functionality like feature detection, histogram comparisons and general computer vision techniques. Overall I have learned a lot of technical knowledge, but have also learned some valuable non-technical skills along the way.

Future Work

The future work of this project involves making improvements to the functionality and effectiveness of the application. If more work were to be put into the project (and I'm almost certain it will), the main objective for any future work would be to enable the search to be carried out on multiple APIs. Flickr has access to 6 million geotagged images as specified earlier, but there are a multitude of other APIs that also contain images with geographical information. Utilising these APIs would bring about a higher success ratio of the application.

The functionality of this project would also be improved in terms of how the search for images is carried out, and how the image comparison finds decent matches. It would follow an incremental approach to make sure that new functionality could be added with each new update.

There would also be a consideration to look at several other areas of interest that need work.

- Redesign the client page to work as a Python GUI (as it's not feasible to make the system into a functioning web app).
- Improve the search accuracy of the system
- Increase the success rate of the comparison process.
- Improve how quickly results are returned to the user.

Any future work carried out this project would be done so with a priority to improve how the user interact with the system. While there is a significant amount of rich functionality, the interaction between a user and system could be vastly improved. As described above, the main work would be to redesign the client side of the application into a standalone Python application. A web page simply does not allow the system to be used to its full potential, which is obviously a concern. When this project was critically evaluated, it was clear that for it to be considered a complete success, the client page must interact flawlessly with the application. A Python GUI would remove the need for a web server, and would allow the application to work more simply, with the application relying on simple screens in Python, rather than the complicated process of uploading an image, starting the process and waiting for a web page to load the results.

The accuracy of the search process without a doubt is in need of improvement. It finds results very well, but requires the use of search tags and an estimated location to be successful. In the future it would need to be improved if the success rate were going to be higher. To do this, a lot of research must be done. It's not a main priority, but definitely something worth considering in the future.

Another part of the application that could be improved upon is the success rate of the comparison process. Currently, it works well and delivers an acceptable match and result to the user, but undoubtedly the process will always have room for improvement. This could include how quickly results are returned, what results the process returns and even how effective it is at returning valid results could be improved. Basically, while it works well and returns valid results, there is always room for improvement.

Future work for this project has to be beneficial to the user. Any work carried out on the system must benefit the user, and not remove current functionality from use. Doing so would be ill-advised as being able to manage future updates involve keeping the user interested. And finally, there is the case of improving the search query. While it has a high success ratio and returns a decent corpus of images to work with, it still could be improved upon. It could be improved to be more efficient and quick, it could be improved on how accurate the results are and so forth. In summary, future work would be done on this project to improve existing functionality and create new functionality. There would never be a need to scrap current functionality as it is currently very successful.

Ideally, if the application were to be successfully made into a Python GUI system, it would need a simple design, where the image could be uploaded into a folder, and the process started by the start button.

1. Start Screen

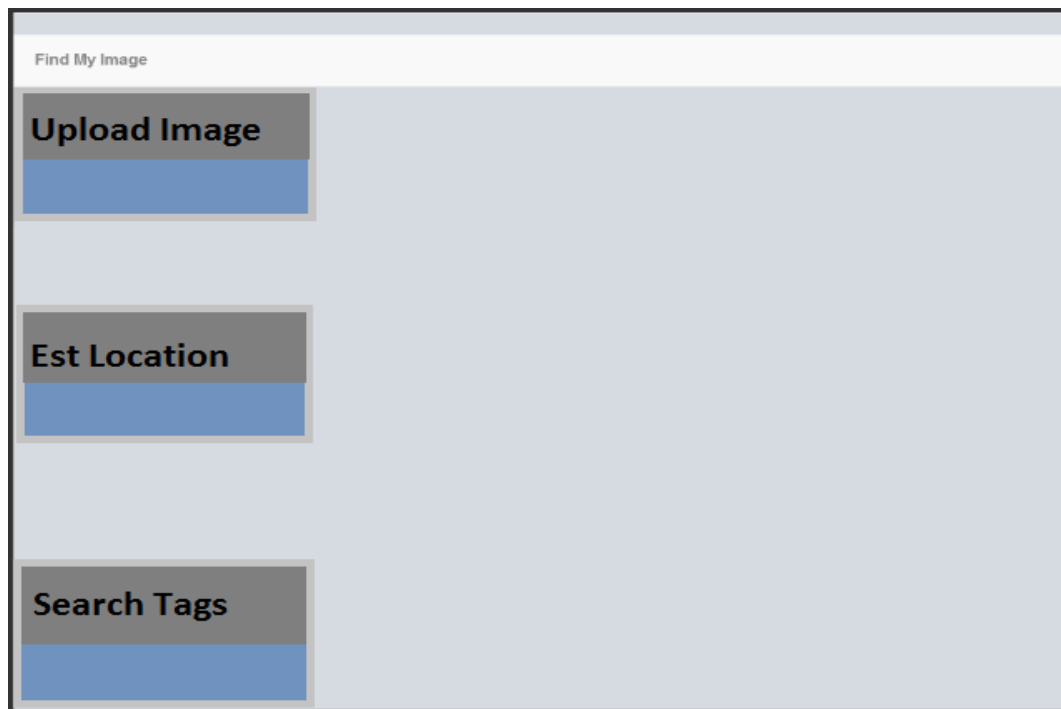


Figure 37 Gui Start Screen

2. Progress Screen.



Figure 38 Gui Progress Screen

3. And an ending screen like:

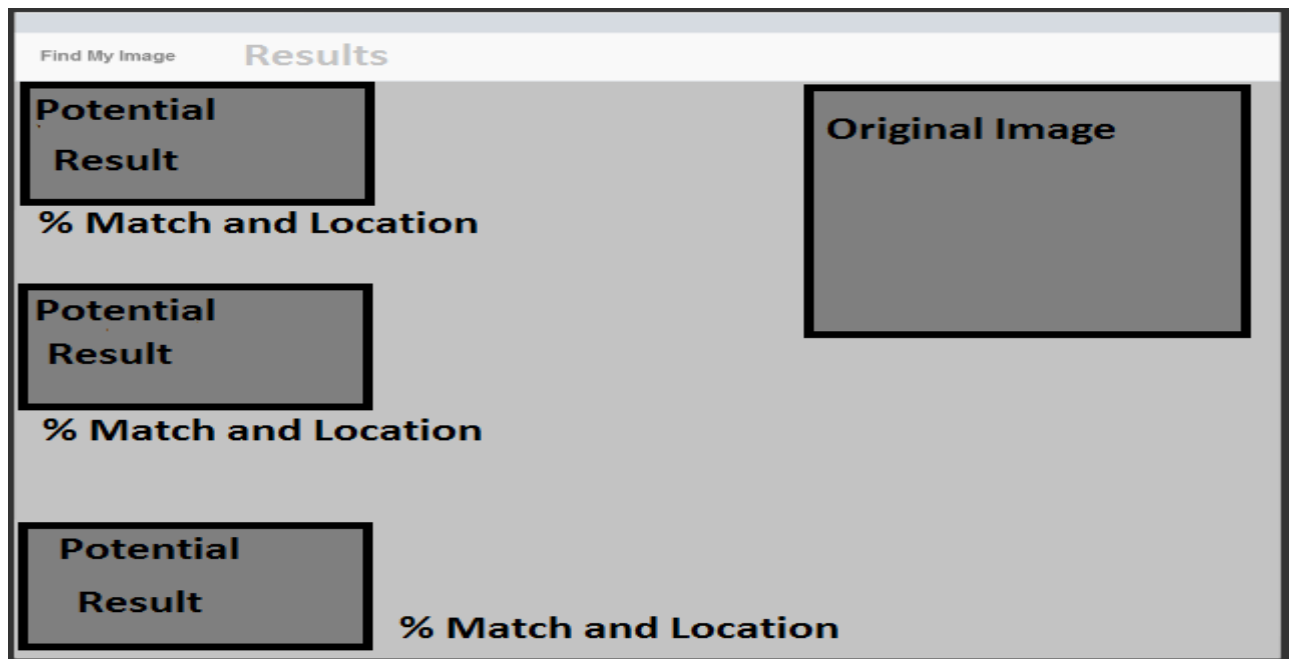


Figure 39 Gui Results Screen

Final Words

While this project does not meet the full requirements and ambition set out at the start of this project, it is clear that this project is still a piece of commendable work. This is because there was a huge amount of work put into making this project a success. And while it may not meet the ambitions that were first set, it still reaches a very high level of success in what it has achieved. The experience gained and lessons learned from this project have been absolutely incredible, and the process of creating this project has been a thought provoking and enjoyable project. The project has been an incredible learning experience for me and it has allowed me to improve my programming ability far beyond anything I could imagine. It has also allowed me to see that creating a project on my own is indeed possible. Having a supervisor as helpful and knowledgeable as mine (Mark Foley, DIT) has allowed me to see new insights into every situation in this project.

This project clearly illustrates the skills I have gained during my time as an undergraduate, and I am proud in the work I have achieved during this project.



Figure 40 Results Map

Appendix A– Results

This appendix contains valid results and figures in the project, such as results tables and further facts and figures.

The results files are included in the folder with this report.

<u>Result Set</u>	<u>Average %</u>	<u>Comments</u>	<u>1-10(how accurate)</u>
GPO(Excel file)	71%	Potential of great matches	7
Custom House(excel file)	70%	High potential of good matches	6
Taj Mahal(excel file)	73%	Lots of good	6

		matches found	
Eiffel Tower(excel file) (The Eiffel tower, being so well known, returns a very high amount of potential matches)	78%	Excellent corpus of matches found.	8

User Testing	Satisfaction	Comments
User 1	6.25\10	Great application, very usable
User 2	6.75\10	Nice application.
User 3	6.75\10	Very exciting and ambitious project, well done
User 4	7.25\10	Good project
User 5	6.75\10	Great piece of software with good results
User 6	6.75\10	Interesting application, great potential
User 7	6.5\10	Lovely to work with, questionable use of UI

Appendix B- Summary of Files Submitted

<u>Name of File</u>	<u>Purpose</u>
Controller.py	Starts the system, by calling the query file
Query.py	Searches for similar images done here
Display.py	Moves found images to folder web server can see.
Featurematching.py	Runs a feature matching process on images found in search
Histogramcomparison.py	Compares the histograms of suitable images found
MyScript.bat	Starts the Controller from the command line.
Original/ folder	Where the original photo is stored
Compare/ folder	Where the results found in search are stored
Compare2/ folder	Results from feature matching stored here
Compare3/ folder	Results from histogram comparison stored here.
Upload.html	Client html page to upload image
Upload.php	PHP script to upload image
Display.php	Displays images found to user on html page

References

- [1] J. Hayes and A. Efros, "IM2GPS: estimating geographic information from a single image," [Online]. Available: <http://graphics.cs.cmu.edu/projects/im2gps/>. [Accessed November 2013].
- [2] Great British Greyhound Walk, "Great British Greyhound Walk," Great British Greyhound Walk charity, March 2014. [Online]. Available: http://greatbritishgreyhoundwalk.org.uk/wp-content/uploads/2014/03/globe.world_.jpg. [Accessed 20 March 2014].
- [3] C. S. C. P. ., B. Henrik Kretzschmar, "Estimating Landmark Locations from Geo-Referenced Photographs," March 2008. [Online]. Available: <http://www.informatik.uni-freiburg.de/~stachnis/pdf/kretzschmar08iros.pdf>. [Accessed 26 March 2014].
- [4] D. Coulter, "PHPFlickr," 2013. [Online]. Available: <http://phpflickr.com/phpFlickr/README.txt>. [Accessed November 2013].
- [5] S. A. Stüvel., "http://stuvel.eu/," Stuvel, March 2008. [Online]. Available: <http://stuvel.eu/flickrapi>. [Accessed 2 January 2014].
- [6] Appcelerator, Inc, "http://pydev.org/," PyDev - AppCelerator, 2008. [Online]. [Accessed 4 February 2014].
- [7] OpenCV, "OpenCV Docs," OpenCV Open Source Developer Community, January 2014. [Online]. Available: <http://docs.opencv.org/>. [Accessed 3 February 2014].
- [8] M. Lutz, Programming Python, Sebastopol, California : O'Reilly, 2011.
- [9] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints paper," University of British Columbia, January 2004. [Online]. Available: <http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>. [Accessed November 2013].
- [10] J. E. Solem, Programming Computer Vision with Python, O'Reilly Media, 2012.
- [11] F. J. Hudson and A. Behforooz, Software Engineering Fundamentals, Towson, Baltimore: Oxford University Press, 1996.
- [12] Computer Training Centers, "Computer Training Centers," Computer Training Centers, October 2012. [Online]. Available: <http://computertrainingcenters.com/wp-content/uploads/2012/10/what-is-agile-development.jpg>. [Accessed 7 March 2014].
- [13] FluidUI, "FluidUI," 2012. [Online]. Available: <https://www.fluidui.com>. [Accessed 2 March 2014].
- [14] "OpenCV," OpenCV(Open Source Developers Community), 2013. [Online]. Available: <http://opencv.org/>. [Accessed December 2013].
- [15] D. D. E. F. M. D. D. T. John Hunter, "Matplotlib," October 2013. [Online]. Available: <http://matplotlib.org/>. [Accessed 23 March 2014].

- [16 Docutils, “Docutils,” 22 July 2013. [Online]. Available:
] <http://docutils.sourceforge.net/>. [Accessed 4 March 2014].
- [17 SciPy, “<http://www.numpy.org/>,” SciPy, January 2013. [Online]. Available:
] www.numpy.org. [Accessed 2 March 2014].
- [18 C. Howitz, “Simcrest blog,” Sim, 1 June 2012. [Online]. Available:
] <http://blog.simcrest.com/what-is-3-tier-architecture-and-why-do-you-need-it/>.
[Accessed December 2013].
- [19 M. G. Anthony Eden, “FlickrJ: Java API which wraps the REST-based Flickr
] API,” Flickr, July 2009. [Online]. Available: <http://flickrj.sourceforge.net/>.
[Accessed 20 January 2014].
- [20 I. Heywood, S. Cornelius and S. Carver, An Introduction to Geographical
] Information Systems, Essex, United Kingdom: Prentice Hall, 2006.
- [21 P. Altimore, “Pat’s Windows Development Blog,” Microsoft, 27 April 2010.
] [Online]. Available: <http://blogs.msdn.com/b/patricka/archive/2010/04/27/what-is-interactive-services-detection-and-why-is-it-blinking-at-me.aspx>. [Accessed
20 March 2014].
- [22 N. I. o. Health, “ImageJ,” 2004. [Online]. Available:
] <http://rsb.info.nih.gov/ij/index.html>. [Accessed 2013 December 10].
- [23 F. Graf, “JFeature Library, Project Page,” [Online]. Available:
] <https://code.google.com/p/jfeaturelib/>. [Accessed 11 December 2013].
- [24 “AngularJS,” Google, 2013. [Online]. Available: <http://angularjs.org/>. [Accessed
] November 2013].
- [25 S. Ellis, “HEA Academy Project Planning,” 17 September 2012. [Online].
] Available:
[http://www.heacademy.ac.uk/assets/documents/funding/tdg/Project_planning_te](http://www.heacademy.ac.uk/assets/documents/funding/tdg/Project_planning_template.pdf)
[mplate.pdf](http://www.heacademy.ac.uk/assets/documents/funding/tdg/Project_planning_template.pdf). [Accessed November 2013].
- [26 N. David, “SitePoint,” 8 January 2013. [Online]. Available:
] <http://www.sitepoint.com/11-reasons-to-use-twitter-bootstrap/>. [Accessed
November 2013].
- [27 PHP, “PHP,” 2013. [Online]. Available: <http://php.net/>. [Accessed November
] 2013].
- [28 “Python Official Website,” Python, 2013. [Online]. Available:
] <http://www.python.org/>. [Accessed November 2013].
- [29 “Python Comparisons to Other Languages,” Python, 1997. [Online]. Available:
] <http://www.python.org/doc/essays/comparisons.html>. [Accessed November
2013].
- [30 “Turnkey Linux LampStack,” Turnkey Linux, November 2013. [Online].
] Available: <http://www.turnkeylinux.org/lampstack>. [Accessed November 2013].
- [31 J. F. Canny, “A variational approach to edge detection.,” *Association for the
] advancement of Artificial Intelligence*, Vols. AAAI-83, 1983.