Landmark-based Camera Location Approximation

Wiktor Aleksander Kaczor

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Cybersecurity and Forensics

School of Computing

October 2020

Authorship Declaration

I, Wiktor Aleksander Kaczor, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Wiktor Aleksander Kaczor

Date: 30/04/2021

Matriculation no: **40428623**

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

Wiktor Aleksander Kaczor

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Current image localisation approaches rely on the satellite-based Global Positioning System (GPS) with the receivers embedded within nearly every modern image-capture device but the design constraints for such portable devices and the signal reflection in urban areas leads to relatively low accuracy ratings. The purpose of this dissertation is to explore the use of Structure-From-Motion techniques for an alternative extensible image localisation solution within urban environments featuring popular landmarks by leveraging a small subset of accurate GPS tagged images, and the availability of untagged high-resolution imagery. Applications for such a system range from law enforcement, tracking members of known criminal groups to a more civilian application of automatically geotagging social media pictures. The proposed method is tested against the higher accuracy assisted GPS tagged images, experimental results show usable results, just under conventional GPS accuracy, possibly better with more sophisticated methods than were employed in this project.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr Sean McKeown, for his guidance and support throughout the course of the project, without who, this dissertation if not entire project would have likely would have fallen flat on its face.

I would also like to acknowledge Dr Petra Leimich, the second marker for this project, for their contribution of providing valuable feedback that helped steer the later direction of my work.

Finally, I would like to thank Google for their endless academic support and the Flying Spaghetti Monster for its moral guidance.

1 Introduction

Photogrammetry is the process of extracting useful information from images, namely information about physical objects contained within the images, used in conjunction with multiple images featuring the same object, one can determine the dimensions of an object with a great deal of accuracy. Combined with a technique called Structure-From-Motion (SfM) allowed the reconstruction of accurate 3D models of the images' subjects. Despite that and the increasing number of papers published on the subject, photogrammetry did not gain widespread adoption until the late 2000s, partly due to the required computational work and partly due to the expensive equipment required to obtain good digital imagery.

Many years have passed since then, the expansion of computing necessitated ever larger amounts of cheaper computing power, the advent of the internet came with many image hosting websites, in fact, even in 2008, a search for the Notre Dame Cathedral returned over 15,000 images. Photogrammetry did not remain stagnant either, the techniques' capabilities of modelling the world and their improving efficiency have led to photogrammetry being available to the unspecialised masses, and with it many ambitious projects, like that of Google Earth 3D buildings, and the easier creation of digital asset libraries for the entertainment industry. Additionally, the use of satellites, a project called the Global Positioning System (GPS) have enabled one to localise devices carrying the necessary functionality anywhere in the world.

There are many reasons one might want to produce a high-quality 3D map of an environment, some examples might include automatic geotagging of social media images, robot navigation within a human-inaccessible environment and tracking the movement of a head mounted display for better virtual reality. Other, more cybersecurity-oriented reasons might be movement pattern tracking through following someone's social media profile images, tracking the wealth of households by what make of car they have outside or within what hotel rooms their images happen to be localised. These applications can even be applied to develop better law enforcement techniques, for example, getting the location of crime suspects through their latest image social status update containing something as simple as a rather obstructed yet still statistically unique building. Yet another hypothetical extension could even be used

to cross-reference the places frequented by suspected members of criminal groups, the possibilities are simply immeasurable.

Despite the improvement in mapping the world, and localisation capabilities for digital devices, adoption of localisation for the images taken by these devices is still proving incomplete, and if available, their accuracy being questionable. Earlier approaches have tried to tackle this problem by matching images to a database of other images containing GPS information and estimating the position between using at least two images as a guideline, leading to large computational requirements for the search as well as the need of a database containing accurate GPS imagery (Zhang & Kosecka, 2006).

The objective of this dissertation is to determine whether one can combine the increasing amount of publicly available data, cheap computing power, ideas of georeferenced 3D models from Google and new photogrammetric techniques to build a robust, extensible, and cheap solution for the localisation of arbitrary images containing recognisable structures, usually focusing on popular landmarks. This includes the collection of a usable photogrammetric dataset from crowdsourced, unstructured image hosting websites, aligning the reconstruction to the GPS datum and extracting the position of novel images from that reconstruction. Additionally, adding the capability to extend the reconstruction, using the additional localised imagery and by merging different reconstructions together on the same GPS datum.

Secondary objectives are to determine the accuracy and effectiveness of this method, how many images, both normal and ones used for georeferencing are required for satisfying results, that is the alignment of the model into its true GPS coordinate system, if the 3D reconstructions can be merged, how that would affect the localisation accuracy as the size of the model increases and how the recovered positions from the localisation can be used to refine, and possibly extend, the reconstructed 3D model.

In doing so, we derive the following research question; "***Is it possible for higher accuracy GPS image locations to be obtained from a georeferenced 3D reconstruction than by using on-board mobile phone GPS chips?***".

The benchmarks for this endeavour are the on-board phone GPS chips due to their widespread adoption in mobile devices, and the accuracy for said chips within urban

environments being around the neighbourhood of 20m. This due to the satellite signals required for triangulation being reflected by the features found in such environments (Merry & Bettinger, 2019). On the other hand, assisted GPS, the use of techniques such as cellular tower triangulation, additional algorithmic improvements in compensating for Doppler shift and higher accuracy GPS receivers yield order of magnitude higher accuracies, around the centimetres to one meter range (Pesyna, K.M., Jr, Heath, & Humphreys, 2014). The evaluation of the project will thus be in comparison to assisted GPS tagged images.

The work is valuable to a number of disciplines, the main beneficiaries include but are not limited to; crime forensics, with emphasis on matching images to the location of the crime, massively increased quality of city maps due to being able to easily localise crowdsourced images, value in the entertainment industry by including the generated 3D models in digital asset libraries, possibly for video games depicting historical or contemporary locations, and robotics, in helping an autonomous agent navigate through a city. Additionally, the geomorphological community can also benefit from this work, as they did from the initial advent of photogrammetry, for tracking the movement of buildings instead of landmasses (Micheletti, Chandler, & Lane, 2015).

In the following section, there will be some information for understanding the processes described above, following that, the objectives of this project will be covered. Next, the methodology section will be used to describe the strategies undertaken to achieve these objectives. Finally, the conclusion, a section detailing an overview of the answer to the research question, a reflection on the conducted research, where the project fell short due to out-of-scope improvements that could have been made and recommendations for future work to be undertaken.

2 Structure of the dissertation

The dissertation is organised as follows:

- The **Introduction** provides a general overview of the scope of the problem, including the concepts of photogrammetry, with references to later sections where things are explained in more detail as a guideline to follow the work. It also provides the main scheme; dataset gathering, feature detection, feature matching with outlier management, reconstruction, model geographical alignment and extraction of information from the model.
- The **Structure of this Dissertation**, to repeat, outlines the structure of this dissertation, that is how each section will be organised and what will be contained within it.
- The **Background** will serve as a literature review and cover a relatively in-depth overview of the near or state-of-the-art of photogrammetry techniques for terrestrial close-range images as well as any specific techniques included in the project. This literature review will also present previous attempts at image-based localisation and automated image dataset gathering.
- The **Methodology** will be used to cover the scope and limitations, essentially, what the project is limited to because of any budget, computational or time constraints and how this might affect the project process and outputs. Then, it will continue with describing the work done for the completion of this project, justifying the choices, any experiments for those justifications, and will be laid out in a format like the introduction, that is, following; dataset gathering, feature detection, feature matching with outlier management, reconstruction, model geographical alignment and extraction of information from the model.
- The **Results** section will be discussing how the project does in satisfying the aims and objectives, and any additional limitations encountered, and provide the results of any small experiments used to justify the choices for the direction of the project. It will also use the ground truth dataset to show the accuracy and precision of the inferred camera positions.
- The **Conclusions** section will provide a concise list of what was achieved, expand on any possible improvements in future work and highlight the limitations of the current work.

- The **References** section will list all documents directly cited in the work, including paraphrasing.
- The **Appendices** section will include all additional material required for submission.

3 Background

3.1 Overview

This section will largely be used to explore the current literature on the 3D reconstruction pipeline and will be relevant to understanding the work described in the methodology section. The work will be covered in layman's terms and based on the following sections:

- First, retrieval of an image dataset for 3D reconstruction, focused on large datasets like image-sharing websites, ensuring the images feature the same subject and have sufficient details for later stages.
- Next, feature extraction which will cover what features are in relation to the computer vision field, the properties of features and why they are used for 3D reconstruction.
- Then, the reconstruction processes, an outline of the different approaches, and the steps involved in deriving object geometry from images.

3.2 Dataset Gathering

3.2.1 Dataset qualities

The photogrammetric dataset, composed of still images and optionally, videos, contributes most of the reconstruction quality despite the ever increasingly complex and efficient techniques of deriving geometry from images. This is because the time taken to process a dataset increases with the amount of data, further so with trying to process dataset outliers and further decreases or increases with its quality and type, for example, if the dataset only contains GPS labelled images, or even video frames that only match to a certain number of images going forwards or backwards, then the process can have a more constrained solution space. This is due to the algorithms involved taking computationally longer and more expensive with unknown variables, like camera focal length, to be estimated and continually refined, or each added image to be compared, in the worst-case scenario, against all other images in the dataset.

Additionally, the resolution, content and "blurriness" of an image directly impacts the number of visual features that can be detected (discussed later). Another significant factor would be the elimination of outliers, the GPS data contained within an image can

be faulty which would add massive error to the georeferencing of the model, this means a good quality dataset is paramount to the success and quality of the project.

There are a few ways to achieve such a dataset:

- Filtering out blurry images using Laplacian variance. This is due to a highly focused image having much higher Laplacian variance, thus better-defined edges, it is additionally good for filtering out images where many features would not be detected later in the SfM pipeline since things like the sky do not feature much variance, and an image almost entirely containing the sky would not pass (Rosebrock, 2015).
- The Fast Fourier Transform (FFT) function for finding improperly coloured images, like those that a drawing would be compared to a photograph, this is due to the sharp lines and colour changes being much higher “energy” and “frequency” to such a function while photographs would have more gradual changes and be comprised of smooth gradients. (Kanwal, Girdhar, Kaur, & Bhullar, 2019)
- ELA (Error-Level Analysis) for detecting digital image manipulation in lossy compression image storage techniques like JPEGs, since an alteration, like adding in a structure, would be lit up under such a comparison, this is due to the JPEG being recompressed, the older (already compressed) image would once again be compressed, resulting in more artifacts compared to the newer digitally added feature (Paganini, 2013).
- One last technique to do image manipulation detection would be to train a neural network, a convolutional neural network, which would derive a much higher dimensional function for doing just that, with accuracy of over 90%, additionally, trained model weights can also be acquired if one does not have the resources for training. That said, neural networks are outside the scope of the project and will not be used nor will they be discussed in detail. (Kim & Lee, 2017)
- Extracting image features, for example, corners, and matching against other images’ features, the matching features would be used to estimate a transformation, like overlaying images on top of each other where they match, to see if the new image features’ estimated geometry support the original image spatial geometry, basically, if all the extracted features that are supposed to be in the same space, overlay, essentially, a geometric sanity check. This also

filters out outliers in the form of moving objects such as pedestrians, cars, airplanes etc. since they would not support the model. (Johannes Lutz Schönberger, True Price, Torsten Sattler, & Pollefeys, 2016)

3.2.2 CBIR (Content-Based Image Retrieval)

Content-Based Image Retrieval is a way of retrieving images from a large database based on their semantics, for example, both images contain a red apple, by using query images and matching candidates. It can also be used for photogrammetry, the semantic metric being along the lines of both images featuring a specific building, or facade of the building. This means that it can be used in conjunction with the internet and image sharing websites such as Flickr and Instagram to semi-automatically build a dense dataset of images containing a landmark or fill in missed sections if one devised such a method in conjunction with Structure-From-Motion techniques (explained later). This is also relevant because contemporary photogrammetry techniques benefit from low outlier ratios and those techniques eliminate them in the dataset gathering phase.

There are four main ways to retrieve an image based on its content:

- Colour histogram, a representation of the distribution of colours within an image. Similarly distributed images would be matched, unfortunately, this is not really a good indicator since it is highly unadaptable to something such as a viewpoint shift or lighting change, for example, day versus night, and it would be rather likely to match a green apple to simple grass. (Rishav Chakravarti & Xiannong Meng, 2009)
- Feature-based, relying on feature detection as explained below, distinct visual features are extracted from an image and matched against a possibly large database of features for your content instance, for example, a certain building. The performance and accuracy are highly dependent on the feature detection and matching algorithm, also explained below. This can also be based on a bag of visual words approach for computational efficiency, the size of said bag will affect both the performance and accuracy of this method wildly. (Afshan Latif, Aqsa Rasheed, Umer Sajid, & Tehmina Khalil, 2019)
- Deep learning based, a trained approximation of a function to find matching images, to a certain object instance. Accuracy will vary wildly from the type of

neural network model used to the data, and random error is also very possible if unlikely. This is the current state-of-the-art CBIR method due to its accuracy, some models reaching well over 90%. (Dubey, Shiv Ram, 2012)

- Hybrid (Feature fusion), this method relies on a mix of the previous methods, the predictions for whether it is the same object in both images are weighted for the selected ways, and the final prediction is based on that.

3.3 Feature Detection and Matching

Feature detection and matching is a fundamental problem in reconstruction from 2D images, which requires a way to match pixel points of one image to corresponding points in another image, despite any distortions, slight viewpoint or lighting changes. No perfect solution currently exists and the internal working of any feature detector, descriptor and matching mechanisms are beyond the scope of the project. This section will instead cover the necessary components of feature detectors for SfM applications as described by (Lowe, 2004).

3.3.1 Scale Invariance

This means that no-matter how much you zoom in or out of an image, the feature descriptor remains the same. This is useful in photogrammetric applications because images taken closer to and further from the subject would otherwise have their detected features described as radically different, instead, the features are able to be matched to their higher or lower scale counterparts.

3.3.2 Rotation Invariance

This means that a feature rotated by some degree is still recognised as the same. An example might be upside down images, a feature detected in an upright image will still match with the same feature in an upside-down image of the same subject, or a less extreme version would be a slightly rotated one.

3.3.3 Translation Invariance

This means that a feature can be slid along any axis and it will still be recognised as itself. To provide an example, the camera can be slid on a straight rail in front of a building facade and the feature will be recognised even when it moves along the images.

3.3.4 Lighting and Illumination Tolerance

This means that detected features will still be matched under different lighting and contrast conditions. Images taken on a cloudy and sunny day should still match, as should they also match in night-time conditions under ideal circumstances.

3.3.5 Illustration

The above can be illustrated using this graphic by (Krause, 2016):

Figure 1 Visual feature invariances

3.4 Reconstruction

3.4.1 Photogrammetry

For the purposes of this project, photogrammetry is defined as a method of measuring physical objects using positive photographs or images and other energy patterns such as LiDAR (a way of detecting how far away an object is from a laser) or depth-sensing cameras. There are various types and generations of photogrammetry, but this dissertation will primarily be focusing on relatively close-range terrestrial photogrammetry, that is, photogrammetry of images usually taken from a position on the ground and within roughly 300 metres of the target like an internet-based dataset would provide.

Contemporary traditional photogrammetric techniques are not well suited to unordered, uncalibrated multiple camera internet sourced images, this is due to the algorithms implemented to do said reconstruction (discussed later) and this affects the computational time and the final reconstruction quality. This problem arises from the unknown camera intrinsic and extrinsic parameters, intrinsic being things like focal length, optical centre, radial and tangential distortions, with the extrinsic being the camera pose, i.e., where it is located and pointing including rotation pitch yaw and roll. This is because the algorithms usually rely on good known intrinsic and extrinsic parameters estimation to reduce their search space for a good solution, i.e., 3D reconstruction.

The major computational time consumption is the feature detection and matching, features are descriptions for certain parts of an image that would be recognised for being the same feature in another differently posed image of the same subject when matched, an oversimplified example in photogrammetry might be corners, and their orientation. The detection of the features takes some time depending on the algorithm, with types existing for real-time applications and post-gathering processing. There are a few features in said feature detectors that must be considered; scale, rotation and translation invariance as well as tolerance to lighting and distortions like shear mapping so that the given feature detector can recognise the same feature under a variety of conditions.

The other, more computationally expensive problem is matching features between images, which can take quite a lot of time depending on the number of features detected, to produce possible image pairs. These are then spatially verified using an algorithm such as RANSAC (Random Sample Consensus) or GHT (Generalised Hough Transform), spatial verification is the process of determining a spatial correlation, i.e., determining it is one landmark in both images, instead of another landmark with which the first one may share some features. This can be done, as an example using RANSAC, by choosing a random sample of observations from the entire set of matches, computing a transformation, for example, affine projection, and seeing if this produces the highest number of matches and thus the minimal error, with a predetermined number of allowed iterations, this should eliminate the outlier image matches and the feature matches for such verified images can be accepted, RANSAC has been known to work satisfyingly with datasets comprised of over 50% outliers (Hast, Nysjö, & Marchetti, 2013).

Another facet of feature matching is that the features for highly repetitive structures might be discarded, this is due to visual descriptors for a fence being easy enough to match anywhere along its line, which eventually shows up as a large amount of error for that structure, making its reconstruction difficult. This is alleviated by using epipolar geometry algorithms, those estimate the camera extrinsic using tie points and check that the matches are along the same epipolar line, doing so leaves the correct matches for the repetitive structures intact because while the descriptor may not change much, the epipolar geometry does, this is also called “guided matching”.

When the feature detection and matching are both complete, the reconstruction begins, and the camera positions, intrinsic and extrinsic are both estimated and refined using a process called bundle adjustment and a type of SfM (Structure-From-Motion), there are three main ones; incremental, where camera poses are calculated one by one and added to the collection to form the final produce, global, where all camera poses are calculated at the same time, and finally, out-of-core, where several sub models are made and then merged together into the final cohesive model.

SfM also uses the bundle adjustment algorithm which does the following; as more images are added and registered in the reconstruction, the point coordinates for a feature are refined alongside the camera intrinsic and extrinsic, outlier points, i.e.

points which are not seen from a predetermined number of images or points which are determined to be erroneous due to having their location be unstable between a range of images, are removed to reduce the “messiness” of the final model. This means that photogrammetry techniques can also be used to find the camera parameters if some constraints are applied.

Additionally, the entire process can be made easier by using video, since the search space for good matches could be restricted to only a couple neighbour frames instead of the entire dataset, or to match video frames to a reconstruction afterwards and run it again with the recovered positions for more reconstruction detail. There are more approaches that can be taken to narrow the space to choose image pairs, examples might include camera GPS coordinates, Ground Control Points (GCPs), i.e. a set of known 3D points identified in an image. Finally, possible image pairs can be specified manually, this means they can even be inputs from a deep learning solution, image hashing or any other algorithm.

3.4.2 Scale-Invariant Feature Transform (SIFT)

SIFT is a feature detection and description algorithm, it detects feature points for images, where the points’ descriptors in an image are likely to be present in the same relative place regardless of how the image is rotated, moved, illuminated and/or scaled. Then, it extracts the feature descriptors for those points, a 128-dimensional vector descriptor, although how that happens is beyond the scope of this project. It is the current state-of-the-art, most stable, feature detector and descriptor, and since the patent has expired as of the writing of this dissertation, free to use. Additionally, the SIFT feature descriptor gives rise to a good number of features per image, although heavily depending on the content of the image, it has been estimated by (Lowe, 2004) to be about 2000 stable features for a 500x500 pixel image.

3.4.3 Structure-From-Motion

The inner workings of structure from motion algorithms are beyond the scope of this dissertation. Instead, a simple overview of incremental structure from motion will be provided, that said, it is a way of recovering 3D positions from a collection of images, or other media such as videos for the purposes of building a 3D model. This process usually involves, in order:

- Detecting and matching features between images, this involves running the point detection algorithm for the desired feature type, then extracting the descriptors of the same feature type, technically, the point detection need not be the same as the feature descriptor, however, SIFT is both a feature detection and description algorithm, considered the best in the field for now.
- Then, using an iterative process called bundle adjustment to keep a statistic called “reprojection error” below a certain threshold, four by default in the OpenMVG software package, this reprojection error is the distance between where a point is shown to be in the image versus where it is in the 3D reconstruction, a straight line from the point should be where it is in the reconstruction under ideal conditions, taking into account the image pose and intrinsic, this is taken as a measure of error.
- This measure of error is passed along as the intrinsic, namely the camera optical properties of distortion and the extrinsic, where the camera is in the world, including where it is pointing, are refined, optimized until the reprojection error is under the set threshold. Then, a new image is localised into the reconstruction and the process continues until there are no more images to be added, this addition process can also be done in batches instead of a singular image.
- There are also additional extensions to the bundle adjustment algorithm called local bundle adjustment, this means that the process is only applied to the images viewing the points being refined, instead of globally, which is how the global variant of bundle adjustment works, nonetheless, global bundle adjustment still must be ran in that scenario, although only at the end to ensure everything looks right and that there are no disjointed structure pieces present.
- Once the process of bundle adjustment has arrived at a suitable solution under the specified error threshold and additional images cannot be localised, one is left with a model of the subject and the intrinsic and extrinsic of the cameras as well as their corresponding images, since some images can share intrinsic parameters.

Finally, once the above problems are solved, there is still the matter of extracting the GPS coordinates for each camera out of the reconstruction, and the possibility of doing the same for an image not used in the initial or refinement reconstructions. This means the model will have to be aligned and scaled using an inferred GPS position from a subset of images containing location EXIF (Exchangeable Image File Format) tags.

New images will need to have their features detected, localised into the reconstruction and the camera pose inferred in the model scaled using GPS coordinates.

Additionally, this only addresses the process for a single landmark, but since the model is georeferenced in GPS coordinates, it should not overlap with anything else, as it would have in relative coordinates. This behaviour is extremely desirable for merging the different reconstructions together into a cohesive 3D point cloud of something large scale like a city or perhaps, given more work, on even larger scales. This also addresses the recovery of which landmark an image contains, since the pose in the world would be recovered, it need only use human intuition while looking at a map.

3.4.4 Complexity Problems

While the relevant information on bundle adjustment is given in the Structure-From-Motion section, it is important to note that bundle adjustments suffer from an inherent problem when it comes to large reconstructions, for example, city-scale reconstructions.

“The traditional bundle adjustment algorithm for structure from motion problem has a computational complexity of $O((m + n)3)$ per iteration and memory requirement of $O(mn(m + n))$, where m is the number of cameras and n is the number of structure points.” (K. Mitra & R. Chellappa, 2008)

This means that while it is not exponential in complexity growth, it is cubic, and the number of points detected per image for SIFT is enough so that it quickly becomes impractical to add new images with each taking on the order of minutes to hours.

3.4.5 OpenMVG

OpenMVG is a framework for 3D reconstruction from images, meaning, besides dataset gathering, it unifies all the above concepts and processes into one comprehensive package. It also features various utility functions for georeferencing a model, constraining the image matching and reconstruction processes using pose priors like GPS and localisation of new images into an existing reconstruction, additionally, it mostly stores its computed data in cleartext readable JSON (JavaScript Object Notation) format. Altogether, these functions can be used for localising images and extracting camera locations. Finally, the library comes with binaries that have

reasonable, albeit user-tuneable parameters by default. The use cases for such photogrammetric software were covered in the introduction section.

3.5 Previous work

There have been a few studies on automatic geolocated image dataset gathering, one approach based on crawling Flickr users and their connections, in other words, the Flickr friendship graph, then crawling through the new set of users' images while cleaning the user-defined image tags using search engine suggestions, both to correct syntactical errors, and account for the variability of how users spell one tag. This data was then indexed using the quad-tree data structure, recursively dividing a world map of tiles until a stopping condition is satisfied, unfortunately, this has a maximum depth before it becomes impractical, the paper proposed a solution by dividing the world map into tiles, created only if there are images within the area bounds of said tile, and then the quad-tree data structure was applied to each individual tile (Hatem Mousselly Sergieh, Doeller, Elod Egyed-Zsigmond, & Harald Kosch, 2014). The shortcoming of this work is that it only dealt with accurate GPS tagged images and did not cover any technique for including untagged, unlocalised images in the dataset.

First, a method for pose estimation given a set of geolocated images was given by (Akihiko Torii, Sivic, & Pajdla, 2011). The study proposed a method of localising images without Multiple View Geometry by constructing a graph, nodes being a image's visual features, weighted by how often they appear in the image to how often they appear in the entire set of images as well as their location on a planar map, and the edges being their links to their spatial neighbours by visual feature matched homography. The localisation problem was then defined as an interpolation of GPS location by image similarity. This method while functional, and scalable to millions of images was simply not as accurate as Multiple View Geometry approaches and suffered from a high number of mismatched positions. That said, the study only used a pair of views for localising a new image even though it is generalisable to a higher amount of localisation query images.

Another work dealt with direct 2D-to-3D descriptor matching for image localisation, the paper dealt with the problem by treating the entire reconstructed location as one big image's worth of descriptors and matching a novel image's descriptors using classic SIFT matching. Then, they used a six point Direct Linear Transform (DLT) and

RANSAC algorithms for triangulation, essentially, they used six matched features to estimate the camera position. Additionally, they extended it using a prioritisation scheme by searching through all 3D point descriptors associated with a certain visual word (Sattler, Leibe, & Kobbelt, 2011). This method does not estimate the intrinsic parameters of the camera which leads to lowered usability for extending reconstructions using the additional registered images.

Finally, a method for localising images as well as estimating the novel images' camera intrinsic parameters was detailed by (Li, Snavely, Huttenlocher, & Fua, 2012). This method contrasts with the previous approach which only approximates a camera pose without the near pixel level accuracy needed for various augmented accuracy applications. This paper demonstrates the ability to use contemporary Structure-From-Motion techniques for accurate pose and camera parameter estimation in relation to a 3D model. Additionally, it highlights the problem of spurious 2D-to-3D matches for bigger 3D model datasets simply due to a much larger amount of plausible image registration locations. Unfortunately, this approach seems to require a reconstructed point cloud by way of Multiple-View Geometry which is rather computationally expensive.

The shared shortcomings of included localisation strategies are that they rely on having a dataset of mostly accurate GPS tagged images when they could have had their localisation capabilities augmented by the orders of magnitude more available unlocalized high-resolution imagery. The geolocated image datasets approach may also not be feasible for hard to access places, places that have not yet benefited from large spread adoption and usage of accurate GPS image-capture devices, or simply places that are located within areas of poor GPS reception, which leads to a gap in the reconstruction and localisation capabilities of the current methods. Additionally, the current research did not cover a way to handle outlier data points for faulty GPS tags due to their usage of solely accurate geocoded images nor does it investigate the localisation accuracy for novel images within the GPS datum in a simple manner, such as metres of distance between the expected and actual position of an image.

3.6 Summary of previous work

In conclusion, there does not seem to be much research on the collection of accurately GPS tagged images of a specified target, instead focusing on gathering all available

GPS tagged images. Next, while there are previous attempts at localising images into GPS coordinates, with and without including photogrammetric 3D reconstruction processes, the latter taken care of by interpolating image positions between N matched views, a vast majority of them seem to rely on a gathered dataset of accurately geotagged images and hardly any put the localisation accuracy into easily understandable terms, like metres from actual position, nor do they compare the localisation accuracy to other localisation solutions, for example, GPS.

4 Methodology

In this section, the tools and materials used will be disclosed and the project implementation with the reasoning behind said design choices will be justified. Additionally, the resultant project output data will be analysed and compared to the benchmark metric of assisted GPS tagged image accuracy.

4.1 Aims

The aim of this dissertation is to ascertain whether accurate GPS locations can be recovered from a photogrammetric 3D reconstruction of a popular landmark, and whether the image dataset for the reconstruction and georeferencing of the 3D model can be acquired semi if not entirely automatically from open image sources.

4.2 Objectives

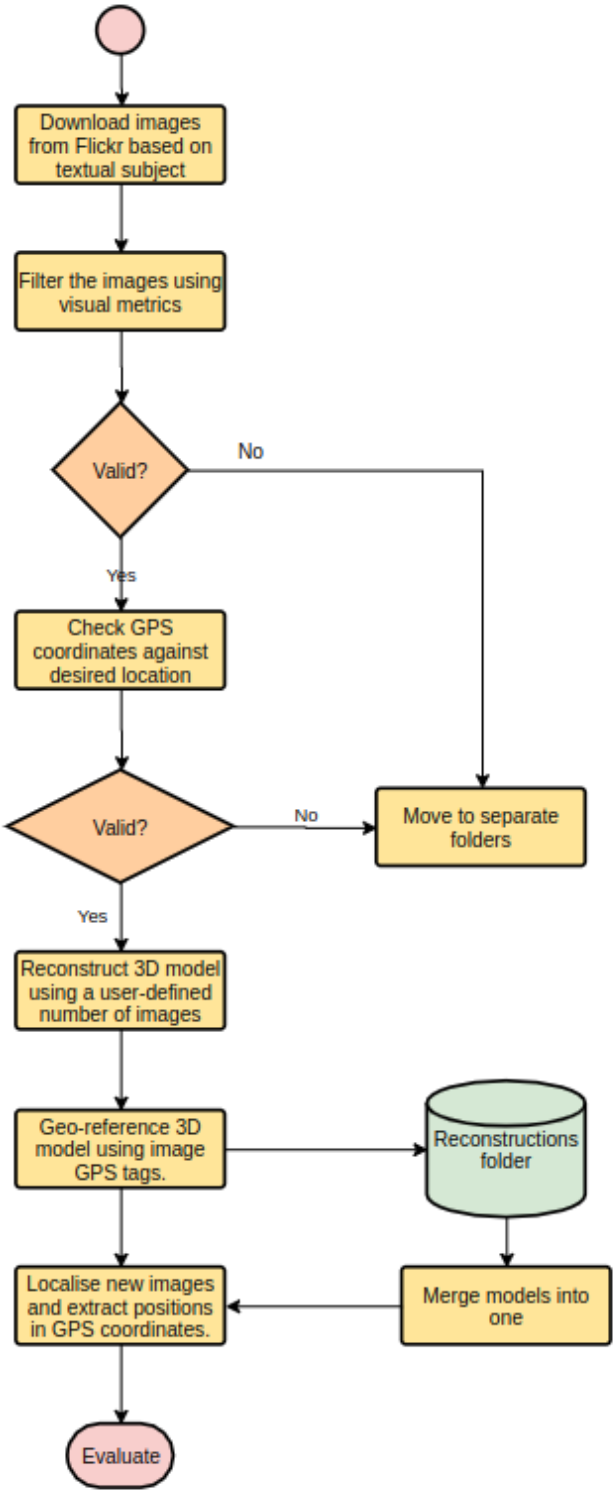
To achieve my aims, the following objectives will need to be achieved:

- Download a variety of plausible landmark imagery based on a query text.
- Filter the bad images out of the dataset, based on factors of enough resolution, little blurriness, no duplicate images and accurate GPS tags.
- Create GPS coordinate aligned 3D reconstruction using Structure-From-Motion techniques.
- Localise new images into the 3D model to compare to their ground truth for an accuracy rating and to iteratively refine the model as more images are localised.

- Merge multiple models together aligned to the GPS coordinate system for higher localisation ability.

A visualisation of this process is available below:

Figure 2 Project Flowchart



There will be several outputs for the project, a rather accurate point cloud of a popular landmark which can later be used for a dense point cloud and eventually a mesh that can be included in any number of digital asset libraries, such a library being a collection of objects that can be included in the creation of other digital media, for example, videogames and movies. The main differences between the three is that a sparse point cloud is usually only used for registering the camera positions and estimating the camera intrinsic parameters, a dense point cloud is more commonly used for reconstructing the entire object with a great many less holes for high quality mesh generation or inclusion into 3D scenes like videogames whose collision calculations benefit from the additional density, finally, a mesh is a 3D model made of polygons, these meshes can then be put through a process called decimation where they are simplified into being more efficient in a video game. A simple visualisation will be provided for the recovered positions using any compatible viewer and a Keyhole Markup Language (KML) file.

4.3 Scope and Limitations

This project will be limited in scope both in terms of the things done and in terms of the things described, this is because:

- I am not a photogrammetrist, or data scientist, and hold no experience or knowledge about the field in a professional capacity, this means; the techniques and parameters used for reconstruction are a combination of a best guess, the software package defaults and light sanity checking by way of manually inspecting the resultant point clouds and image locations.
- The reconstructed locations were only from the facades most photographed by the image sharing platform’s users. Additionally, only the outside of the landmarks was considered even where inside reconstructions could have been due to GPS tagged images being assumed of lower accuracy while inside.
- Multiple 3D reconstruction tools were not evaluated due to problems in designing an interface for each set of storage conventions used and not all of them featuring robust image localisation functions. Additionally, not all the tools considered even provided functions to georeferenced models without manually tagging ground control points in a few images.
- Another limitation is that the localisation positions are not compared against a known ground truth, that would require pinpoint accuracy only granted with the

use of a laser scanning device like LiDAR or a dataset made with the use of such. Such datasets could not be found, and I must do the next best thing, the assumption that images containing a bearing tag are more accurate, and this seems to hold true for the ones checked manually. Another sanity check was also implemented in checking the distances to the reconstruction subject longitude and latitude, in the hopes of rejecting replicas located in other countries, or widely inaccurate GPS tags.

- Next, the assumption that those bearing tags containing images are also accurate in listing the correct altitude was made, this also seems to hold true.
- The computational time it would take to do every experiment conceived during the work for best results or choosing a specific path to go would be more than the time until the deadline of this project. Similarly, not many landmarks were computed for the same reason, this is only a proof of concept. This is also the reason for not using anything beyond five hundred images per reconstruction, it simply takes too much time to be of any worth for the diminishing returns on localisation capabilities.
- The spurious feature matches problem for bigger collections of 3D model points and descriptors was not considered, this means that the implementation for this dissertation is not viable for ever continuing 3D model extensibility for higher localisation capabilities.

4.4 Hardware

The system used for this project has the following specifications, for estimating time requirements on other machines, only relevant items are included:

- CPU: Ryzen 9 3900X @ 4.1GHz
- RAM: 4 x 16GB (64GB total) @ 3200MHz
- SSD: 512GB Sabrent Rocket NVMe
- HDDs: 2TB (2 x 1TB) in RAID0.
- OS: EndeavourOS (ArchLinux-based distro) with kernel version 5.11.16.

Although this is an above average amount of system memory by contemporary standards, it should be noted that the feature detection process can take even more than that depending on the allowed number of threads and describer pre-set. The reconstruction process used does not scale up well and would instead benefit from

faster threads. For comparison of runtime on other machines, the process usually takes a day for the entire project pipeline on this system.

4.5 Software

- The version control system known as Git and a widespread Git repository cloud hosting platform known as GitHub will be used to manage the code for this project. The code for this dissertation will be available at; https://github.com/wiktoraleksanderkaczor/honours_approaches

- The programming language of choice is Python, version 3.8, alongside a few module dependencies, all listed in the requirements file within the GitHub repository.

- The 3D reconstruction library and tool package called OpenMVG (develop branch commit; “5e98d504bb76ba2d1d07ae80ac2acb10b3d6f97d”) will be leveraged for the initial 3D model reconstruction, georeferencing and image localisation.

- The CloudCompare point cloud viewing software, version 2.11.3 (Anoia), stable edition for Linux 64-bit. This was used for viewing and manually inspecting point clouds generated by OpenMVG.

- The GpsPrune image data viewing software, implemented in Java, a comprehensive solution to viewing, editing, and converting GPS coordinate formats. This was used for an initial inspection of image dataset GPS data for extreme outlier detection.

4.6 Reconstructions

The reconstructions made as part of this project will be detailed here, a point cloud size statistic and their respective subjective strengths and weaknesses, it also features the Root Mean Square Error (RMSE), i.e., how far from the line of best fit are the prediction errors:

4.6.1 Descriptions of each reconstruction

Table 1 Reconstructions

Subject name	Point cloud size	Strengths	Weaknesses	RMSE

Stonehenge	210,328 points	The circle of stones was reconstructed perfectly, in detail, and only the circle, no other objects. All the major stones, and some of the stumps on the ground were extracted.	It has a fair amount of noise, extraneous sparse points, possibly due to the various image artifacts triangulated erroneously.	0.581722
St Giles Cathedral	160,202 points	The front façade was detailed and clean with comparatively little noise as well as featuring additional neighbouring buildings and statues.	The back half was not reconstructed in the slightest and the sides of the structure lack much discerning detail. Almost blending with other buildings on one side.	0.722516
Sagrada Familia	82,629 points	The detail for the reconstruction extended to even the small pillars and there was an astounding lack of 'noise'.	Only the frontal façade was reconstructed, and even that was with a crane in the background, this leads to lower localisation capabilities.	0.630553
Palace of Versailles	106,504 points	The lines all look straight, no warping due to faulty camera intrinsic estimation.	Only the alcove of the front of the palace was reconstructed alongside a small section to the right, the gate featured in a fair few image was not included.	0.74517
Edinburgh Castle	431,965 points	It was reconstructed completely, from all sides, it is the largest, in terms of sheer area covered, reconstruction.	The lesser seen inner contained buildings were not reconstructed properly, nor were the inner facades of the outer seen buildings.	0.607079
Cathédrale Notre-Dame de Paris	395,300 points	The building was reconstructed from all three of its allowed entry facades, including some extraneous structures, like the statue in the back gardens or a building while approaching from the Seine river.	It is a rather noisy model, there are small points, likely due to visual artifacts on the featured camera sensors, or temporal coherence based on tree growth, it might negatively impact final localisation accuracy.	0.714352
Big Ben, London	188,627 points	It is subjectively a very clean model with lots of additional reconstructed building faces increasing the feature matching and localisation power.	The entire half behind the most popular one is not reconstructed; it seems the pictures were only taken from one side.	0.635672

4.6.2 Images for each reconstruction

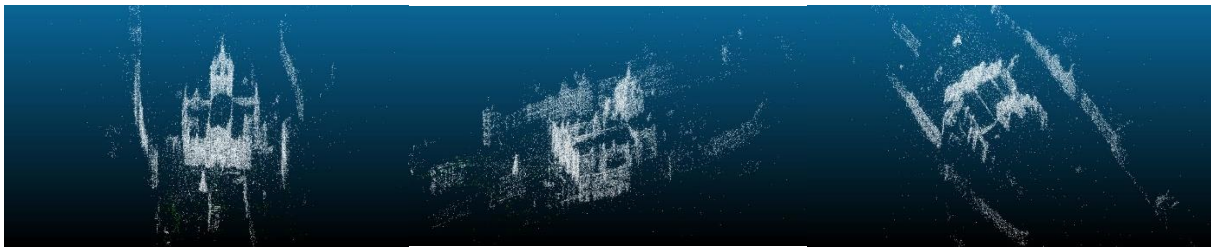
The images are in the same order as they are included in table one, from at least three viewpoints, the green dots are reconstructed camera positions:

4.6.2.1 Stonehenge

Figure 3 Stonehenge reconstruction images

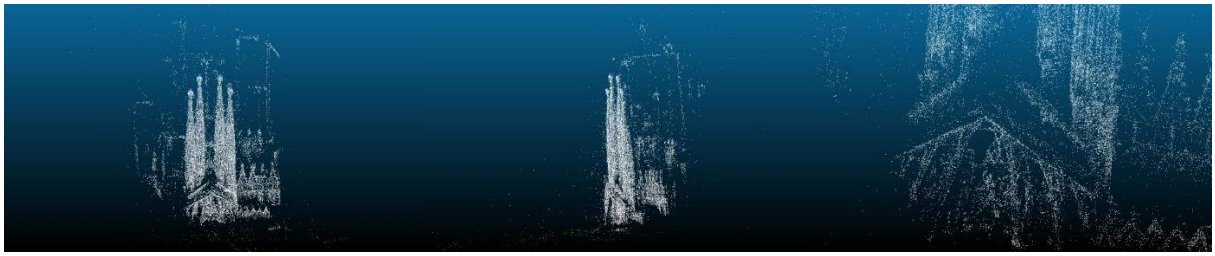
4.6.2.2 St Giles Cathedral

Figure 4 St Giles Cathedral reconstruction images



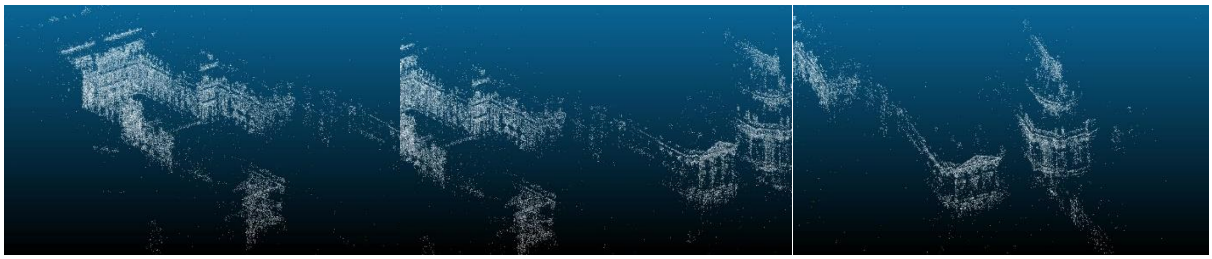
4.6.2.3 Sagrada Familia

Figure 5 Sagrada Familia reconstruction images



4.6.2.4 Palace of Versailles

Figure 6 Palace of Versailles reconstruction images

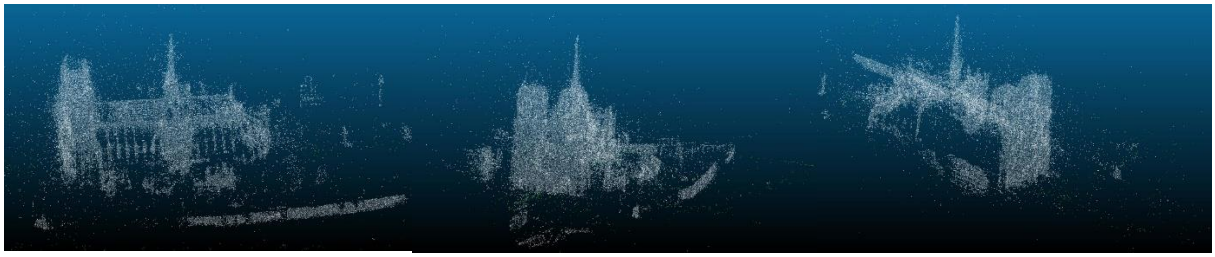


4.6.2.5 Edinburgh Castle

Figure 7 Edinburgh Castle reconstruction images

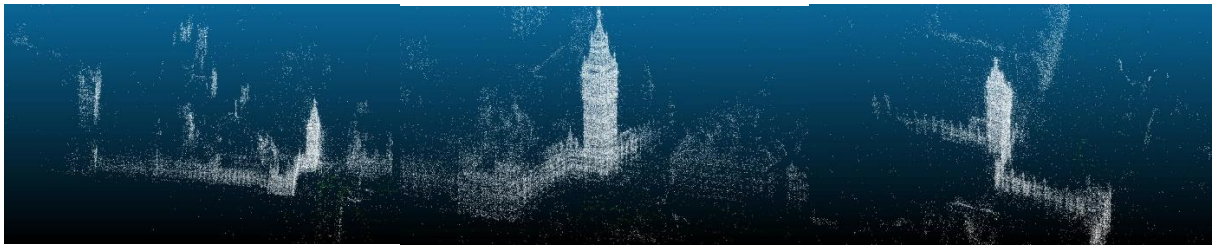
4.6.2.6 Cathédrale Notre-Dame de Paris

Figure 8 Cathédrale Notre-Dame de Paris reconstruction images



4.6.2.7 Big Ben, London

Figure 9 Big Ben reconstruction images



4.7 Approach

The problem is split into three subjects, and explained in the subsections to follow:

1. Acquisition of a suitable photogrammetric dataset including validation data.
2. Reconstruction, model merging and localisation.
3. Analysis of results and comparison to validation data.

4.8 Dataset and validation data acquisition

4.8.1 Image acquisition and visual filtering

To satisfy the requirement of several hundred images of a few famous landmarks with a subset for each containing valid and accurate GPS tags. A couple methods were

investigated for this task, unfortunately, Google makes it incredibly hard to scrape images from their Google Images search results, DuckDuckGo makes it easy to retrieve their image search results, and even download the images, however, it suffers from a high number of images having watermarks, lack of EXIF tags, including the required GPS information and not being very high-resolution overall, the last considered was a self-hosted version of Searx, unfortunate, this suffered from the same problems as all other search engines. Finally, the platform chosen was the image-sharing website called Flickr. Flickr seems to provide a free, key-authenticated, albeit rate-limited (server load alleviation) API to most of their website services, including search results and easy links to download them. Additionally, the programming language of choice, Python, has a handy and well-maintained, although unofficial, module dealing with the Flickr API.

The decision to use Flickr led to writing the code for retrieval of image links from said website, unfortunately, retrieving untargeted images is not very useful. Thus, the project requires a textual query for the landmark to be reconstructed, for example, the “Palace of Versailles” or “Notre-Dame Cathedral”. Additionally, while a Flickr API representation of an image is retrieved, it does not automatically lead to a downloadable link, Flickr holds its image in a variety of sizes, the highest available size, with special priority given to the original files, was chosen for the highest amount of detail available, and thus the greatest number of matchable features, and the images were downloaded. Next, all available links will be retrieved even if the downloading will stop at a certain point, monitoring code and early stopping behaviour was implemented to combat this issue. The relevant code is available in appendix four under “Download”.

The images downloaded were usually in the JPEG format the reconstruction software accepts but some were in PNG (Portable Network Graphic) or BMP (Bitmap Picture). These images were converted to JPEG by usage of an image conversion function from the popular ImageMagick suite. The original unused format images were then deleted. Next, another utility called “jpeginfo” to ensure their encoded form was still readable and the images that did not pass this test were immediately deleted. The relevant code snippet is available in appendix four under “Conversion”.

Then, after some manual inspection, it was determined some of the images in the dataset were duplicates, and since this additional redundant information, or very small

viewpoint changes are not useful in the reconstruction process, they had to be filtered. This was achieved by an application of image hashing, the difference hash (dHash) algorithm, the hashes can then be subtracted from each other elementwise, the result is the hamming distance of the hashes and can be used as a direct measure of whether the images are identical by a threshold value. The images that failed this test were moved to a duplicates folder. The dHash method is invariant to a couple useful properties, for example, colour changes, small perspective changes, image size and aspect ratio i.e., how it might be stretched or altered in colour. A threshold of five was determined to be the best after manual inspection of the pairs detected as identical, it was robust against false positives, and still eliminated very close image matches, alongside their rather minimal degree adjustments in viewpoint. The code for this is available in appendix four under “visual checking”.

Another check implemented was a filter on the number of pixels in each image, each image not containing at least 307200 pixels, otherwise known as 480p or the resolution 640x480 was moved to a separate folder. The final photometric filter was a Laplacian variance, essentially, a Laplacian filter is applied to an image for sharp edge detection, the edges which rise and fall quickly, the variance of this resultant derivative image is taken as the metric for whether an image is sharp, i.e., the camera was well calibrated and focused when taking the photo, if this metric is under a certain user-defined value, the image is considered good enough. The default number used, 125, was determined by running the method against a dataset of variably blurred images, and manually checking to see at which point it provided sufficient protection against the unusably blurred images, and where it picked up false positives. The code for this is also available in appendix four under “visual checking”.

4.8.2 Additional options

Additional options that could have been explored, and were, in some part, are image clustering techniques, for example, using deep neural networks to extract features, brain-like representations of features, like spires, or building, etc. before matching those features to find images featuring the same object, or building, hopefully specific to façade, in the hope of skipping the original image matching stage and saving computation time in later stages to increase the size of the reconstruction. Those would have run on the graphics card and would have hopefully utilised more specialised hardware to save the general-purpose processor. Unfortunately, those features, in the

way it was implemented for a prototype, did yield clusters of similar images but did not put similar clusters of images together, plus, some of the clusters had “noise”. This additional method was then discarded, both because of the performance and in favour of remaining a general processor solution only, accessible to everyone.

4.8.3 Validating the validation dataset

Once the images were sorted into their folder structure based on visual features, the project needed a ground truth dataset, a reconstruction geo-registration dataset and a normal image reconstruction dataset:

First, the project needed a good number of images for the 3D reconstruction, as mentioned in the background section, those images work best if they have a lot of visual features that can be matched to other images and eventually, those features used to register the position of the image in the registration. Unfortunately, the image and feature matching as well as the actual reconstruction processes take an increasing amount of time as the number of images increase, and even more so for higher resolution images in the geometric verification stage of feature matching. This was done by, first, prioritising high-resolution images to increase the number of detected features, second, prioritising high quality images, achieved in the last section. The overall limit of images used is a user-defined number to be set depending on the capabilities of the user’s computer and the user’s patience. This code is available in appendix four under “Moving images”.

Next, the problems of the reconstruction geo-registration set were solved in the exact same way as the reconstruction dataset except for the limit for images, while still user-defined, it should be adjusted to the semi-automatically acquired images, to maximise the chances of a successful and accurate geo-registration and provide enough images left over for accuracy evaluation. Although the ideal choice for production use would have been to use them all, only fifteen per reconstruction were used. The images were first inspected with the software called GpsPrune, this showed some extreme outliers, for example, some images were in entirely differing countries, to solve this; a simple GPS sanity check was devised.

4.8.4 GPS tags sanity check

Another check was to retrieve the latitudinal and longitudinal position of the landmark in question using the help of the GeoPy Python module, said module includes a function called Nominatim (“by-name”) to query OpenStreetMap to retrieve the data associated with a particular textual query, like an address or popular place name. The distance in metres of each images’ GPS tags was then compared against this longitude and latitude, and if over a certain user-defined threshold, the image GPS tags were considered incorrect, and the image was moved to another folder. Additionally, the GPS images were separated into two groups, ones with a bearing and altitude, and ones only containing latitude and longitude, these were then separated into their own folders. This was done because the images which included both an altitude and bearing in their tags seemed to be accurate, at least for the ones investigated, and only for the latitude and longitude tags. The altitude tags were dealt with using the relative model, explained in the subsection below. The rest of the GPS images used for localisation accuracy testing had their GPS tags extracted into a file, and a version of the image with cleared EXIF tags was saved. Additionally, the GPS images without an altitude or bearing tag were saved in a similar state. This code is available in appendix four under “GPS sanity check”.

4.8.5 Retrospective GPS validation

Furthermore, and although done in retrospect, the GPS image dataset had to be investigated, to see if the positions for each image are where they should be on a map, this was done using the GpsPrune software, it showed that they were off by a bit, the general twenty meter range of error in normal GPS, this would not have mattered had there been a fair bit more accurately GPS tagged images or if the dataset gathering technique used more sophisticated techniques to find images in the first place but it will likely cause wildly erroneous geo-registrations for reconstructions that do not have enough GPS tagged images to stabilise the process.

4.8.6 Validating GPS by leveraging the relative model

Finally, the validation of the GPS positions for geo-registration was accomplished using the OpenMVG built-in Least Median Squares estimation (LMeds) function during model georeferencing, also called “robust estimation”. This method essentially checks that the 3D positions in the relative reconstruction fit with the GPS points defined in the

tags of the images, the best fitting model win i.e., voted on by the majority, the faulty images had their poses and features in the structure deleted. The method used was chosen to be LMeds due to the decreasing sum of errors, max, median, mean, and average, all in metres, for each localised good GPS tagged image or collection of images that make up a model compared to just using all available points. The code snippet for this is in appendix four under “geo-registration”.

The “Total” in the graph represents all reconstructions, the error is only calculated using the latitude and longitude values without altitude being considered and all values are rounded to three decimal places:

Table 2 Georeferencing estimation using all GPS images.

Subject name:	Sum Error	Average	Maximum	Minimum	Images localised	Images for geo-registration
Stonehenge	702.171	26.006	109.284	2.723	27 / 37	12
St Giles Cathedral	907.642	60.509	168.661	11.467	15 / 16	19
Sagrada Familia	204.972	25.622	108.123	7.047	8 / 16	8
Palace of Versailles	33652386.817	1246384.697	2158687.61	4022082.981	27 / 49	5
Edinburgh Castle	398.424	79.685	277.686	7.474	5 / 7	19
Cathédrale Notre-Dame de Paris	25845148.038	807660.876	1062442.972	34755.457	32 / 41	13
Big Ben, London	37167.657	4129.74	35879.704	48.22	9 / 9	18
Total	59536915.7	2058367	3257674	4056915	123/175	94

Table 3 Georeferencing estimation using LMeds 3D similarity outlier filtering.

Subject name:	Sum Error	Average	Maximum	Minimum	Images localised	Images for geo-registration
Stonehenge	306.698	11.359	92.145	0.636	27/37	6
St Giles Cathedral	990.337	66.022	175.625	9.15	15/16	9
Sagrada Familia	480.05	60.006	360.939	1.141	8/16	4
Palace of Versailles	3163.414	117.163	669.437	6.01	27/49	2
Edinburgh Castle	363.636	72.727	290.887	4.386	5/7	9
Cathédrale Notre-Dame de Paris	1148.349	35.886	663.624	1.965	32/41	6

Big Ben, London	2481.734	275.748	1403.129	27.518	9/9	9
Total	8934.218	638.911	3655.786	50.806	123/175	44

Further analysis for this table will be provided in the analysis section.

4.9 Reconstruction, model merging and localisation

4.9.1 Preparation

Now that a suitable, high-resolution, filtered image dataset is available for reconstruction purposes, with a subset of relatively accurate GPS tagged images.

The reconstruction software used, OpenMVG, has its own internal representations of the various elements required for a successful reconstruction, for example, the SIFT features for an image, the geometrically verified matches and the intrinsic camera parameters and poses. It also provides a wide variety of utility functions, exposed using a multitude of commands and command line arguments, used to generate the various intermediary steps' outputs and the final product of a good reconstruction. Below, the various steps involved in processing the gathered dataset are described.

First, the visual features of an image must be extracted, the SIFT features referred to in the background section, invariant to a wide variety of visual distortion properties, unfortunately, OpenMVG has its own internal representation of those features, without the ability to import the plaintext American Standard Code for Information Interchange (ASCII) SIFT features format described by the inventor of said feature extraction, description and matching algorithms, necessitating the use of the general-purpose processor limited feature extraction module present in OpenMVG. While the module is by no means slow, it suffers from a notable problem of keeping detected features in memory, in some cases, leading to out-of-memory errors. There are a variety of options for this function, including using higher densities for feature detection and different feature describers altogether but the options used were the defaults suggested by the OpenMVG documentation.

Next, the features must be matched and geometrically verified, once again, this was done using OpenMVG internal utilities, once more limited to general-purpose processing only. This stage tries to use an image matching algorithm called Cascade Hashing, the description of which is outside the scope of this project, to match images for lowered processing requirements in the next stage, matching individual features,

once again outside of the scope, and proceeding to verifying the geometry behind said matched features, the results are stored in a binary OpenMVG representation. Once again, the options used for this utility were the defaults.

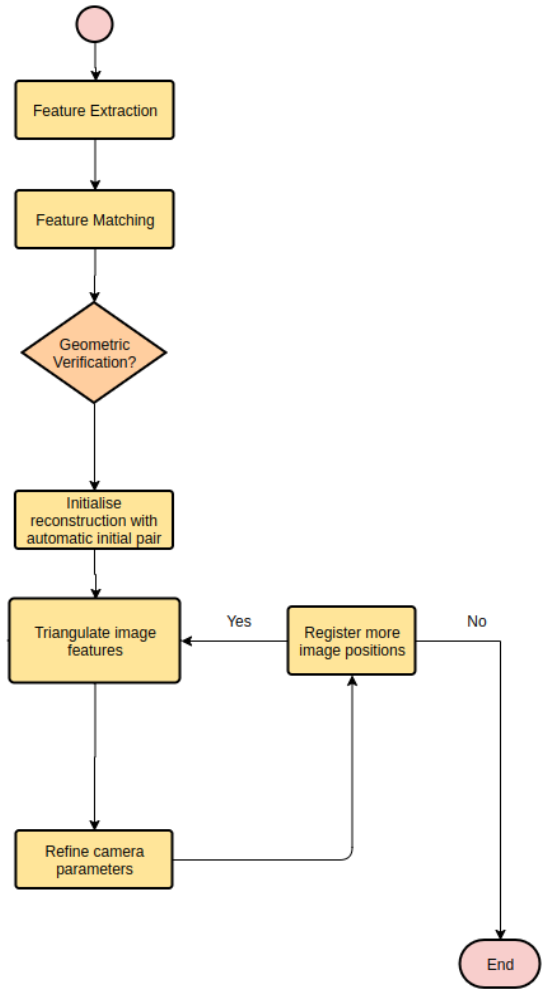
This was accomplished as shown in appendix four under “feature extraction and matching”. The defaults were usually used for all options, that said, where possible, the number of threads used was raised to the available system maximum, and the feature matching option had the “guided matching” option enabled, the OpenMVG word for geometric verification, despite the additional execution time, about two to three times as without, this was well worth it, to keep a greater part of the repetitive structures in the datasets used, the reasons they would be missing detailed in the background section, within the photogrammetry subsection.

4.9.2 Model reconstruction

Next, the two most popular Structure-From-Motion reconstruction pipelines are incremental and global. The incremental pipeline initialises the reconstruction by automatically selecting an image pair to start, usually the images with the highest number of matches with other images, and iteratively registering new image positions while refining their camera intrinsic and extrinsic parameters. On the other hand, the global reconstruction process considers and solves for all image positions simultaneously, unfortunately, this leads to global reconstruction not being as robust to the same amount of error as that of the incremental method. Additionally, global reconstruction tends to favour a dataset of images with a large amount of overlap.

The incremental pipeline steps, to be more specific; initialisation from the most matched pair, incrementally registering images with their features and using the bundle adjustment algorithm to refine their positions and the camera parameters, have already been described in the “Structure-From-Motion” subsection of the “Reconstruction” section in the background. Instead, a flowchart of the incremental pipeline used for this project is provided, more specifically, the implementation the OpenMVG library uses, is shown in figure ten below, to provide a visual guide to the process.

Figure 10 Structure-From-Motion Incremental pipeline



The OpenMVG library provides options for both methods, and thus both were tested for the Stonehenge reconstruction, only one reconstruction was used due to the computational requirements of said process. The results for the incremental pipeline are available in figure eleven, located below, while the results for the global pipeline are available in figure twelve, also located below.

Figure 11 Incremental Pipeline Results

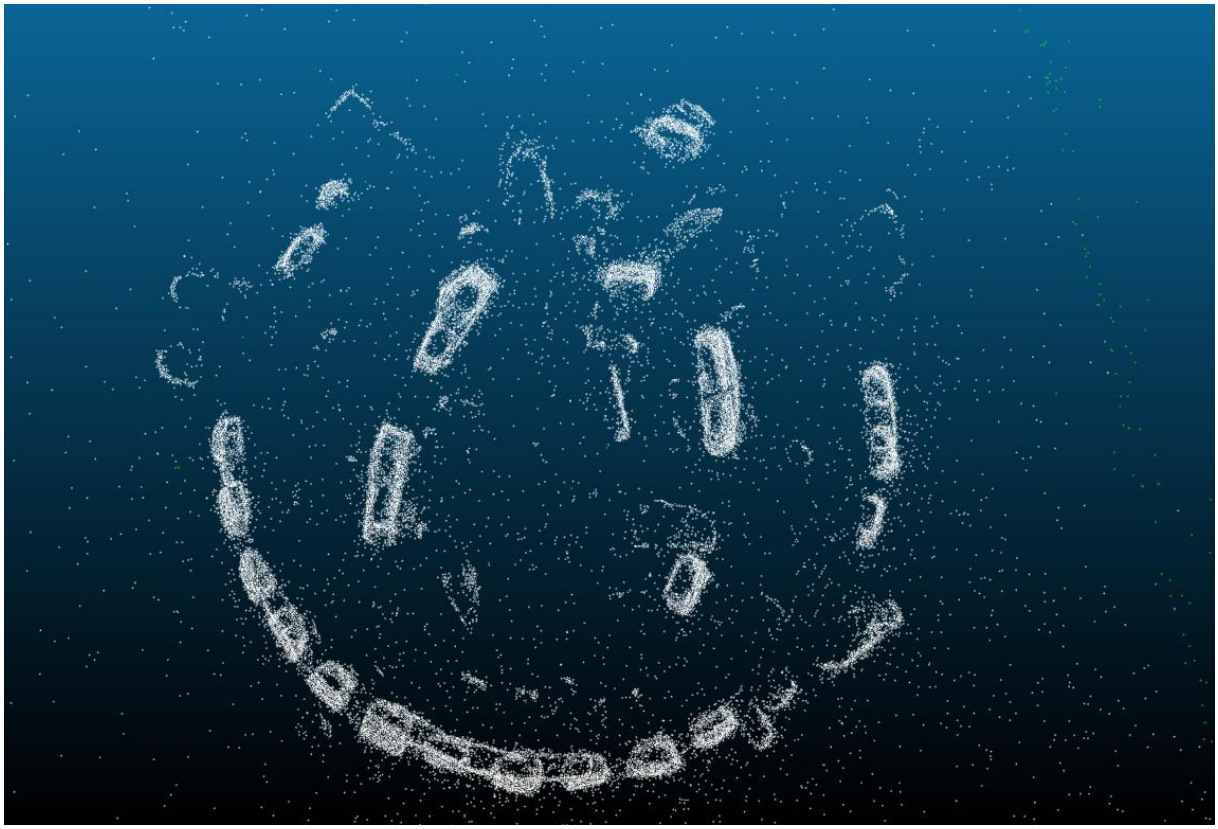


Figure 12 Global Pipeline Results

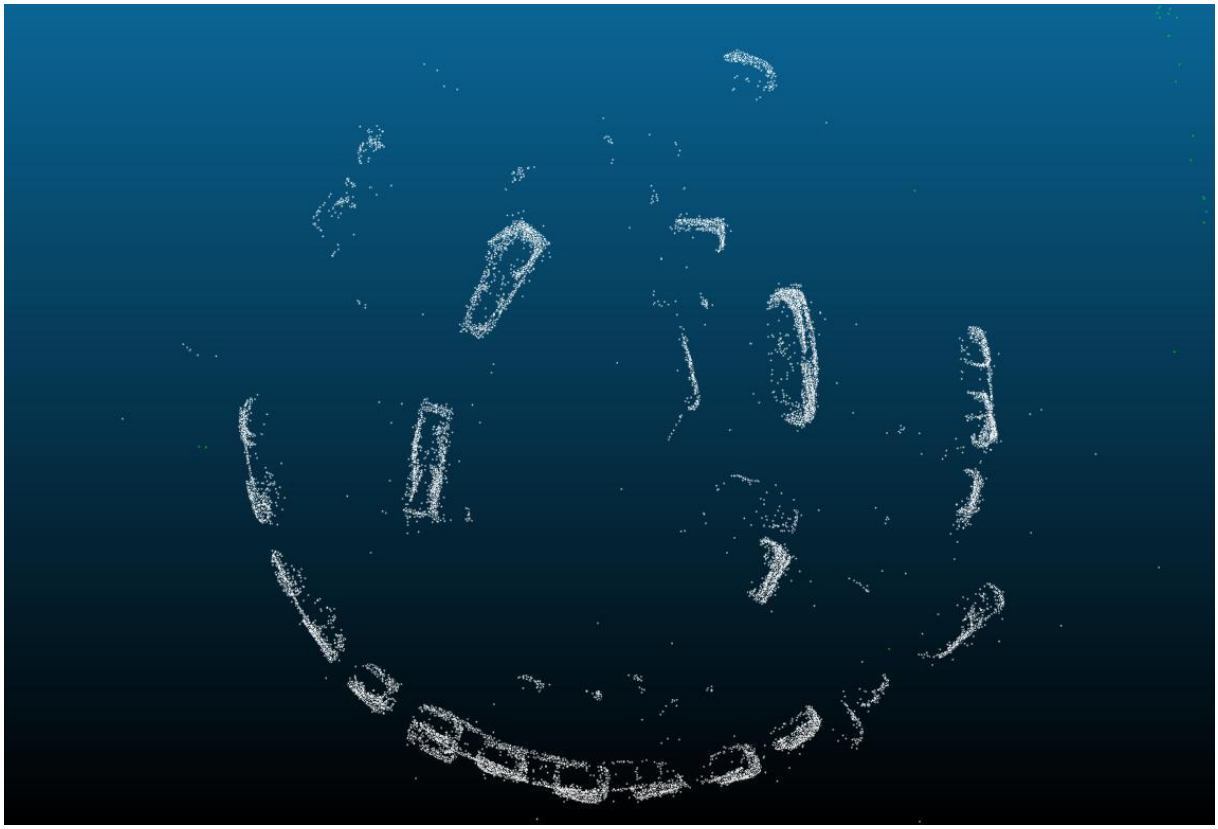


Figure 13 Photographic comparison to pipeline results

Image taken from (Hawkes, 2019).

As shown in figures eleven and twelve compared to the photographic reference in figure thirteen, the global pipeline did not manage to fully reconstruct the scene, the inner side of the circle's façade left too sparse to be usage, the stone on the top left barely a shadow which might get mistaken for an error and the various smaller stones, in another inner circle, left missing entirely while still just a shadow on the incremental pipeline. The reasons this happens is outside of the scope of this dissertation.

Additionally, the table four, located below, shows that despite looking cleaner, the global reconstruction is statistically worse, with a fewer number of matchable points, less registered images, a lesser number of tracks, that is, sequences of images at least two in length for the object motion estimation necessary in reconstruction and greater deviation from the model line of best fit. To that end, the incremental pipeline was chosen for the purposes of this project.

Table 4 Comparison of incremental and global reconstruction pipelines

Pipeline:	**Incremental:**	**Global:**
Number registered images:	417	211
Root Mean Squared Error:	0.581722	0.604633

Number of points:	210,328	40,757 points
Number of tracks:	209,911 tracks	40,546 tracks

The code for using this pipeline is included in appendix four under “incremental reconstruction”.

4.9.3 Model geo-registration, localisation and merging

The geo-registration method, alongside a reference to the relevant code snippet was already described in the section dealing with validating GPS data.

The localisation method used was also a utility included by the OpenMVG package. This method’s implementation is a direct 2D-to-3D feature matching algorithm, leading to a lack of reliance on matches to the entire dataset of images and the localisation taking negligible time, especially compared to the initial 3D reconstruction. While this utility provides many options, the options used were the defaults. The relevant code snippet is available in appendix four under “localisation”.

Finally, model merging functionality was achieved by usage of the georeferenced reconstructions JSON representations, writing some simple code to merge them into one file while updating their internal indexes with the values expected and read as valid by OpenMVG. The supporting files such as extracted features were copied over to a merge folder structure, fortunately, this method does not require the binary match files, or a list of pairs, simply the features. Then, the localisation images were also copied over, and localisation was executed using the new model. This works because the models are already geo-registered at this point, since superposition is not possible within the GPS datum assuming such static structures like buildings, they appear, inside the merged file, having the same distance as they would in the GPS datum in real life, depending on how accurate the initial geo-registration. The code for this is available in appendix four under “OpenMVG merging”.

4.10 Analysis

In combination with the manually inspected point clouds, as shown in figures three to nine, featuring fairly incomplete buildings, such as a missing façade or two, another problem with the dataset was highlighted, it is not extensive enough for the validation of this project, this was found out by working out that there simply are not enough images between certain views of facades to bridge the gap, join them together and

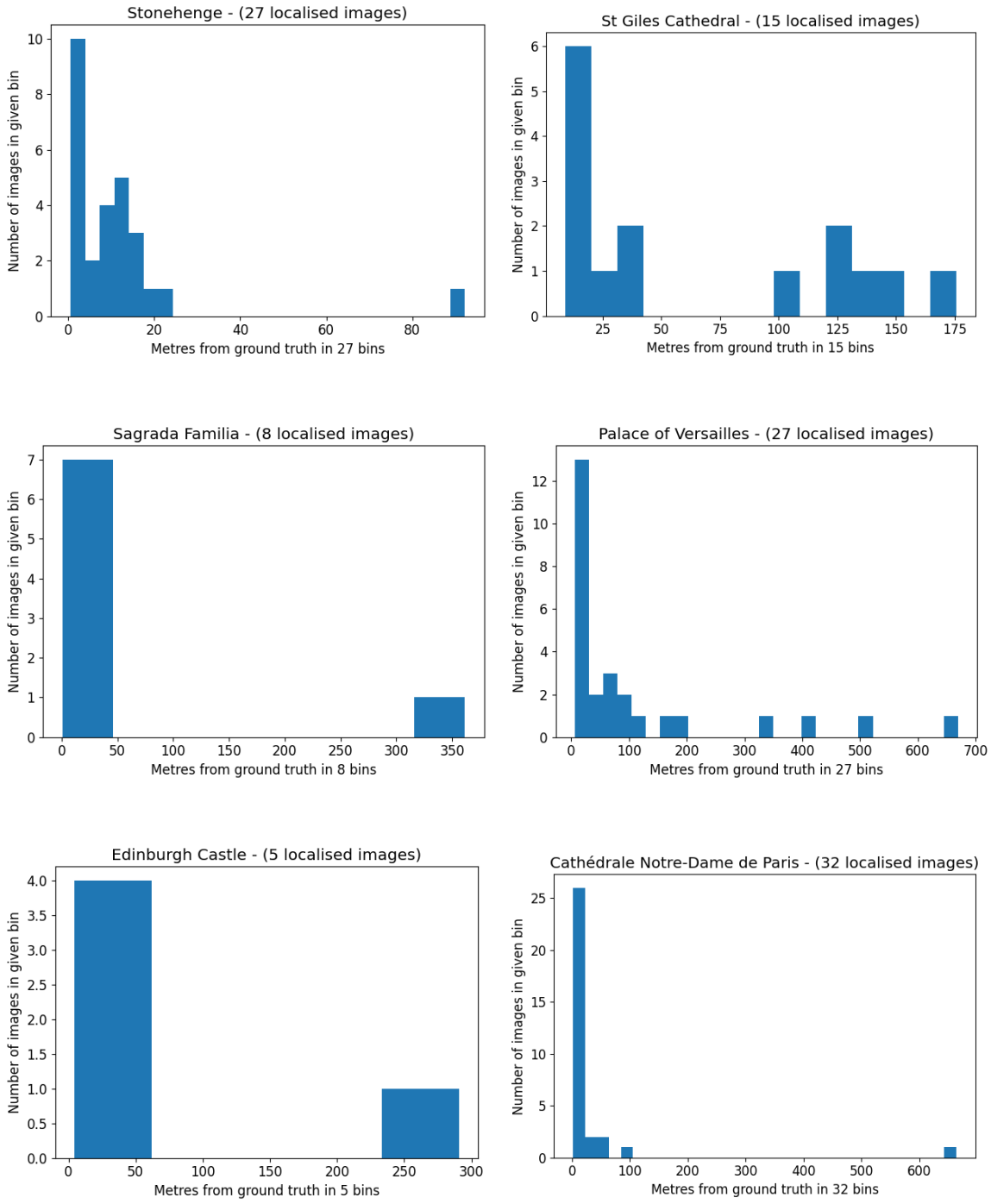
continue the reconstruction, assuming the images to continue the reconstruction exist in the first place, such as the case of the Notre-Dame Cathedral in Paris which has a façade that is not viewable by the general public. Additionally, they also showed a substantial amount of visual noise, extraneous floating points, also mentioned in table one. This was likely because only two views of a point were required for triangulation and could have been avoided had the requirement been set to three or more. The RMSE was not a reliable indicator of reconstruction quality but it was a good indicator for point cloud noise, described later, lower is better.

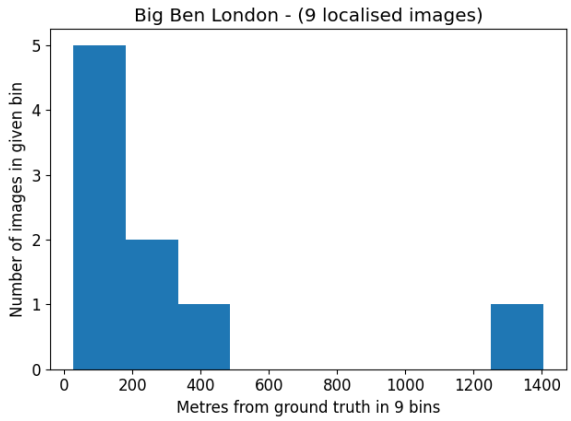
Table two leads to another problem, some of the reconstructions have more than fifteen images to do geo-registration, this is a problem because only fifteen were copied over and any additional ones must have used different tags that slipped detection, even by a popular way of extracting them, those additional ones were not validated and may have negatively impacted the localisation capabilities. It was not investigated whether those undesirable images were included in the geo-registration process.

Table three gives valuable information for the metrics of project success, and the viability of each reconstructed model being used for image localisation, that is, the average localisation error for good GPS images kept out of the reconstruction's dataset, unfortunately, this does not give the entire story, since even one extremely localised image would nudge the value to great heights, however, the localisation errors plotted into N bins histogram, with N being the number of localised images, leads to a more complete picture.

Looking at the histograms below, figure fourteen, the ranges of 'x' values and the lack of middle values in some while being that most predicted position errors are concentrated on or around the lowest value bins with little to no middle values, the preferred bins would cover the range between zero and a hundred metres. This is also well shown in figure fifteen where Stonehenge and Notre-Dame Cathedral dominate the first bin, followed by the Sagrada Familia albeit with its low number of images and then the other reconstructions have their respective, smaller pieces, showing that they can localise images with sufficient accuracy, in some cases.

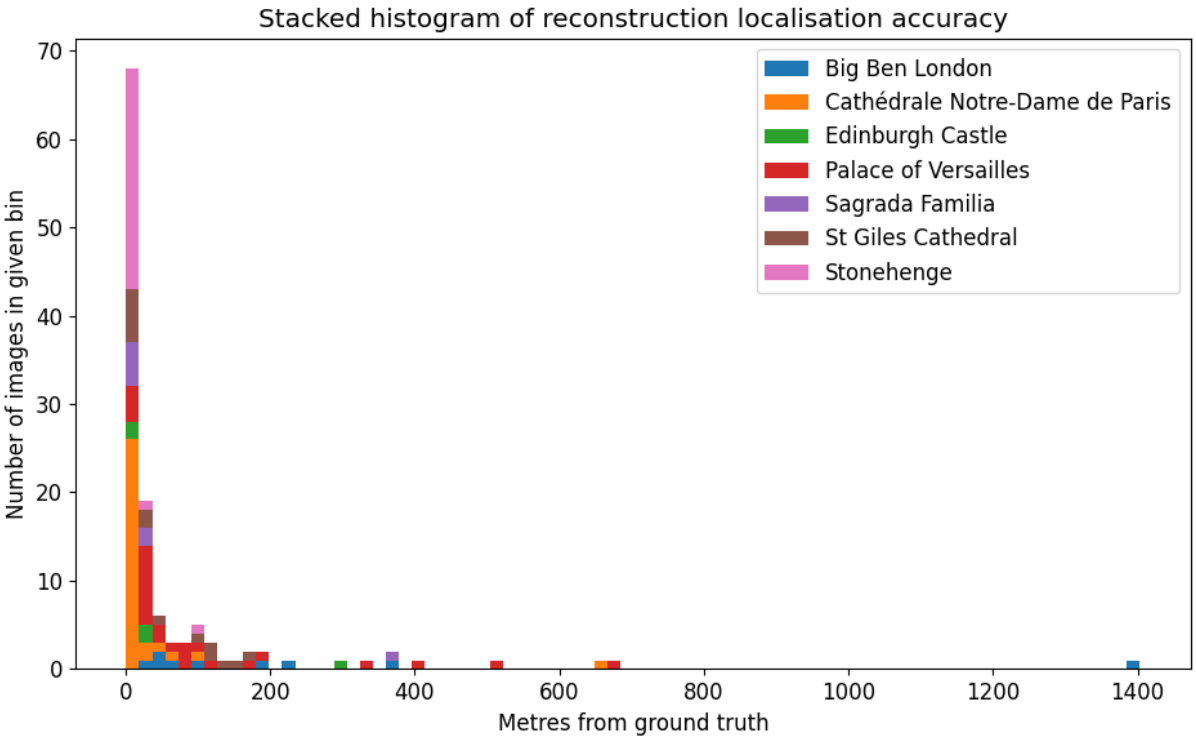
Figure 14 Histograms of localisation accuracies





The data from the above histograms was also taken to construct a stacked histogram of the localisation values in each bin for all unmerged reconstructions. This was done for an easier comparison of accuracy ratings for each reconstruction.

Figure 15 Reconstruction localisation accuracy histogram



The extreme localisation outliers were also investigated manually, to see if they were improperly tagged in the first place, and if the reconstruction were not simply fixing their tags and showing that as a large error from the expected, for some, this was indeed true, and for some, it was simply localising them improperly, the allowed error option could and should have been altered for higher certainty, unfortunately, since no actually known ground truth dataset was used for testing, the prevalence of the

improper localisation problem versus the image tags actually being incorrect cannot be investigated, at least without manually tagging each image with some inaccuracy for the human error of manual localisation.

4.10.1 Merged model to original localisation comparison

The accuracies achieved with this method of merging reconstructions were investigated by the sum of errors, max, median, mean, and average, all in metres, for each localised good GPS tagged image or collection of images that make up a merged model, the pairs of reconstructions for merging each model was decided arbitrarily from the set of the ones not yet merged, the error is only calculated using the latitude and longitude values without altitude being considered, all values are rounded to three decimal places, the titles of the reconstructions are given as below for readability:

Table 5 Merge model localisation changes

Subject name:	Sum Error	Average	Maximum	Minimum	Images localised
Edinburgh Castle and St Giles Cathedral	1322.749	69.618	289.46	4.808	19/23
Above plus Stonehenge	1890.177	40.217	290/204	0.724	47/60
Above plus Cathédrale Notre-Dame de Paris	1361473.193	17233.838	1359358.431	0.754	79/101
Above plus Palace of Versailles	319835.062	3075.337	249964.799	0.392	104/150
Above plus Sagrada Familia	14395616.879	125179.277	13514639.934	0.586	115/166

Table five shows that the change to localisation accuracy varies by the joined geo-registered reconstructions and the localisation process, that is, assuming the localisation images did not get their features wrongly matched to the wrong reconstruction, the scenario not investigated is when reconstructions might be registered such that they overlay each other. That said, there were five missing images in the final merge localisation and a large amount of error in metres, as a sum, for the positions of many images. The former can be addressed by the fact that the pipeline used is not deterministic, the results do not necessarily have to be the same between reruns, it is possible that the images would have been localised had the localisation step been ran again, additionally, the positions gained by localisation are subject to change, once again due to the non-deterministic nature of the algorithm used for localisation. In the case of the final model, it resulted in some images being localised correctly despite the original models, and in some images having error added to their positions, and in some cases not found at all. To get more complete results, the

process would have to be reran with better datasets containing a ground truth, multiple times, or a deterministic localisation method would need to be implemented.

4.10.2 Comparison of merged model to ground truth

In the final, every reconstruction merged model, the localisation accuracy changes as a result of the merge were investigated, this was done by checking whether all images originally localised were still there, how the predicted differences in the final merge differed from each images' original model, and how much closer or further away they were from the original image's actual GPS coordinates, the table for that is shown below, rounded to three decimal places:

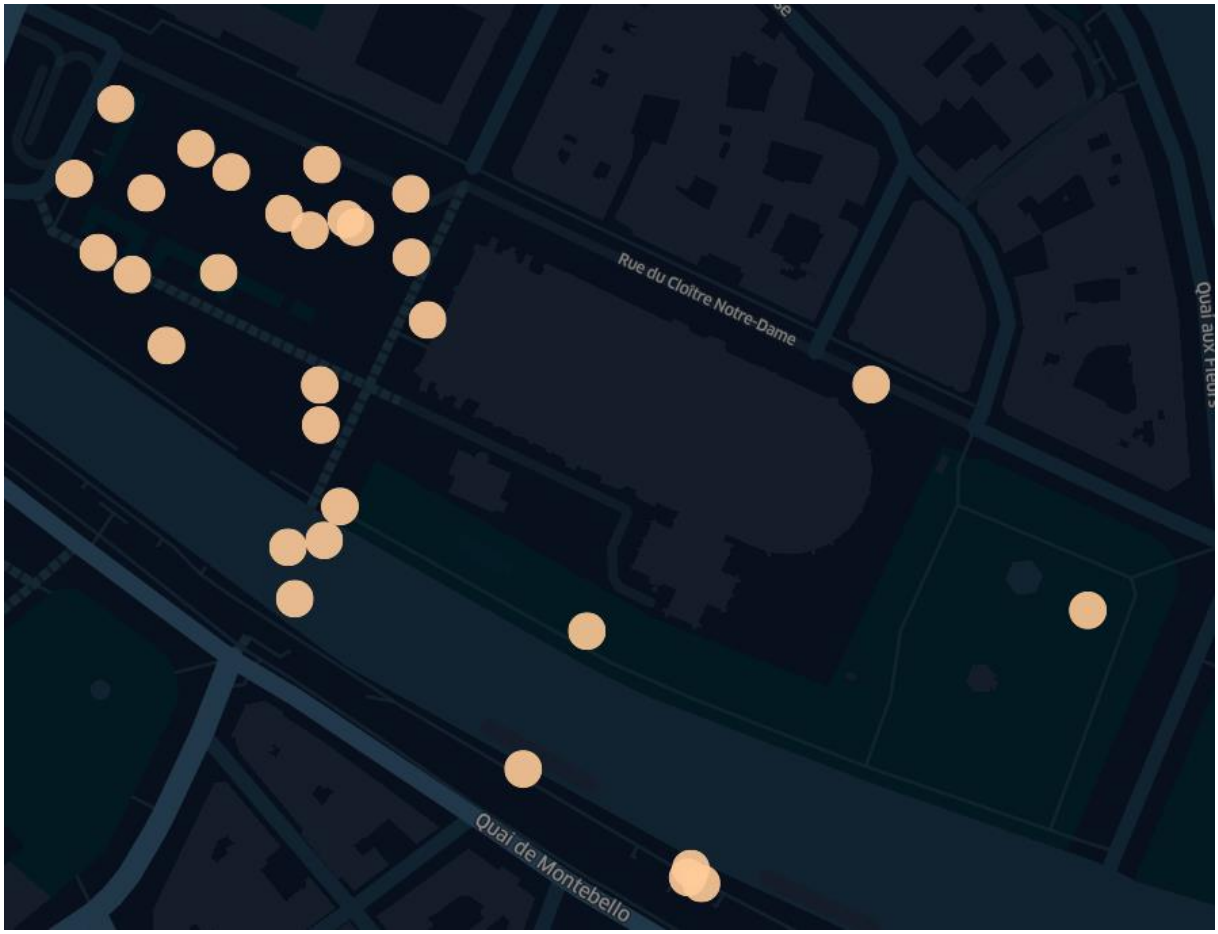
Table 6 Merged to original position changes.

Final merged reconstruction:	
Number of images missing:	5
Sum metres distance changes for each image:	1213.903
Sum metres distance from actual change:	4.714
Average metres distance from actual change:	0.043
Average metres distance change:	11.035
Maximum metres distance from actual change:	143.91
Maximum metres distance change:	306.072
Minimum metres distance from actual change:	-257.168
Minimum metres distance change:	0.001

Table six, above, deals with the accuracy of the merged model in comparison to the GPS data 'ground truth' and the localised positions of each image in their original reconstruction. In essence, it shows that five of the originally successfully localised images are missing, that the final change sum for error from GPS positions increased by 4.714 metres, unexpected compared to the 1213.903 metres the image positions were moved. The benefit to this is that while some images gain a lot of error, as in the case of the biggest gain in error, 306.072 metre, some images manage to get localised much closer to their expected positions, getting closer by 257.168 metres. The average for changes in distance to actual GPS position suggest that the changes are negligible on the scale of multiple reconstructions, at 0.043 metre. On the other hand, the average move for an image position due to different localisation is 11.035 metres, highly unstable. Yet some images seem resistant to this change, the minimum change experienced in the position being 0.001 metres. It was not investigated whether this increases as more reconstructions are merged. An illustration of this average image position change is shown in figure sixteen below, the radius of a point representing

how by how much each localised image would have their position moved on average (11.035 metres).

Figure 16 Image localisation position uncertainty due to model merge



4.10.3 Case study

Finally, a small case study on what makes a good reconstruction for localisation, as in the case of the Big Ben Tower in London versus St Giles Cathedral in Edinburgh; the localisation error sum on them, in order was 2481.734 and 990.337 metres. While both were reconstructed to satisfaction, the average 3D fitting errors for both are 58.7415 and 12.8532 respectively, both with nine inliers. This shows the high relative geo-registration stability of the latter compared to the former, and coincidentally, the difference between a good and bad example, barring reconstruction quality and completeness, the accuracy and stability of geo-registration is the single determining factor, the former achieved with truly accurate GPS tagged images, and the latter with their quantity, due to the position voting process relying on the majority group being correct to reject outliers. In the case of the Big Ben tower, the dataset did not prove favourable due to the methods of gathering it, while in the case of St Giles Cathedral,

sufficiently accurate images happened to be used for the forty-unit smaller error on average.

4.10.4 Conclusion of analysis

The answer to whether this solution achieves its goal of being more accurate than simple GPS in urban environments, where GPS is supposed to do its worst, is that it depends, in the St Giles Cathedral example, manually checked for the extreme outliers and having said outlier images that looked localised properly not considered, led to a three decimal place rounded sum error of 194.924 metres, and an average of 21.658 metres, it should also be noted that the geo-registration method 3D fitting error (in the target coordinate system of GPS) gave the following table:

Table 7 3D fitting error for St Giles Cathedral

3D Fitting Error for St Giles Cathedral:	
Minimum:	3.69096
Mean:	12.8532
Median:	12.7201
Maximum:	19.7174

Additionally, there were two images that were localised on the opposite side of the seemingly structurally mirrored building; the side they were localised on looked much the same, which suggests that there might be problems with reconstructing and matching to uniform structures, and this is supported by popular literature in the area, some examples include but are not limited to; (Saovana, Yabuki, & Fukuda, 2020), (Wilson & Snavely, 2013) and (Tao Xiang & Loong Fah Cheong, 2003), mainly dealing with epipolar geometry verification and deep learning based rejection of outside region of interest features.

Next, there were images that looked perfectly localised in relation to the relative model but their ground truth or model geo-registration not quite right. This was shown in table six. This also leads to the conclusion that should the error have been adjusted for the improper geo-registration and a better ground truth dataset been used for validation, the accuracy would have likely been determined to be well above conventional GPS.

4.10.5 Difficulties

In general, there were a few difficulties encountered; from lack of sufficient dataset gathering techniques resulting in faulty tags on assumed good GPS images, lack of computational power and no usage of truly scalable algorithms limiting the reconstructions to set number of images for reasonable run times and a lack of knowledge of Structure-From-Motion techniques. All of those could have contributed to the observed high error rates, and this project may in fact work perfectly well given a small change, alas, the project does not benefit from an experienced photogrammetrist and produces some results regardless.

5 Conclusion

In conclusion, and in answer to the research question; "*Is it possible for higher accuracy GPS image locations to be obtained from a georeferenced 3D reconstruction than by using on-board mobile phone GPS chips?" …* the varies, depending on the reconstruction and the images that make it up, including the accuracy of the GPS tags of those images, best case scenarios of 0.636 metres and worst-case scenarios 1403.129 metres for singular unmerged models, most images, for a good enough reconstruction, for the purposes of novel image localisation and its social media tagging applications, being under a hundred metres of accuracy and some fitting the majority under thirty or fifty depending on the geo-registration accuracy. That said, there are a fair few image localised at under ten metres of error, which is well under the twenty metres described by the studies on GPS accuracy in urban areas. On the other hand, merged models seem to suffer from images being localised at the wrong model location because of matching features and the non-deterministic nature of the algorithms for localisation, the same problem also occurs on a smaller scale in singular subject models.

5.1 Strengths and weaknesses

5.1.1 Strengths

The previous work, described in a subsection of the background, does not seem to deal with individual targeted reconstructions and their dataset gathering while this project does, albeit in a rather simplistic manner, nevertheless, it did succeed at gathering targeted enough datasets for several reconstructions, for example Stonehenge and Edinburgh Castle.

The localisation accuracies are investigated in easy to understand and visualise error metrics, namely, meters. The project also provides easy visualisation for the accuracy of a given geo-registered model, as well as its localised image positions in the form of a KML file for any compatible viewer.

Although their accuracy was not investigated, the project also features localisation in the Z axis, altitude. Often left untouched and largely unmentioned in previous works. Given accurate georeferencing, this would allow one to guess at the floor and specific

room of a building from which an image was taken in urban environments, leading to applications of modelling wealth from the prices of given hotel rooms on the social media of various people.

5.1.2 Weaknesses

The method used for crawling and retrieving data from Flickr was overly simplistic, relying only on a user defined query for retrieving enough images for a reconstruction, not to mention the plain luck method of finding images to download with valid GPS tags written into EXIF instead of checking the Flickr tags for a location, additionally, as mentioned in one of the papers, Flickr implements an accuracy rating for locations attached to images, those could have been leveraged to weight the importance of the values provided for geo-registration, if they had been retrieved in the first place.

The images retrieved were usually taken from the viewpoint of the most popular facades, in consequence, there were several subject features that were not properly reconstructed, or at all. This could have been remedied by a better image location searching technique as detailed above, and the use of OpenStreetMap Points-Of-Interest around the subject which could have been appended onto the query and those images downloaded too.

The algorithm used for model georeferencing was not suitably robust to erroneous invalid image GPS tags, methods for more accurate georeferencing from papers that dealt with that problem could have been sought and more robust outlier rejection could have been implemented.

Merging models into one was a largely unnecessary task since the images could have just tried to have been localised into each model sequentially until their positions were found. This would have also reduced the problem of matching similar features from other models for erroneous positions as the reconstructions grew.

The project used an older version of the reconstruction software, missing some critical fixes and improvements due to a curious way of naming software branches, and what was set as default, this made the project miss out on additional, more efficient image matching techniques and parallelism improvements that could have enabled the usage of a larger dataset.

Finally, the project suffers from non-deterministic behaviour, leading to localisation likely providing differing results on each run, sometimes, not all images get localised, and sometimes, they get localised in erroneous spots. Unfortunately, no popular techniques currently exist to alleviate this issue, one way of dealing with this problem might be running the localisation process multiple times and taking the mode (highest occurring value) of the estimated poses.

5.2 Further work

Many experiments have been left to do in the future, with more knowledge, time and computational power, there are a few things that could have been implemented, but were not, some of them listed below:

- A better method for crawling through and retrieving datasets from Flickr, and possibly other platforms, should be implemented, ones that would find georeferenced images around the reconstruction subject as well as their accuracy rating. The image searching would also be augmented by the Points-Of-Interest taken from the OpenStreetMap.
- The localised image GPS positions would be exported back into the image tags, said images would then be useful for different reconstruction projects, as a viable speed up in the matching process due to only considering spatial neighbours. Alternatively, the localised, in-reconstruction poses could be used to further extend the structure.
- The usage of video for reconstruction, to save computational power on camera intrinsic calculation and later georeferencing by using localisation on known GPS images and georeferencing using those.
- The used of advanced, neural-network aided image clustering and matching techniques for quick pair list generation to speed up the matching pipeline and reject outliers that do not below to any well-defined cluster.
- Investigating the use of more parallel Structure-From-Motion techniques, such as the OpenMVG Incremental Version Two, Global, or Distributed Structure-From-Motion, for the purposes of quicker reconstructions given the available additional hardware.
- The use of neural networks to estimate camera intrinsic parameters to decrease the time spent on reconstruction and parameter estimation by bundle adjustment techniques.

- Content Based Image Retrieval (CBIR) techniques using five or so reference query images for retrieval of a dense dataset to fill in the gaps. This would decrease the initial outlier count in the database to almost zero.

6 References

Afshan Latif, Aqsa Rasheed, Umer Sajid, & Tehmina Khalil. (2019, August 26). Content-Based Image Retrieval and Feature Extraction: A Comprehensive Review. Retrieved April 19, 2021, from ResearchGate website: https://www.researchgate.net/publication/335397244_Content-Based_Image_Retrieval_and_Feature_Extraction_A_Comprehensive_Review

Akihiko Torii, Sivic, J., & Pajdla, T. (2011, November). Visual localization by linear combination of image descriptors. Retrieved April 22, 2021, from ResearchGate website: https://www.researchgate.net/publication/221430025_Visual_localization_by_linear_combination_of_image_descriptors

Dubey, Shiv Ram. (2012). A Decade Survey of Content Based Image Retrieval using Deep Learning. Retrieved April 19, 2021, from arXiv.org website: https://arxiv.org/abs/2012.00641

Hast, A., Nysjö, J., & Marchetti, A. (2013). *Optimal ransac-towards a repeatable algorithm for finding the optimal set*. Václav Skala-Union Agency.

Hatem Mousselly Sergieh, Doeller, M., Elod Egyed-Zsigmond, & Harald Kosch. (2014, March). World-Wide Scale Geotagged Image Dataset for Automatic Image Annotation and Reverse Geotagging. Retrieved April 20, 2021, from ResearchGate website: https://www.researchgate.net/publication/259802674_World-Wide_Scale_Geotagged_Image_Dataset_for_Automatic_Image_Annotation_and_Reverse_Geotagging

Hawkes, J. (2019). How ancient masons harvested rock for Stonehenge. *Nature*, *566*(7745), 429–429. https://doi.org/10.1038/d41586-019-00565-4

Johannes Lutz Schönberger, True Price, Torsten Sattler, & Pollefeys, M. (2016, November). A Vote-and-Verify Strategy for Fast Spatial Verification in Image Retrieval. Retrieved April 19, 2021, from ResearchGate website:

https://www.researchgate.net/publication/307591406_A_Vote-and-Verify_Strategy_for_Fast_Spatial_Verification_in_Image_Retrieval

K. Mitra, & R. Chellappa. (2008). *A scalable projective bundle adjustment algorithm using the l infinity norm*. 79–86. https://doi.org/10.1109/ICVGIP.2008.51

Kanwal, N., Girdhar, A., Kaur, L., & Bhullar, J. S. (2019). Detection of Digital Image Forgery using Fast Fourier Transform and Local Features. *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*. https://doi.org/10.1109/icactm.2019.8776709

Kim, D.-H., & Lee, H.-Y. (2017). Image manipulation detection using convolutional neural network. *International Journal of Applied Engineering Research*, *12*, 11640–11646.

Krause, M. (2016, April 23). What is translation invariance in computer vision and convolutional neural network? Retrieved March 27, 2021, from Stack Exchange website: https://stats.stackexchange.com/q/208949

Li, Y., Snavely, N., Huttenlocher, D., & Fua, P. (2012). *Worldwide pose estimation using 3D point clouds* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid, Eds.). Springer Berlin Heidelberg.

Lowe, D. (2004). Accepted for publication in the. In *International Journal of Computer Vision*. Retrieved from website: https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf

Merry, K., & Bettinger, P. (2019). Smartphone GPS accuracy study in an urban environment. *PLOS ONE*, *14*(7), e0219890. https://doi.org/10.1371/journal.pone.0219890

Micheletti, N., Chandler, J., & Lane, S. (2015). Structure from Motion (SfM) Photogrammetry. *British Society for Geomorphology Geomorphological Techniques*, *2*(2). Retrieved from https://www.geomorphology.org.uk/sites/default/files/geom_tech_chapters/2.2.2_sfm.pdf

Moulon, P., Monasse, P., Perrot, R., & Marlet, R. (2016a). Openmvg: Open multiple view geometry. *Springer*, 60–74.

Moulon, P., Monasse, P., Perrot, R., & Marlet, R. (2016b). Openmvg: Open multiple view geometry. *Springer*, 60–74.

Paganini, P. (2013, October 25). Photo Forensics: Detect Photoshop Manipulation with Error Level Analysis - Infosec Resources. Retrieved March 27, 2021, from Infosec Resources website: https://resources.infosecinstitute.com/topic/error-level-analysis-detect-image-manipulation/

Pesyna, K.M., Jr, Heath, R. W., & Humphreys, T. E. (2014). Centimeter positioning with a smartphone-Quality GNSS antenna. Retrieved April 20, 2021, from ResearchGate website: https://www.researchgate.net/publication/289955231_Centimeter_positioning_with_a_smartphone-Quality_GNSS_antenna

Rishav Chakravarti, & Xiannong Meng. (2009, May 29). A Study of Color Histogram Based Image Retrieval. Retrieved April 19, 2021, from ResearchGate website: https://www.researchgate.net/publication/224503719_A_Study_of_Color_Histogram_Based_Image_Retrieval

Rosebrock, A. (2015, September 7). Blur detection with OpenCV - PyImageSearch. Retrieved March 27, 2021, from PyImageSearch website: https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/

Saovana, N., Yabuki, N., & Fukuda, T. (2020). Development of an unwanted-feature removal system for Structure from Motion of repetitive infrastructure piers using deep learning. *Advanced Engineering Informatics*, *46*, 101169. https://doi.org/10.1016/j.aei.2020.101169

Sattler, T., Leibe, B., & Kobbelt, L. (2011). Fast image-based localization using direct 2D-to-3D matching. *2011 International Conference on Computer Vision*. https://doi.org/10.1109/iccv.2011.6126302

Tao Xiang, & Loong Fah Cheong. (2003, February). Understanding the Behavior of SFM Algorithms: A Geometric Approach. Retrieved April 29, 2021, from ResearchGate website: https://www.researchgate.net/publication/220660269_Understanding_the_Behavior_of_SFM_Algorithms_A_Geometric_Approach

Wilson, K., & Snavely, N. (2013). Network Principles for SfM: Disambiguating Repeated Structures with Local Context. *Cv-Foundation.org*, 513–520. Retrieved from https://www.cv-foundation.org/openaccess/content_iccv_2013/html/Wilson_Network_Principles_for_2013_ICCV_paper.html

Zhang, W., & Kosecka, J. (2006). Image Based Localization in Urban Environments. *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*. https://doi.org/10.1109/3dpvt.2006.80

7 Appendices

7.1 Appendix 1 Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Non-deterministic approach to location tracking through landmark recognition

Overview of Project Content and Milestones

My project is to determine location from images, hopefully featuring popular landmarks, possibly with camera positions in GPS coordinates, with some fallback if there are no such landmarks present, like the ground type matched with satellite imagery or if the image contains EXIF data, correlating sunrise and sunset times for the timestamp contained. This would likely be done through a combination computer vision-aided feature extraction, matching (possibly using a deep learning solution for the extraction and matching) and photogrammetric reconstruction as well as simple scripting. A possible addition of OCR (Optical Character Recognition) for keywords or language recognition could be included.

The project will be segmented into stages, with the first being ongoing:

- Management of the project using popular project management software solutions including visualisations.
- Research into similar solutions, possible solutions, and a literature review of what already exists for my project.
- Development of my solution using my previous research, including gathering of any required materials.
- Creation of my report, detailing my research, solution, results etc.

In summary, The milestones for my project, including some information from the "Work to be undertaken" section, are:

- Research feature extraction and matching using deep learning and conventional photogrammetric techniques, state-of-the-art and older if applicable.

- Research ground type recognition techniques and find out a way to match it to satellite imagery (including any colour shifts that might arise as the optical instrumentation on different satellites might be different). Finally, research OCR from images to language recognition.

- Gather a dataset of good enough quality for my solution, both for landmarks and any additional deep learning-based feature extraction, like ground type recognition.

- Validate that my solution works properly with proper and hopefully intuitive visualisation of results, a way of constructing a ground truth to validate the camera locations might also be necessary, possibly from images already containing GPS data.

- Write up my report and any relevant research.

The Main Deliverable(s):

The deliverables for this project are:

- A solution for detecting location data from photos.

- A report detailing my process to the standard of undergraduate honours work.

- A high-quality dataset of landmark images and another of multiple environmental features like images containing different soil types etc.

The Target Audience for the Deliverable(s):

I think the target audience for my project would be security professionals for the purpose of people tracking, law enforcement for the purposes of determining the location of crime scenes depicted in digital images, social media platforms or search engines for named entity recognition to do automated annotation and individual users interested in personal geotagging, finally, other researchers could use it as reference material for their own projects, it would serve as a good collection of links to relevant research on the subject topic.

The Work to be Undertaken:

I will have to research feature extraction using various convolutional neural networks, to find something that would be able to find the same object in two images regardless of distortion, occlusion or scaling and be able to compare the image against a large image database relatively quickly. Another way I could go about it would be photogrammetric techniques such as Scale-Invariant Feature Transform (SIFT) to extract features, and compare them with other SIFT features for other images, a certain threshold and it might make for a good enough object recognition method, as well as being linearly scalable with more images. It would also have to be efficient for adding a lot of new landmarks and be able to recognise points on repetitive structures like multiple domes, or a fence, properly.

Another way for faster results is any possible Optical Character Recognition (OCR) in the scene, perhaps a road sign would help narrow down the location. Of course, the language on said road signs would help narrow down the country. One more, slightly expensive method of achieving an even more accurate location, would be through photo-triangulation, I could possibly tell where, relative to a highly popular landmark, a particular photograph might have been taken, with full coordinates and the direction the camera is pointing, but that would require quite a lot of data (some with location in the EXIF for model geo-registration).

I will need to collect a reasonably large database of images, using reasonably open platforms or scraping web data, for example Flickr, Bing and DuckDuckGo or pre-existing collections such as the Google Landmark Dataset V2. The ideal dataset would have a good number of classes, preferably balanced in terms of number of examples per class and possess sufficient resolution with enough for any solution to extract its information.

Next, I will need to develop a way to visualise the results, hopefully in a rather intuitive way; a heatmap of probable locations overlaid on a map, or something like that. A simple Keyhole Markup Language (KML) file could be used if the location is narrowed down with sufficient accuracy.

Then, I will verify my solution works by testing it using some percentage of data kept from the training set to ensure it isn't just memorising answers, for the case of the deep learning model. Next, some non-set images will verify the model doesn't show high confidence on things it knows nothing about either. Basically, I need to verify it shows expected, sane and correct results for most images. The camera location could be verified by running my solution without location data for some images that have it, and checking the provided ones in those images to my calculated ones via a top-down look through Google Earth using KML files.

Finally, I will write-up my research, code, and techniques used with their results, advantages and disadvantages, including the reasoning behind why I went the way I did with my research. This will be in the form of a full academic report, with an abstract, introduction, method, results, discussion, conclusion, relevant references and appendices if necessary.

Additional Information / Knowledge Required:

- Computational photogrammetric techniques; specifically 3D reconstruction, feature extraction and matching as well as photo-triangulation.
- Deep learning, with a focus on convolutional neural networks for feature extraction and recognition.
- Optical character recognition, with a focus on road signs as opposed to documents.
- Various computed data caching techniques, if I don't want to waste time on repeating calculations. Possibly a memory database.

Information Sources that Provide a Context for the Project:

- Google Reverse Image Search does it in a comparatively "dumb" way, as in only checking the images against a large database, not doing any analysis of the actual images.
- https://cloud.google.com/vision/
- Google Cloud Vision AI API, it analyses the image in some of the ways I want, object recognition from the image like saying it has buildings or a cathedral, it also tries to match popular landmarks to the image, and derives the possible location for these landmarks from what I assume to be Google Maps data.
- Colmap, photogrammetry software with a good command line interface, free and with GPU acceleration that can do point cloud and dense reconstructions. (https://colmap.github.io/)

The Importance of the Project:

I think it's an intelligent way to do image matching that hasn't been done before in any large project, something that takes into account what's actually in the image; signs, weather data,

architecture, other features like whether it contains a certain type of building as well as what text might be contained within the image.

It also tries to find out from which side of a popular landmark the photo might have been taken using photo-triangulation techniques from the photogrammetry field. Naturally, once that image is matched to existing data with some degree of certainty, it can be used to further refine the model, leading to even better accuracy over time.

Some refinement to the algorithm and you can find which parts of the data are more reliable in matching a particular landmark with the most confidence, leading to optimisation improvements, an example might be out of the 500 or so images of a landmark, 200 of them usually show 70% certainty of the landmark being the same when compared with other images containing it, while the other 300 show >=90%, the 200 can then be replaced and refined to different, better images.

Additional significance for the project is that it would be open-source, in contrast to the usually proprietary and paid APIs Google exposes for us mere mortals. This also lends some novelty to the project, considering I still haven't found a free project using all the things that can be done with images beside just near-match finding using image hashing.

The Key Challenge(s) to be Overcome:

The main challenges I can anticipate is my lack of knowledge of the majority of the technologies required for this project, so I will have to research and read a lot of material to get up to speed. Even then, as I read any material I will need to learn the mathematics behind it at the same time, which will slow it down to near crawling speed especially since I haven't internalised any of the math concepts either. Finally, the written concise mathematics language used in academic papers will pose the most challenges for me, since it is one I don't have any training in.

Another problem I can foresee is lack of computational power to complete everything within some reasonable time frame and fast storage to feed whatever computer I use in real-time. I already got a more powerful relatively up to current generation graphics card for training models using Tensorflow with CUDA acceleration but even then, some of the larger models are out of my reach due to memory limitations on said card.

Finally, there may be time constraints as my time isn't only dedicated to this project, but also various other modules and whatever work that entails simultaneously. This means that I may need to manage my time efficiently and disregard certain time-consuming paths I could go for the completion of my project.

7.2 Appendix 2 Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Wiktor Kaczor

Supervisor: Sean McKeown

Second Marker: Petra Leimich

Date of Meeting: 27/11/2020

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

- Recorded meetings, regular meetings

Please comment on the progress made so far

- Not concerned about progress, lots of work done
- Looks like lots of the practical work has been done
    - Student says it's pretty much good to go as is
- Need to work on the write-up/lit review

Is the progress satisfactory? **yes***

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

- Reconstruct positional representations and GPS of images based on point clouds
- Map random pictures, such as Instagram, map to places in the world, locate where they were taken
- OSINT / investigative use cases.
- No landmark? No worries:
    - Heat map of potential location based on time of day, foliage, OCR, etc.

Does the student have a plan of work? **yes**

If yes then please comment on that plan otherwise write down your suggestions.

- No spreadsheet/chart yet, but
- Petra: todo:
    - Evaluation, actual experiments
    - Writing, narrative, literature/citations
    - How to capture the points, how to write it up

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

- Ground truth data
- Use images which already have GPS tag and see if removal and reconstruction would re-place it correctly on the map
    - Measured using distance between 2 coordinate points.
- There are also existing datasets to work with

Any other recommendations as to the future direction of the project

- Be clear in the project, and in the code, which parts are your contribution and which parts are re-used from other libraries/projects
    - Modifying code is somewhere in between
- Multiple locations with same name – may be an issue, though project scope won't take this into account.
- More clearly define scope, so we can define what is in-scope and what is out of scope.
- Introduction is too technical too fast -should have a higher-level introduction with motivation, examples of use case, etc. The kind of thing which is probably in the IPO already.
- Check Acronyms for definitions on first use. Glossary?
- CVG github is close to this but doesn't handle GPS or positional data on maps. Position your work with respect to this -> enhancing the approach by

combining it with positional data. This should be in the lit review to make it clear that there is a gap.

- For VIVA:
    - Prepare 5-10 minute show+tell to show off what the work actually does to show off practical work.
    - Recording something in advance would be ideal – can skip over processing times and focus on the main points, outcomes, contribution.

Signatures:	Supervisor	Sean McKeown
	Second Marker	Petra Leimich
	Student	Wiktor Kaczor

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

7.3 Appendix 3 Diary Sheets (or other project management evidence)

The project management plan for this dissertation was recording the supervision meetings as a way of keeping track of what I am doing each week, this was done to minimize the paperwork. In said weekly meetings, we discussed what was done during the week and what was to be done during the coming week as well as touching on the problems encountered, and their potential solutions. The recordings taken can be provided should anyone care to request them.

Changes to the project code, on the other hand were tracked through the code version control system in the form of commit messages. The version control system used was Git and the cloud hosting provider was GitHub. The repository can also be provided at should anyone request them.

Apart from all that, an initial project overview document served as the starting point and statement of objective for this project, it listed; what this project was meant to be, what it was meant to deliver, why it was important that it be completed, who would be the target audience for the project, some information sources as a context for the project and some of the problems that had to be managed for the success of the project. This is available in Appendix 1.

Finally, the last piece of paperwork was the Second Formal Review Output, where the second marker, available in Appendix 2, assessed my work until week nine of the project, provided helpful thoughts and suggestions, and essentially gave the go ahead to the project being on track for academic requirements.

7.4 Appendix 4 Relevant code

7.4.1 Download

```python
# https://joequery.me/code/flickr-api-image-search-python/
def links_from_flickr(topic):
    KEY = '************************'
    SECRET = '************'

    SIZES = ["url_o", "url_k", "url_h", "url_l",
             "url_c"]  # in order of preference

    """
    - url_o: Original (4520 × 3229)p
    - url_k: Large 2048 (2048 × 1463)
    - url_h: Large 1600 (1600 × 1143)
    - url_l=: Large 1024 (1024 × 732)
    - url_c: Medium 800 (800 × 572)
    - url_z: Medium 640 (640 × 457)
    - url_m: Medium 500 (500 × 357)
    - url_n: Small 320 (320 × 229)
    - url_s: Small 240 (240 × 171)
    - url_t: Thumbnail (100 × 71)
    - url_q: Square 150 (150 × 150)
    - url_sq: Square 75 (75 × 75)
    """

    extras = ','.join(SIZES)
    flickr = flickrapi.FlickrAPI(KEY, SECRET, cache=True)
    week = 60*60*24*7
    flickr.cache = flickrapi.SimpleCache(timeout=week, max_entries=99999)

    photos = flickr.walk(text=topic,  # it will search by image title and image tags
                         extras=extras,  # get the urls for each size we want
                         privacy_filter=1,  # search only for public photos
                         per_page=50,
                         sort='relevance')  # we want what we are looking for to appear first

    counter = 0
    try:
        for photo in photos:
            got_photo = False
            for i in range(len(SIZES)):  # makes sure the loop is done in the order we want
                url = photo.get(SIZES[i])
                if url:  # if url is None try with the next size
                    yield url
                    got_photo = True
                    break
            if not got_photo:
                counter += 1
            if counter >= 100:
                return
    except Exception as e:
        print(e)
```

7.4.2 Conversion

```
def images_rename(path):
    files = glob(path+"*.*", recursive=True)
    for image in files:
        split = image.split(".")
        fname = split[-2]
        ext = split[-1]
        if ext == "jpeg":
            move(image, fname+".jpg")

images_rename("intermediate/images/")

commands = [
    # Convert PNGs to JPGs.
    """mogrify -format jpg {}*.png;""".format(
        "intermediate/images/"),

    # Remove PNG duplicates
    """rm {}/*.png;""".format("intermediate/images/"),

    # Check for faulty JPGs, if so, remove.
    """jpeginfo -cd {}*.jpg;""".format("intermediate/images/")
]
for cmd in commands:
    os.system(cmd)
```

7.4.3 Visual checking

```python
# https://www.pyimagesearch.com/2017/11/27/image-hashing-opencv-python/
# https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/
def check_task(image, hashes, RESOLUTION_THRESHOLD, BLURRINESS_THRESHOLD, under_res, too_blurry, hashing=True):
    if image not in hashes:
        img = cv2.imread(image, cv2.IMREAD_GRAYSCALE)

        # Find image size in pixels, if too low, move to under resolution folder.
        height, width = img.shape
        val = width * height
        if val < RESOLUTION_THRESHOLD:
            move(image, under_res + filename(image))
            return

        # Find blurriness value, if too high, move to blurry folder.
        val = cv2.Laplacian(img, cv2.CV_64F).var()
        if val < BLURRINESS_THRESHOLD:
            move(image, too_blurry + filename(image))
            return

        # Hash image.
        if hashing:
            hashes[image] = str(dhash(Image.open(image)))


def get_duplicate_images(path, duplicate, threshold=5):
    print("CALCULATING SIZE FOR EACH IMAGE FILE")
    sizes = {}
    images = glob(path+"*.jpg", recursive=True)
    for image in images:
        sizes[image] = Path(image).stat().st_size

    print("COMPUTING CLOSE IMAGES:")
    files = glob(path+"*.jpg", recursive=True)
    with open("logs/hashes.json", "r") as infile:
        hashes = json.load(infile)

    for key, hash in hashes.items():
        hashes[key] = hex_to_hash(hash)

    close = []
    from copy import deepcopy
    comparison = deepcopy(files)
    for image1 in tqdm(files):
        comparison = {path: hashes[path] - hashes[image1]
                      for path in comparison if path != image1}
        for image2, result in comparison.items():
            if result <= threshold:
                try:
                    # Chose the one with the smaller size to be moved.
                    if sizes[image1] > sizes[image2]:
                        if image2 not in close:
                            close.append(image2)
                    else:
                        if image1 not in close:
                            close.append(image1)

                except FileNotFoundError:
                    # The offending file was likely moved in a previous pair
                    continue
        comparison.pop(image1, None)

    print("MOVING FOUND ITEMS TO 'duplicates' FOLDER")
    for image in close:
        move(image, duplicate + filename(image))

    return close
```

7.4.4 Moving images

```python
def select_and_copy_GPS_images(data_dir, good_gps, NUM_GPS_IMAGES, NUM_LARGEST_IMAGES, openMVG_images):
    # Load previous images used if the file exists.
    gps_images = []
    if os.path.isfile("logs/images_for_georeferencing.json"):
        with open("logs/images_for_georeferencing.json", "r") as infile:
            gps_images = json.load(infile)

    # Sort possible GPS images by size (maximise ability to be included) and set number to be
    # added to minus what is already there.
    possible_gps = sorted(glob(good_gps+"*.jpg"),
                          key=os.path.getsize, reverse=True)

    # Add only enough images to fill the quota, not removing those there.
    # If there aren't enough, all will be used.
    for image in possible_gps:
        if len(gps_images) >= NUM_GPS_IMAGES:
            break
        if image not in gps_images:
            gps_images.append(image)

    # Save the images for later.
    with open("logs/images_for_georeferencing.json", "w+") as outfile:
        json.dump(gps_images, outfile, indent=4)

    # Move images to folder
    for image in gps_images:
        copy(image, openMVG_images + filename(image))

    # Sort by size and move the amount needed.
    sorted_by_size = sorted(glob(data_dir+"*.jpg"),
                            key=os.path.getsize, reverse=True)

    images_used = []
    if os.path.isfile("logs/images_used.json"):
        with open("logs/images_used.json", "r") as infile:
            images_used = json.load(infile)

    for image in sorted_by_size:
        if len(images_used) >= NUM_LARGEST_IMAGES:
            break
        if image not in images_used:
            images_used.append(image)
            copy(image, openMVG_images + filename(image))

    with open("logs/images_used.json", "w+") as outfile:
        json.dump(images_used, outfile, indent=4)
```

7.4.5 GPS sanity check

```python
def get_gps(data_dir, some_gps, good_gps, bad_gps, location, METRES_THR=500):
    files = glob("{}*.jpg".format(data_dir), recursive=True)

    images_with_gps = {}
    nothing = []
    for image in tqdm(files):
        try:
            lat, lon, alt, bearing = get_exif_location(get_exif_data(image))
            if lat and lon:
                # Not really good but there seem to be a lot of images that just say 0m, have to filter.
                if alt and alt != 0:
                    images_with_gps[image] = {
                        "lat": lat, "lon": lon, "alt": alt, "bearing": bearing}
            else:
                nothing.append(image)
        except Exception as e:
            nothing.append((image, e))

    print("GPS FOUND FOR:", len(images_with_gps.keys()))
    print("NOT FOUND FOR:", len(nothing))

    incorrect_items, correct_items = {}, {}
    for image in images_with_gps.keys():
        distance = measure(images_with_gps[image]["lat"], images_with_gps[image]["lon"],
                           location.latitude, location.longitude)
        if distance < METRES_THR:
            correct_items[image] = distance
        else:
            incorrect_items[image] = distance

    for item in incorrect_items.keys():
        move(item, bad_gps + filename(item))

    good_gps_to_save = {}
    some_gps_to_save = {}
    for item in correct_items.keys():
        fname = filename(item)
        # If image has bearing tag.
        if images_with_gps[item]["bearing"]:
            move(item, good_gps + fname)
            good_gps_to_save[fname] = images_with_gps[item]
        else:
            move(item, some_gps + fname)
            some_gps_to_save[fname] = images_with_gps[item]

    with open("intermediate/gps_data_from_images.json", "w") as outfile:
        json.dump(good_gps_to_save, outfile, indent=4)

    with open("intermediate/some_gps_data_from_images.json", "w") as outfile:
        json.dump(some_gps_to_save, outfile, indent=4)
```

7.4.6 Geo-registration

```
cmd = \
"""
openMVG_main_ConvertSfM_DataFormat \
    -i openMVG/output/sfm_data.bin \
    -o openMVG/output/sfm_data.json
"""

os.system(cmd)

LMeds_usage = ""
while LMeds_usage not in ["y", "n"]:
    LMeds_usage = input("Use of the OpenMVG LMeds model for georegistration [y/n]? ")

if LMeds_usage == "y":
    LMeds_usage = "0"
else:
    LMeds_usage = "1"

commands = [
    """
    openMVG_main_ConvertSfM_DataFormat \
        -i openMVG/output/sfm_data.json \
        -o openMVG/output/sfm_data_modified.bin
    """,

    """
    openMVG_main_geodesy_registration_to_gps_position \
        -i openMVG/output/sfm_data_modified.bin \
        -o openMVG/output/sfm_data_geo.bin \
        -m {} \
        | tee logs/georegistration.txt
    """.format(LMeds_usage),

    """
    openMVG_main_ConvertSfM_DataFormat \
        -i openMVG/output/sfm_data_geo.bin \
        -o openMVG/output/sfm_data_geo.json
    """,

    """
    openMVG_main_ConvertSfM_DataFormat \
        -i openMVG/output/sfm_data_geo.bin \
        -o openMVG/output/sfm_data_geo_cloudcompare_viewable.ply
    """
]
for cmd in commands:
    os.system(cmd)
```

7.4.7 Feature extraction and matching

```
commands = [
    """
    openMVG_main_SfMInit_ImageListing \
        -i openMVG/images \
        -d sensor_database.txt \
        -o openMVG/init \
        | tee logs/image_listing.txt
    """,

    """
    openMVG_main_ComputeFeatures \
        -i openMVG/init/sfm_data.json \
        -o openMVG/data \
        --describerMethod SIFT \
        --describerPreset {} \
        --numThreads {}
    """.format(DESCRIBER_PRESET, cpu_count()),

    """
    openMVG_main_ComputeMatches \
        -i openMVG/init/sfm_data.json \
        -o openMVG/data/ \
        --guided_matching 1 \
        --force 1 \
        | tee logs/matching.txt
    """
]
for cmd in commands:
    os.system(cmd)
```

7.4.8 Incremental reconstruction

```
cmd = \
"""
openMVG_main_IncrementalSfM \
    -i openMVG/init/sfm_data.json \
    -m openMVG/data \
    -o openMVG/output \
    --prior_usage 0
"""

os.system(cmd)
```

7.4.9 Localisation

```
commands = [
"""
openMVG_main_SfM_Localization \
    -i openMVG/output/sfm_data_geo.bin \
    --match_dir openMVG/data \
    --out_dir openMVG/localization_output/ \
    --query_image_dir openMVG/localization_images/ \
    --numThreads {}
""".format(cpu_count()),

"""
openMVG_main_SfM_Localization \
    -i openMVG/output/sfm_data_geo.bin \
    --match_dir openMVG/data \
    --out_dir openMVG/some_gps_localization_output/ \
    --query_image_dir openMVG/some_gps_localization/ \
    --numThreads {}
""".format(cpu_count())
]

for cmd in commands:
    os.system(cmd)
```

7.4.10 OpenMVG Merging

```python
def merge_sfm_data(sfm_data_one, sfm_data_two, output):
    print("READING THE TWO FILES FOR MERGING")
    with open(sfm_data_one, "r") as infile:
        sfm_data1 = json.load(infile)

    with open(sfm_data_two, "r") as infile:
        sfm_data2 = json.load(infile)

    new = {
        "sfm_data_version": "0.3",
        "root_path": "openMVG/images",
        "views": sfm_data1["views"],
        "intrinsics": sfm_data1["intrinsics"],
        "extrinsics": sfm_data1["extrinsics"],
        "structure": sfm_data1["structure"],
        "control_points": []
    }

    view_ptr = new["views"][-1]["value"]["ptr_wrapper"]["id"]
    view_key_and_id_view_offset = new["views"][-1]["key"] + 1
    int_id_offset = new["intrinsics"][-1]["key"] + 1
    ext_id_offset = new["extrinsics"][-1]["key"] + 1
    print("ADDING SECOND VIEWS")
    for view in tqdm(sfm_data2["views"]):
        view["key"] += view_key_and_id_view_offset
        view_ptr += 1
        view["value"]["ptr_wrapper"]["id"] = view_ptr
        view["value"]["ptr_wrapper"]["data"]["id_view"] += view_key_and_id_view_offset
        if view["value"]["ptr_wrapper"]["data"]["id_intrinsic"] != 4294967295:
            view["value"]["ptr_wrapper"]["data"]["id_intrinsic"] += int_id_offset
        view["value"]["ptr_wrapper"]["data"]["id_pose"] += ext_id_offset
        new["views"].append(view)

    last_int_ptr = new["intrinsics"][-1]["value"]["ptr_wrapper"]["id"]
    print("ADDING SECOND INTRINSICS")
    for intrinsics in tqdm(sfm_data2["intrinsics"]):
        intrinsics["key"] += int_id_offset
        last_int_ptr += 1
        intrinsics["value"]["ptr_wrapper"]["id"] = last_int_ptr
        new["intrinsics"].append(intrinsics)

    print("ADDING SECOND EXTRINSICS")
    for extrinsics in tqdm(sfm_data2["extrinsics"]):
        extrinsics["key"] += ext_id_offset
        new["extrinsics"].append(extrinsics)

    highest_structure_id = sfm_data1["structure"][-1]["key"]
    print("ADDING SECOND STRUCTURE")
    for structure in tqdm(sfm_data2["structure"]):
        highest_structure_id += 1
        structure["key"] = highest_structure_id
        for observation in range(len(structure["value"]["observations"])):
            structure["value"]["observations"][observation]["key"] += view_key_and_id_view_offset

        new["structure"].append(structure)

    print("WRITING NEW FILE")
    with open(output, "w+") as outfile:
        json.dump(new, outfile, indent=4)
```