UNIVERSITY COLLEGE LONDON

MSc DISSERTATION

# Optimum Frames Selection for 3D from Video

*Mahmoud Abdelrazek*

Supervised By
Dr. Jan Boehm
Mr. David Griffiths

September 2019

## Appendix II – MSc Dissertation Submission

Student Name: MAHMOUD MOHAMED HUSSEIN ABDELRAZEK (BLOCK CAPITALS)

Programme: MSc Spatio-temporal analytics and big data mining (*e.g. MSc GIS)*

*Supervisor:* Dr. Jan Boehm, Mr. David Griffiths

*Dissertation Title:* Optimum Frames Selection for 3D from Video

**DECLARATION OF OWNERSHIP**

- I confirm that I have read and understood the guidelines on plagiarism, that I understand the meaning of plagiarism and that I may be penalised for submitting work that has been plagiarised.

- I declare that all material presented in the accompanying work is entirely my own work except where explicitly and individually indicated and that all sources used in its preparation and all quotations are clearly cited.

- I have submitted an electronic copy of the project report through Moodle/turnitin.

Should this statement prove to be untrue, I recognise the right of the Board of Examiners to recommend what action should be taken in line with UCL's regulations.

| | | | |
|---|---|---|---|
| **Signature:** | Mahmoud Abdelrazek | **Date:** | 09/09/2109 |

## Abstract

One of the pillars of photogrammetry framework is to obtain satisfactory image coverage of the target object (Konecny 2014). Typically, an overlap ratio of 80% between successive images is advisable. This is generally achieved by proper flight planning. However, in a complex urban environment, achieving this percentage can prove challenging. Capturing a video covering the target using is Unmanned Aerial Systems (UAS) is one method that has the potential to be deployed successfully in such situations (Rakha & Gorodetsky 2018). Videos provide over redundant dataset, and thus selecting optimum frames from the video for 3D reconstruction can be crucial for both the required resources and the quality of the produced model. This research investigates possible methods of selecting optimum frames from videos for 3D reconstruction.

Two main methodologies were used: a feature displacement based approach that depends on the movement of features in 2D image coordinates, and visual odometry based approach that estimates the pose and position of the camera and select frames based on it. This is followed by a second stage of image quality assessment, which extracted the sharpest images out of the presented set. Finally, a 3D model is built out of each of the selected frames set and the quality of the whole process is estimated using three matrices.

By comparing the results with a baseline that was established to evaluate the selection processes, it was found that both the methodologies provide a lower number of frames and therefore require less processing time for each model. Visual odometry based method was found to perform better than other methods used.

## Acknowledgements

First and foremost, I would like to thank Allah Almighty for giving me the strength, knowledge, ability and opportunity to undertake this research study and to persevere and complete it satisfactorily. Without his blessings, this achievement would not have been possible.

I would like to thank **Mr. David Griffiths** for his continued help and support without which this study would have not be concluded. The advice he gave me since the selection of the project until a few seconds before the submission is the reason for this work to finished properly. I would also like to extend my thanks and gratitude to **Dr. Jan Bohem** who provided guidance, valuable discussion and access to resources which I would not have been able to produce results without.

I would like also to thank the team of **Trik** ; Fatih Küçüksubaşı, Pae Natwilai, Monika Zibolyte, and Tracey Wright for the access to the data and the companies resources.

Also, I would like to thank Chevening scholarships for providing the funding for my education and making this work possible.

Finally, I would like to thank my **mother, father, brothers** and **sister** for all the love, help and support accorss the years and during my study. Thank you to all my friends who helped make this work possible, and thanks to ACM industries for all the technical support.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Motivation and Problem Statement

Photographs are a reliable means of estimating measurements of objects remotely, and have been in use for decades. Many researchers have studied the extraction of geometric information from photographs, a field that has evolved into a part of photogrammetry. Traditionally, developments in photogrammetry have been connected to advancements in photography and aviation (Linder 2016). However, in more recent years, rapid progress was fueled by the introduction of computers and efficient photogrammetric algorithms. Photogrammetric models are developed using photographs that show the target object from several angles, with the ideal case being one that gives us the entire the surface of the object in a set of overlapping photographs. Obtaining such a set of photographs with the required accuracy depends largely on the platform and the method of image acquisition (Konecny 2014).

Recent advances in Unmanned Aerial System (UAS) technologies have made image acquisition possible at a fraction of the cost and time normally required for aerial photogrammetry. UAS can be used in small-scale urban projects to provide optimal solutions for planning, mapping, modelling and inspection (Rakha & Gorodetsky 2018). The emergence of UAS is changing the methodologies and frameworks of established industries (Rao et al. 2016). It is now possible to obtain a high-quality video for modelling objects using cameras mounted on UAS (Kamate & Yilmazer 2015, Gu et al. 2019, Pérez-Alvárez et al. 2019).

However, in certain cases, using the obtained video directly in developing a 3D model might not be the best solution. High-quality videos feature a higher number of frames and higher image resolution; both can be computationally expensive to process. Additionally, frames within the same video differ in image quality, which might have an impact on the developed 3D model.

Despite the clear utility of using UAS video in 3D modeling, a well-defined, practical framework for extracting optimal frames is not yet available. This study addresses this research gap by investigating a method for selecting frames based on UAS relative location and image quality.

## 1.2 Aims and Objectives

This research work aims to answer the following question:

Does the selection of frames from video improve the resources utilization

and the quality of a 3D model generated based on the selected frames as compared to the 3d model generated based on sampling of the frames at constant temporal rate?

This central research aim is supported by the following objectives:

1. Build a basic frame selection algorithm based on tracking features across frames
2. Build a more advanced selection algorithm based on visual odometry
3. Define and use a method for image quality assessment
4. Evaluate and compare the methods used
5. Develop a a more critical understanding understanding of photogrammetry

# 2 Literature Review

This chapter aims to explore the relevant literature, prior research into areas of interest, including photogrammetry, feature detection and selection, and image quality.

## 2.1 Photogrammetry

Photogrammetry is the science and technology of deriving geometric information about objects using image measurements. It revolves around the idea of restoring 3D geometry of the shape, size and location of objects using the 2D representation on photographs. A simpler form of it can be traced back to the beginnings of photography, where optical and mechanical plotting instruments were used (Sandau et al. 2010). The shift from mechanical and analogue electronic technology to digital electronics allowed for faster and more accurate solution to the photogrammetric restoration process(Sandau et al. 2010).

Photogrammetry can be classified based on camera position and object distance into: a) *macro photogrammetry*, where images have scale of less than one, b) *close range photogrammetry*, where the distance is less than 300 meters, c) *aerial photogrammetry*, where aerial photographs are captured from a height of 300 meters or higher, d) *terrestrial photogrammetry*, where measurements are collected from a fixed terrestrial location, and e) *satellite photogrammetry*, where images are collected using a satellite orbiting at 200 Kilometers or higher (Luhmann et al. 2014).

This research can be classified under close range photogrammetry, as most of the videos are collected from less than 300 meters, despite using UAS in acquiring the videos.

### 2.1.1 Geometric background

Extraction of geometric information about an object requires at least two images overlooking at least one feature of that object as shown in Figure 1. The information needed to solve for the location of the object point on the image plane includes both intrinsic information, which is related to the camera used for collecting the images, and extrinsic information, which relates to the position of the object to the camera (Szeliski 2011).

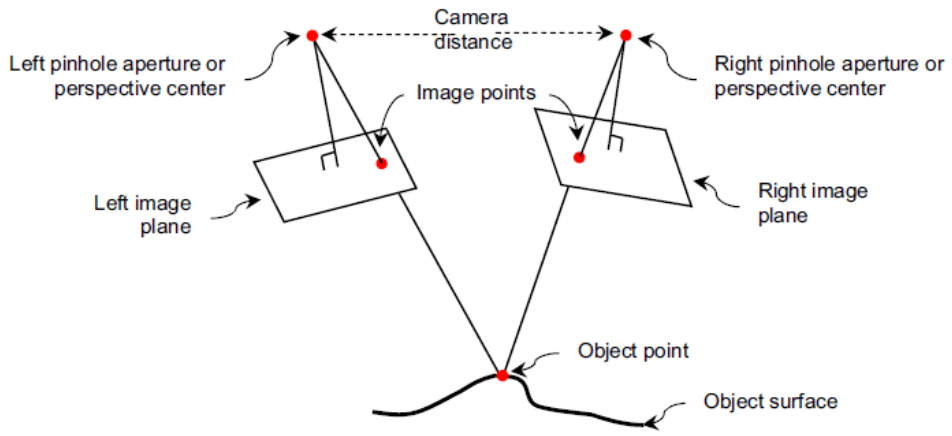


Figure 1: after (Thevara & Vasanth Kumar 2018)

To solve this problem a camera model needs to be assumed. The simplest one is the pinhole camera model that assumes no lens, but a hole where the light passes through a single aperture and projects inverted on a plane, to form the image as shown in Figure 2.

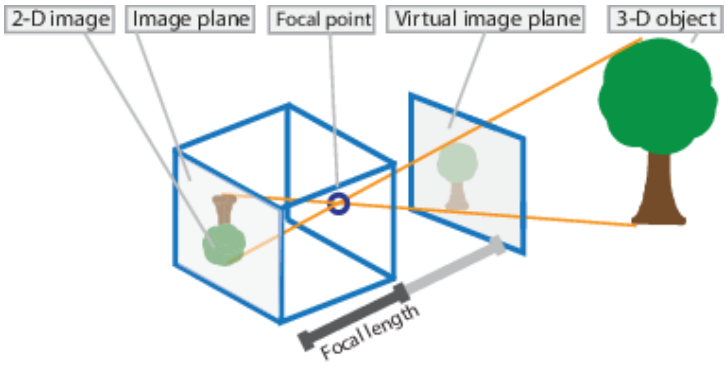


Figure 2: after (*What Is Camera Calibration* 2019)

The intrinsic parameters are usually expressed in the form of camera intrinsic matrix, k, which is defined as:

$$k = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix}$$

where $f_x, f_y$ are the focal length which is the distance between the focal point and the image plane in the dimensions x and y respectively [*see Figure 2*], and $c_x, c_y$ are the optical center, where the optical axes of lens intersect with the image plane in the same x and y dimensions, all expressed in pixels (Szeliski 2011).

The extrinsic parameters denote the transformation from 3D world coordinates to 3D camera coordinates. They are expressed in the rotation matrix $R$ and the translation vector $t$ as follows:

$$[R|t] = \begin{vmatrix} r_1 1 & r_1 2 & r_1 3 & t_1 \\ r_2 1 & r_2 2 & r_2 3 & t_2 \\ r_3 1 & r_3 2 & r_3 3 & t_3 \end{vmatrix}$$

Therefore the transformation from the 3D world coordinates to the 2D image coordinates can be expressed as follows:

$$s \begin{vmatrix} u \\ v \\ 1 \end{vmatrix} = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} r_1 1 & r_1 2 & r_1 3 & t_1 \\ r_2 1 & r_2 2 & r_2 3 & t_2 \\ r_3 1 & r_3 2 & r_3 3 & t_3 \end{vmatrix} \begin{vmatrix} X \\ Y \\ Z \\ 1 \end{vmatrix}$$

where $s$ represents the scale and X,Y,Z are the 3D world coordinates.

### 2.1.2 Feature extraction and matching

Feature extraction and matching is fundamental to photogrammetric reconstruction. Features are points or regions in the image with distinctive and unique properties that allow for easy tracking across several frames. They vary from points such as corners to more complex forms such as edges and blobs. Several researchers have developed methods for feature extraction and matching. Some of the most common are given below:

- **Harris Corner Detection:** Harris et al. (1988) identified a *uniform region* as an area which has no change in intensity in all directions around it, while an *edge* has noticeable change in one direction and a *corner* has changes in all directions. The changes are calculated using Sum of Square Differences (SSD).

- **Good Features to Track** Shi et al. (1994): introduced a change to the scoring function of the Harris Corner Detector by applying a threshold to limit the features detected resulting in better features to track.

- **Scale-Invariant Feature Transform (SIFT)**: Lowe (2004) introduced a robust four-step technique for extracting features. The first step involves calculating the Difference of Gaussian (DoG) for the multiple octaves of the image, and then a scale space function is used to find the local extrema by comparing the point to the eight surrounding neighbors and the nine neighboring pixels in each scale level above and below. Then a filtering of the detected points is preformed to eliminate those that are sensitive to noise. In the next step, orientation invariance is achieved by normalising the direction of gradient variance around the point. Finally, a descriptor for the point is created by dividing the area around the point into 4 x 4 zones as shown in Fig 3. For each zone, an 8 bin histogram of the gradient direction is created. The vector describing the point is summarized to 4 x 4 x 8 = 128 element feature vector. The values in the vector are then normalised to the unit length, then thresholded to no longer than 0.2 and then normalized again to reduce the effect of illumination of the feature detection.

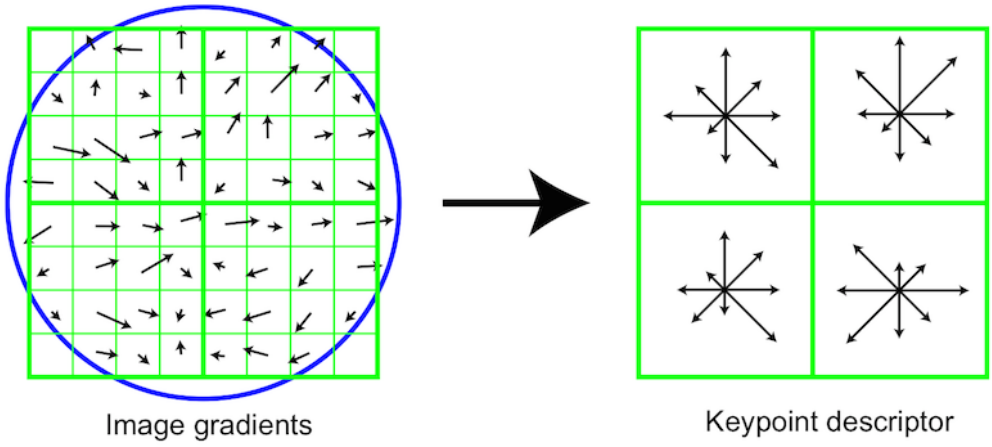


Figure 3: SIFT descriptiors, after (Lowe 2004)

- **Speeded Up Robust Features (SURF)**: Bay et al. (2006) proposed a similar technique to SIFT. Both techniques approximate Laplacian of Gaussian (LoG); however, Lowe (2004) uses DoG while Bay et al. (2006) use a box filter. SURF is more computationally efficient, while SIFT is more robust.

- **Features from Accelerated Segment Test (FAST)**: Rosten & Drummond (2006) developed a corner detector that is considered to be one of the fastest and most computationally efficient detectors currently in use. The detector checks the brightness of the candidate pixel $p$ against a circle of sixteen pixels placed in a circle around $p$, as seen in Fig 4. The pixel $p$ is considered a corner only if all the sixteen are brighter than $I_p + t$, or all darker than $I_p - t$, where $I_p$ is pixel intensity and $t$ is a threshold selected by the user. The algorithm starts by checking only four pixels 1, 9 ,5 and 13, then it checks the rest of the sixteen pixels only if the candidate passed the initial test. One of the main disadvantages of FAST is its sensitivity to noise.

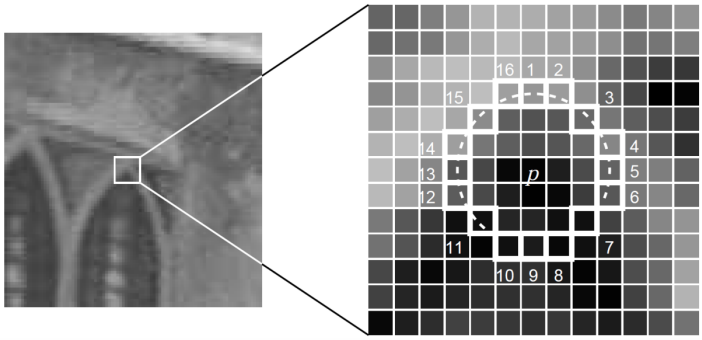


Figure 4: after (Rosten & Drummond 2006)

- **Binary Robust Independent Elementary Features (BRIEF)**: Calonder et al. (2010) introduced an efficient feature descriptor that uses binary strings to define regions around key points. BRIEF does not extract features but only provides description, which means a feature decorator FAST, SIFT, SURF should be used in conjunction with it.

- **Oriented FAST and Rotated BRIEF (ORB)**: Rublee et al. (2011) introduced a fusion between FAST and BRIEF with some enhancements to avoid known weak points of the algorithms. ORB starts by applying FAST to detect initial features, and then uses Harris corner to select the top set. Since FAST is not natively rotation invariant, direction is defined by computing the intensity center of the region surrounding features, and then the vector connecting the feature point to the intensity center is designated as the direction. The features description is then created by steering BRIEF to the orientation of the feature point and then computing the binary string.

### 2.1.3 Visual Odometry

Visual odometry is defined as the process of estimating three dimensional motion in terms of translation and rotation with respect to a reference frame of an agent, by using the input of a single camera or multiple cameras attached to it. The term was promoted by Nister et al. (2004), although it was already in use by Srinivasan et al. (1996). It resembles in concept of wheel odometry which, incrementally estimates the motion of a vehicle by counting the number of revolutions of its wheels over time. Similarly, visual odometry uses cumulative estimation of the location and orientation of the vehicle by exploring the images produced by its onboard cameras and examining changes induced by motion in them. Thus, the problem visual odometry tackles can be considered a special case of Structure From Motion (SfM). SfM tackles the problem of 3-D reconstruction of both the structure and camera poses from sequentially ordered or unordered image sets (Yousif et al. 2015).

Visual odometry requires a sequence of frames that share suitable number and distribution of common features. Additionally, the scene should host sufficient texture and should be relatively stable to enable tracking. Furthermore, ample level of light should be available in the environment for the sensor to capture measurable responses reflected from the objects in the scene Yousif et al. (2015). Visual odometry is susceptible to trajectory drift; therefore it can be assisted by the input of several sensors such as inertial measurement units (IMUs) or Global Navigation Satellite System (GNNS) receivers (Thrun et al. 1999).

This method can be classified into two main types, *stereo* and *monocular*. In stereo visual odometry relative 3D position of the features is directly measured by triangulation at every frame, and is used to derive the relative motion, while in monocular, only bearing information is retrieved. These methods are then further divided into feature matching (matching features over a number of frames) (Talukder et al. n.d.), feature tracking (matching features in adjacent frames) (Dornhege & Kleiner 2006) and optical flow techniques (based on the intensity of all pixels or specific regions in sequential images) (Zhang et al. 2009).

## 2.2 Image quality

Image quality assessment techniques can be classified into *subjective*, which represents the perception of human beings, and *objective*, which uses statistical methods to quantify the image quality. The subjective quality of an image can be measured by aggregation of the individual ratings of perceived quality assigned by human subjects, which can be

represented numerically as the Mean Opinion Score (MOS), or categorically as the Differential Mean Opinion Score (DMOS) (Series 2012). Objective quality assessment can be classified into three methods according to the availability of the original image: a) full-reference, b) reduced-reference, and c) no-reference (Wang et al. 2004). This research will focus on the no-reference objective quality assessment.

The quality of the images used in 3D reconstruction has a major impact on the quality of the developed model. Sets of images with a good resolution and level of detail should lead to a good reconstruction (Bianco et al. 2018). The introduction of noticeable blur caused by camera shake, object motion, or camera focus considerably degrades the performance of traditional feature descriptors when they are used in feature matching and tracking (Wang & Bovik 2005).

Pertuz et al. (2013) compared the performance of thirty six focus measure operators in different levels of noise, saturation, contrast and window size. A group of the tested operators are grouped as Laplacian based operators, they focus on measuring the amount of sharp edges in the image through the second derivative. This group generally performed better than others and therefore, will be considered in this research to estimate the quality of the image Pertuz et al. (2013).

# 3 Materials

The data used in current research comprise five sets of videos captured by UAS and provided by TRIK. TRIK is an enterprise software solution that allows efficient use of drone photography for structural inspection. It automatically turns drone photo feeds into an interactive 3D model, which can be measured and annotated. The 3D model created also doubles as a database allowing the consumer to search for photos, draw additional data, detect structural changes and create maintenance projects directly from TRIK's platform (*TRIK - Drone mapping and 3D reporting software for structural inspection* n.d.).

The video sets range a between four to thirteen videos per set. Every set of video comprises of several shots of an urban building captured by a UAS controlled by a human on the ground. The videos are recorded at 25 frames per second. Each frame has a width of 3840 pixels and a height of 2160 pixels. The provided dataset consists of 37 videos divided into five groups. Each group is addresses a single urban building. The videos were obtained using DJI Inspire 2 UAS with DJI Zenmuse X5S camera. The videos are of varying length and quality. Some of them display the major part of a lap around the target building, while others address a specific part of the building with a close-up shot. Three of the video groups show the target buildings in sections where each video show part of

the building with no overall shot. All the groups include in part of it a shot of the building roof. All the videos are shot in RGB channels. Most of the video frames are sharp while there are a few exhibiting hazy focus or motion blur. The average length of the video is 4571 frames; the maximum is 16355 frames and the minimum is 758 frames. Fig 5 shows the distribution of video length per frame.

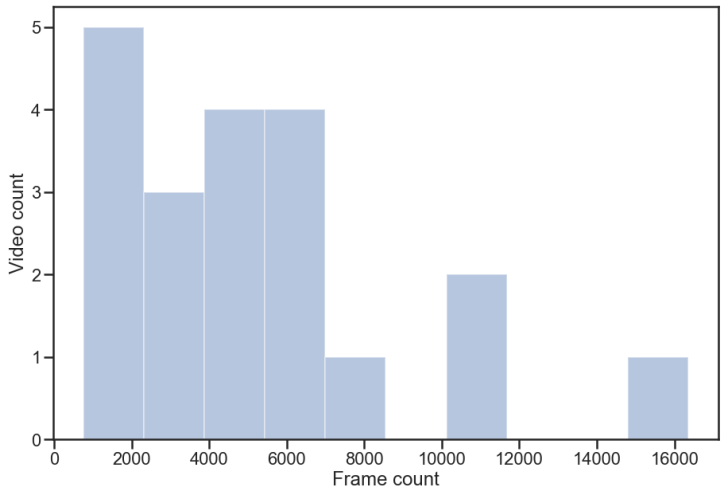


Figure 5: The distribution of video length per frame

Most of the feature matching algorithms process the images in form of two dimensional matrices. Therefore, the first step in processing the data would be to transform the three channel frames (RGB) into gray one channel matrix. Additionally, the size of the frames is 3,840 x 2,160, which means that the feature detection algorithms will have to process 8,294,400 pixels per frame; this is computationally expensive. To reduce this computational cost, the frames are reduced in size. This process will result in loss of some of the features and decrease in accuracy, but it would improve the processing speed. Fig 6 shows the effect of the frame size on the processing time of the frame. To study the relationship between the frame size, time of processing and the accuracy of photogrammetric reconstruction, an experiment of matching two consecutive frames was designed, where the size of the frame is reduced by dividing both the width and the height of the frame by a scalar that is set to increase for every loop. Features are then extracted out of the two frames using the SIFT algorithm that is discussed in 2.1.2, and then the features are then matched using brute force matching algorithm, which is discussed in 4.1.2. Then the matches are filtered to eliminated bad matches, as explained in 4.1.2. The number of good feature matches is used as proxy for reconstruction accuracy.

The figure shows that the number of matches drop gradually while the processing time drops to less than one-third in the first iteration. As the aim of this research is to provide an efficient solution to the problem, the resizing value of 5 was considered to be optimal as it provides a reasonable number of matches, more than 2000, while the processing time is low enough to process at least five pairs of frames per second.

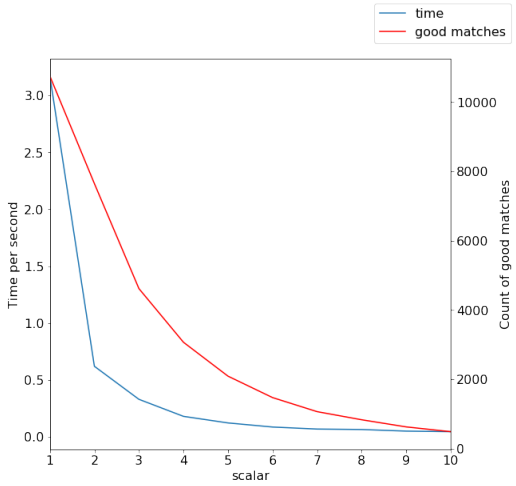


Figure 6: The relationship between resizing the frame, time of processing and number of good matches detected

# 4 Methodology

The methodology suggested for the research is outlined in Fig 7, and can be detailed as follows:

1. **Tracking and initial selection:** In the stage of feature detection and matching, algorithms were used to select and match features in the videos in order to estimate the trajectory of the camera. Upper and lower thresholds of overlap are set to extract sequences of candidate frames.

2. **Image quality assessment and final selection:** A quality value is assigned to every frame of every sequence of frames selected in the previous step, and the highest quality frame is selected out of each sequence.

3. **3D Reconstruction and assessment:** The set of selected frames is used to produce

a 3D model and then an assessment metric is applied to quantify the quality of the reconstruction.

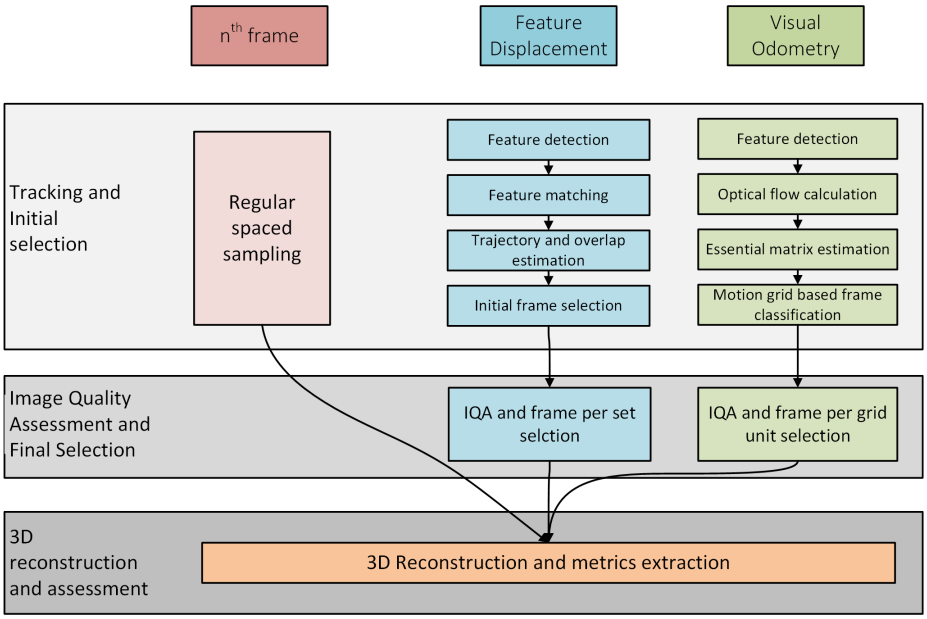


Figure 7: Methodology outline

**A note on programming:** All stages of implementation are conducted using python code. Most of the functions used are implemented in the library opencv. Additional analysis is conducted using numpy, matblotlib, seaborn and pandas. The last stage is implemented using python code that utilizes agisoft metashape python (Application Programming Interface) API. The code is attached in the appendix.

**A note on processing:** All the processing of the first and second stages has been conducted using an Intel Core i7-8750H processor with 32 GB of RAM, and Intel UHD graphics 630 graphics card. The third stage; 3D reconstruction was conducted using an Intel Core i7-5820k processor, with 32 GB of RAM and two NVIDIA GeForce GTX 1080 ti Graphics Cards. Access to the machine and the 3DIMPact lab was provided by the supervisors.

| Algorithm | Average detection time per seconds |
|---|---|
| SIFT | 1.687 |
| SURF | 1.539 |
| Good Features to Track | 0.428 |
| ORB | 0.360 |
| FAST | 0.257 |

Table 1: Average time for feature detection

## 4.1 Tracking and initial selection:

The main concept of the research is to select optimum frames from video for 3D reconstruction. Videos captured using a high resolution camera with high frame rate per second pose a challenge to process. The large computational power required to process the full length of the video would make this option practically unfeasible. Therefore, it is necessary to select a set of frames that would both satisfy the basic requirement of overlap between adjacent images and be small enough for a reasonable processing time.

### 4.1.1 Feature detection

Several algorithms provide the utility of detecting features. Some of them are described in 2.1.2. In Fig 8 the features extracted by five of these algorithms (ORB, SIFT, SURF, Good Features to Track and FAST) are displayed along with the original image. Additionally the average time of feature detection for each of the five algorithms has been calculated using the same parameters and is displayed in Table 1. The defining factors in selecting the matching algorithm include the computation speed and robustness. FAST is typically less computationally intensive and shows acceptable robustness, while both SURF and SIFT are computationally expensive and more robust (Csurka et al. 2018).

### 4.1.2 Feature matching

After detecting features in two images the following step is to match the features across the two images through their descriptors. The basic idea of matching is to search the data-set of features of one image to find features of another image, through matching the descriptors of the target and selected features. A direct method of matching is brute force matching where every feature from the selected image is checked against every feature from the target image until a match is found. Naturally this method is computationally expensive as the number of the computations relates to the multiplication of

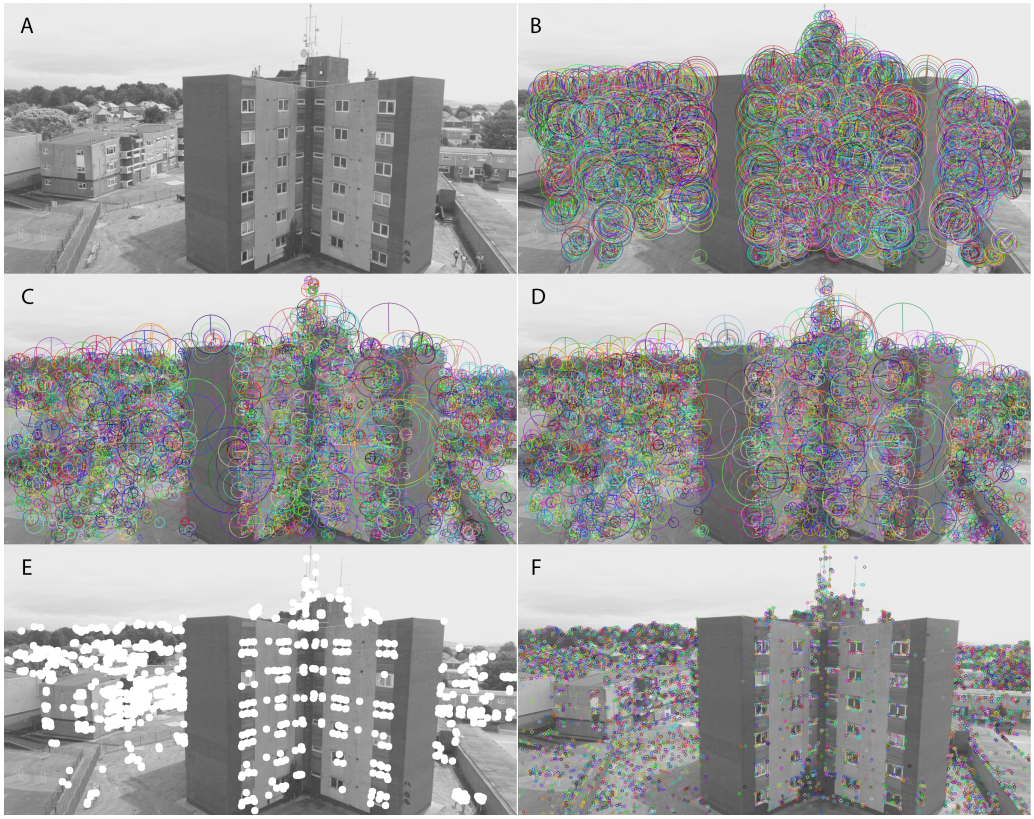


Figure 8: Features extracted by several feature detectors plotted on the original image. A) Original image, B) features extracted by ORB, C) features extracted by SIFT, D) features extracted by SURF, E) features extracted by Good Features to Track, F) features extracted by FAST

the sizes of the two data-sets. Another method of matching is using the Fast Library for Approximate Nearest Neighbors (FLANN) proposed by Muja & Lowe (2009). FLANN is library written in C++ which consists of a collection of algorithms for fast approximate nearest neighbor searches in high dimensional spaces and a system to select from these algorithms and define the optimal search and indexing parameters automatically based on the properties of the data-set (*FLANN Fast Library for Approximate Nearest Neighbors* 2019).

Matching the descriptors is essentially a nearest neighbor search problem. It can be defined as follows : Let $P$ be a set of points in vector space $X$ where $P = (p_1, p_2, ...., p_n)$. Find the nearest neighbors for $q$ in $P$, where $q$ is query point satisfies $q \in X$. To solve this problem, brute force uses a linear search, as it is the most efficient exact search

algorithm for high-dimensional spaces. However, this search can be approximated using more efficient algorithms with minor loss of accuracy.

FLANN uses kd-tree and hierarchical k-means tree algorithms to approximate the nearest neighbor search. In kd-tree the multidimensional space is partitioned using the data points as leaves for the tree as shown in Fig 9. k-means tree is a clustering algorithm where the data points are classified into groups progressively with each level of the tree until a certain threshold of the group count $K$ is reached. The approximation is achieved in both algorithms by limiting the number of data points acting as leave nodes. The number of points is determined through a user defined variable of search precision which is used by FLANN to define the number of nodes at which the search would stop (Muja & Lowe 2009).

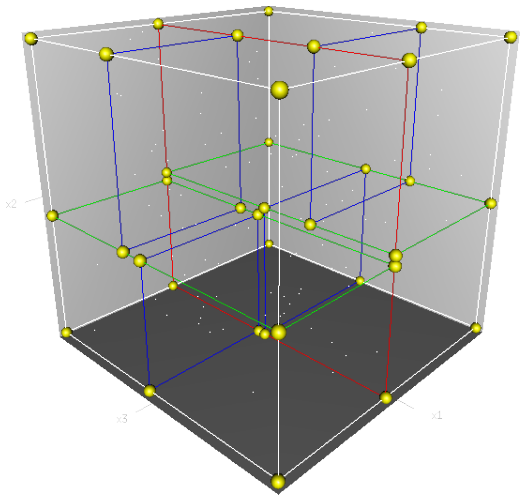

Figure 9: kd-tree in 3D space, after (Wikipedia 2012)

Optimizing the feature matching against false matches of points that have no similar match in the data-set can be difficult. As the matching itself is defined by the minimum Euclidean distance between the target and the query point, some of the query points will eventually for a false match with target points from the data-set. Lowe (2004) suggested a method of identifying the false matches by comparing the Euclidean distance of the closest neighbor to that of the second-closest neighbor. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching. For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space (Lowe 2004). Fig shows the result of analysis which Lowe (2004).

The Probability Density Functions (PDF) for correct and incorrect matches are shown in terms of the ratio of closest to second-closest neighbors of each keypoint. Matches for which the nearest neighbor was a correct match have a PDF that is centered at a much lower ratio than that for incorrect. Lowe (2004) suggested a to reject all the matches with distance ratio higher than 0.8 which eliminates 90% an false matches and only 5% of the correct matches.

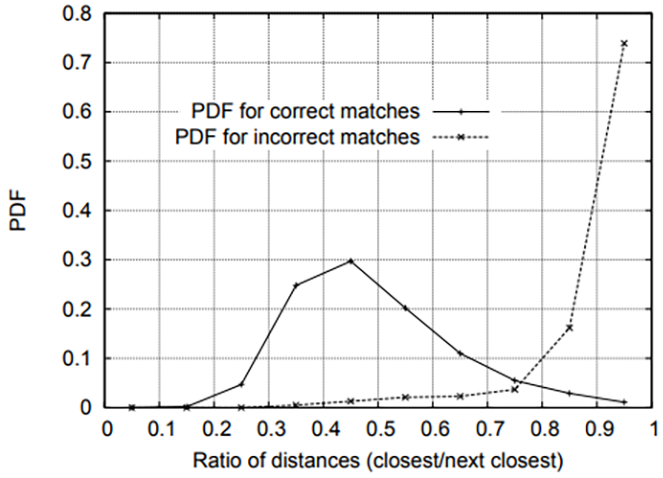


Figure 10: The relationship between the probability of correct and false matches with the ratio of distance from the closest neighbor to the distance of the second closest and the after (Lowe 2004)

Additionally, this research uses an implementation of Locality Sensitive Hashing (LSH) designed by Lv et al. (2007) for indexing the data-set. This implementation uses multi-probe LSH which improves over the entropy based LSH, method by reducing the number of hash tables while preserving the efficiency in terms of time.

#### 4.1.3 Features displacement based approach

The idea of selecting optimum frames revolves around estimating the motion of the camera, which by consequence results in change in the images captured and the features displayed in the images. In this section features are detected and matched between frames and then the displacement of the features is estimated across the 2D plane of the image itself. Every frame is then designated a value to represent the displacement of the frame as a whole. Finally a lower and upper threshold of the overlap with the initial frame is set, and sequence of frames within this range are selected, and then the initial frame is reset

and the process is repeated again.

To implement this approach feature detection and matching algorithms should be selected. In this research SIFT, and ORB are used. SIFT provides the highest accuracy of detection in the set of available feature detection algorithms while being the most computationally expensive. ORB in the other hand is slightly less accurate while providing a faster processing. For the feature matching FLANN is used with ORB, while brute force is used with SIFT. The aim of this combination is test the limits of accuracy and speed. Using SIFT with brute force should provide the most accurate solution, although it is expected to be slower, while using ORB with FLANN should provide the fastest solution with a compromise in terms of accuracy. Multiple combinations are possible to implement, however the selected two should together provide some insights into the effects of speed and accuracy on frame selection.

The algorithm starts by viewing the first frame in the video, which is considered to be the initial frame in this first loop. The second step is to reduce the frame size to the optimal values and reducing the RBG image to gray scale image as discussed in 3. This step will be conducted for all the frames of the video as they are read. The The third step is to extract the features and the descriptors attached to them using either ORB or SIFT depending on the implementation. The algorithm then ends the loop and starts a new one. In the second loop the algorithm starts by viewing the second frame of the video, and then copies the features and descriptors extracted from the previous loop to new variables to represent the previous frame. Then the same process of feature detection and descriptors generation is performed on the second frame. Following this, the descriptors of the features extracted from the previous and current frame are matched using either brute force or FLANN. The produced matches are then filtered to eliminate incorrect matches using the same method explained in 4.1.2.

Once the incorrect matches are filtered out, an internal loop starts to check all the matches and identify the location of the features matched together in the current and previous frame. The feature locations is retrieved in 2D image coordinates and then the liner shift between the two feature locations is calculated in its $x$ and $y$ components. This process is repeated for all the matches found and the values of the shift are stored in lists for each component. The following step is to represent the shift of the current frame as a whole, from the position at which was the previous frame. The mean of the list of shift was initially considered but then it was found that it provides unrealistic results due to the distortion by outliers. Therefore the median was selected to represent the shift of the frame. This loop results in extracting the shift of the image in terms of image pixels in both the $x$ and $y$ components separately. The values extracted from the frame are then

added to accumulators which represent the shift of the frames starting from the initial frame till the current processed frame.

The initial frame selection involves extracting a set of candidate frames, which will be then processed by the second stage. The selection of the set of frames is controlled by the overlap ratio between the initial frame and the candidate frame. The overlap between the two frames is calculate in the current implementation using a the shift of the pixels as in the following equation

$$Overlap\,pixels\,count = (Frame\ width - |Shift_X|) * (Frame\ height - |Shift_y|)$$

where $Shift_x$ and $Shift_y$ is the accumulated shift in the $x$ and $y$ direction respectively. The resulted number is then divided by the frame size (count of the frame pixels) to produce a ratio of overlap.

An internal checker is implemented to check the ratio of overlap against upper and lower limit. The checker starts with an empty list of frames and another empty list of sets of frames. Once the upper limit of overlap is reached the checker records the frame ID to the empty list of frames and then initiates a new loop of the algorithm. If the overlap value of the new frame was found to be within the overlap limits then the new frame ID is added to the current list of frames and a new loop of the algorithm is initiated. This process is repeated until the overlap limits are exceeded. Once this occurs, the algorithm will stop adding frames to the list of frames and will add the list of frames itself to the list of sets. Then a new initial frame is assigned and the both $x$ and $y$ shift accumulators are rest to null. This process is repeated until all the video frames are processed. The end result of this process is a list of sets of frames that fall within the limits of overlap with their respective initial frames. This list of sets will be processed in the second stage to produce the final list of frames for 3D reconstruction.

This approach assumes that the camera is moving towards a certain roughly stable direction within 2D plane. As the overlap is expressed by the subtracting the shift of the frame size it uses a single value for the whole frame it is not robust against either tilting or camera movement in the $z$ direction (perpendicular to the assumed plane of movement).

#### 4.1.4 Visual odometry based approach

Selecting frames based on visual odometry depends on the estimation of the movement of the camera and then classifying this recorded movement into segments. Frames within each segment are tested for their quality and then the best frame is selected. The end product is a list of frames which represents the movement of the camera in regular spatial intervals. The steps of the suggested methodology can be summarized a follows:

1. **Feature extraction**
2. **Optical flow calculation**
3. **Essential matrix calculation and decomposition**
4. **Accumulation and plotting of camera movement**
5. **Frame spatial classification**

The code used in this approach is modified after (*uoip/monoVO-python: A simple monocular visual odometry project in Python* n.d.).

**Feature extraction** The features displacement based approach in section 4.1.3 compared two feature selection and description algorithms (ORB and SIFT). Although SIFT is considered to provide the highest accuracy with the downside of being the slowest process, it did not provide a significant advantage in terms of accuracy over ORB as shown in section 6. ORB consists of a fusion between a feature detector; FAST and feature description algorithm; BRIEF as explained in section 2.1.2. Since FAST provided the shortest time in the tested feature detectors as in table 1 and the second stage (Optical Flow) does not require feature description, FAST was chosen as the feature detection algorithm in this implementation.

**Optical flow calculation** Optical flow can be defined as the apparent movement of the objects in the images as a result of the camera or the objects movement (Bouguet 2000). The flow is typically estimated between two consecutive frames. Consider a pixel $I$ at $(x, y, t)$ where $x, y$ are image coordinates and $t$ is the time of the initial frame. The movement of the object represented by this pixel in the frame with time difference of $dt$ would be $(dx, dy)$. Assuming that the movement is small and $I$ do not change intensity between the two frames then

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

then with Taylor series the equation can be written as

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\delta I}{\delta x}dx + \frac{\delta I}{\delta y}dy + \frac{\delta I}{\delta t}dt$$

then

$$\frac{\delta I}{\delta x}dx + \frac{\delta I}{\delta y}dy + \frac{\delta I}{\delta t}dt = 0$$

then dividing by $dt$ we get

$$\frac{\delta I}{\delta x}\frac{dx}{dt} + \frac{\delta I}{\delta y}\frac{dy}{dt} + \frac{\delta I}{\delta t}\frac{dt}{dt} = 0$$

which can be written as

$$\frac{\delta I}{\delta x}V_x + \frac{\delta I}{\delta y}V_y + \frac{\delta I}{\delta t} = 0$$

where $\frac{\delta I}{\delta x}$ and $\frac{\delta I}{\delta y}$ are image gradients in both $x, y$ directions. Similarly, $V_x = \frac{dy}{dt}$ and $V_y = \frac{dx}{dt}$ which are optical flow components in $x, y$ directions respectively. Since both the components are unknown, the equation cannot be solved as such. Several solutions were suggested to these optical flow equations by introducing additional equations and new constraints (Lucas & Kanade 1981, Horn & Schunck 1981, Humphreys & Bruce 1989, Beauchemin & Barron 1995, Glocker et al. 2008).

Lucas & Kanade (1981) suggested a solution based on linear approximation where the area around the tracked pixel is assumed to share the same movement. If a kernel of 3 x 3 pixels around the tracked pixel is considered, the equations of local image flow can be written as follows:

$$\begin{aligned}
I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\
I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\
I_x(q_3)V_x + I_y(q_3)V_y &= -I_t(q_3) \\
&\vdots \\
I_x(q_9)V_x + I_y(q_9)V_y &= -I_t(q_9)
\end{aligned}$$

where $q_1, q_2, ....q_9$ are the pixels in the kernel and and $I_x, I_y, I_t$ are the partial derivatives with respect to $x, y, t$ respectively. These equations can be summarized in three matrices where $A$ is positional matrix of $(I_x(q_i)V_x, I_y(q_i)V_y)$, velocity matrix of $(V_x, V_y)$ is $v$, and $b$ is time matrix of $-I_t(q_i)$. The result of using the kernel is an over-determined state of two unknowns and nine equations which can be resolved using the least squares principle resulting in the following equation:

$$\begin{vmatrix} V_x \\ V_y \end{vmatrix} = \begin{vmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{vmatrix}^{-1} \begin{vmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{vmatrix}$$

Additionally, to limit the effect of small movement assumption, the current implementation of the algorithm is using pyramids of the original image. Pyramids are reduced

copies of the original image produced to provide efficient solution for several computer vision processes (Bouguet 2000).

This research uses the list of features produced by the first step to estimate the optical flow of every frame using Lucas-Kanade method. Firstly a list of features are produced from the initial frame and then tracked in the second frame to extract a list of optical flow vectors that will be used in the following step to calculate the essential matrix. For each following frame the list of features is checked to determine the number of still visible features and in case the count drops under a specific threshold, and then the feature detection algorithm is initiated again to repopulate the list of features again. This check is implemented to increase the efficiency of the algorithm by extracting the features only when they are required.

**Essential matrix calculation and decomposition** Estimating the relative position shift of the camera between two consecutive images can be accomplished through examining the positions of corresponding features within the two frames. If the initial camera position and orientation is known, then the new position and orientation can be calculated using the rotation matrix $R$, and the translation vector $t$. These two terms together constitutes the Essential matrix. The rotation matrix is 3 X 3 matrix which represents the change in orientation while the translation vector has a length of 3 and represents relative shift of the camera's position. The relationship between the essential matrix $E$ and both the rotation matrix and the translation vector $t$ can be written as follows:

$$E = [t]_x R$$

Estimation of the essential matrix can be accomplished using a set of corresponding points and applying total least squares method. This approach as known as Eight-point algorithm (Longuet-Higgins 1987). It uses a set of eight corresponding points to estimate the essential matrix. Several estimation methods were suggested as improvements to this approach. Nistér (2004) suggested a solution based on five points correspondence. To further optimize the approach, Random Sample Consensus Scheme (RANSAC) is used. RANSAC is an iterative process of randomly selecting a subset of the matched features to perform the five point algorithm on and then checking the consistency of the provided solution across the matched features space. Then the process is repeated with a different set of randomly selected features. Finally the solution that provides the least error is selected. This additional step using RANSAC is performed to minimize the effect of outliers on the solution Nistér (2004).

In the current implementation the optical flow vectors are used to find corresponding

points in every two consecutive frames. The list of corresponding points is then presented to the essential matrix estimation algorithm. The estimated essential matrix is decomposed and both the rotation matrix and the translation vector are extracted.

**Accumulation and plotting of camera movement** The extracted rotation matrix and translation vector are calculated in relative terms. In a pair of frames, the rotation of the second frame is calculated relative to the orientation of the first frame and not the orientation of the absolute 3D world axes. Similarly, the translation vector is calculated with regards to the orientation of the camera in the first frame and no regard to the scale, as it is unknown.

Extracting the rotation matrix with regards to the original axes direction is done using the following equation:

$$R_{absolute} = R_{initial}.R_{current}$$

where $R_{absolute}$ is the orientation of the second frame with regard to the absolute axes orientation, $R_{initial}$ is the orientation of the second frame with regard to the absolute axes, and $R_{current}$ is the estimated rotation matrix with regard to the first frame orientation. The translation vector $t_{absolute}$ is calculated using a similar method of calculating the dot product with of the estimate translation vector $t_{current}$ with the rotation matrix of the first frame $R_{initial}$.

$$t_{absolute} = R_{initial}.t_{current}$$

Once this step is achieved the translation vector is decomposed to its three components of $x, y, z$. Plots of the translation of the camera are initially generated from the first pair of frames and then updated with the following pairs. The initial implementation included plotting both $(x, y)$ and $(x, z)$ movement, however only $(x, z)$ was used due to the nature of movement of the camera in the videos used. By examining the videos as in section 3 it was found that the camera is not moving in vertical manner, but rather in horizontal directions across the facade or around a building.

**Frame spatial classification** The resulting plot of the $(x, z)$ is produced in values of frame pixels. Every point in the plot represents the estimated position of the camera in a frame. To classify the frames spatially, each frame is assigned a location and then classified according to a grid. The location is calculated by accumulating the translation vectors of each frame. The second step is to overlay a grid of predefined cell size over the plot, where every grid cell would contain a set of frames assigned spatially according to their location as shown in fig 11. This grid serves as a classification method, as it classifies the frames according to their locations within the grid itself. The camera movement

is traced in the coloured line, where the start of the movement is colored green and gradually changes to red as the time progresses. The grid has a cell size of 3 X 3 pixels. The grid is constructed in square shaped cells despite the frame being rectangular in shape because the classification is based more on the movement of camera itself.

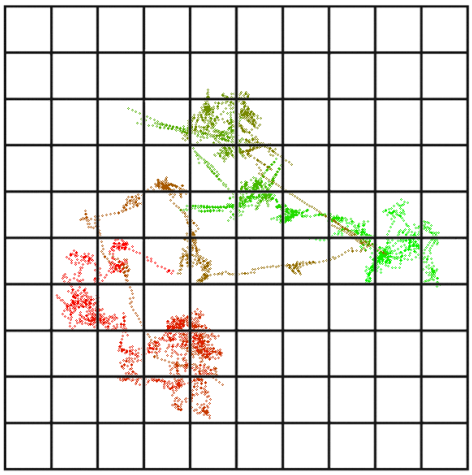

Figure 11: Spatially classifying the camera movement according to a grid

The result of this step is a list of frames per each grid cell which will be presented to the image quality assessment algorithm to produce a final list for reconstruction.

## 4.2 Image quality assessment and final selection

The second stage of the frame selection is to estimate the quality of each frame and select the best quality within each set to present it to the 3D reconstruction stage. As shown in section 2.2, image sharpness is used as the measure for image quality in this research.

A good measure for image sharpness is the variance of Laplacian of an image (Pertuz et al. 2013). The Laplacian filter is a conventional representation for the Laplacian operator, which is a differential operator given by the divergence of the gradient of a function on Euclidean space. In image space, this means that the filter will pronounce the areas where a rapid gradient change is occurring, which is common where edges are located. The calculation is conducted using a conventional filter, which can be represented by a window sliding across the image, and producing a value which depends on the intensity of the pixels covered by the window, and the values predefined in the filter itself.

Convolution filter works by adding each pixels of the image to its local neighbors weighted by the kernel and then dividing by the count of pixels. The kernel size and values define the behaviour of the filter. The size of the kernel is defined by the value $k$ which represents the number of lines around the central pixel, for example a kernel with $k = 1$ would be a 3 x 3 window, with a total of 9 pixels, while a kernel with $k = 3$ would be 5 x 5 window with a count of 25 pixels. This means that the larger the kernel, the greater the processing time, and the larger the count of pixels around the central pixel which will have impact on final result of the convolution. The value of the kernel pixels contributes to the result as well. Identity kernel for example -shown in Fig 12 - returns the same image as the input.

$$Kernel = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

Figure 12: Identity Kernel

Laplacian operator works by producing the second derivative of the image. It can be approximated using the kernel shown in figure 13. The filter results in an image where the edges are pronounced. Variance of Laplacian measures the distribution of values

$$Kernel = \begin{vmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{vmatrix}$$

Figure 13: Laplacian Kernel

returned from Laplacian filter. The higher the amount of edges in an image, the higher the value of the variance and vice versa. This can serve as a representation of the image quality as the images with better defined, clear edges are most likely to have features which are easier to track and in accurate locations, which will impact the quality of the 3D reconstruction.

The variance of Laplacian can be represented in the following equation:

$$Lap\_var(I) = \sum_{m}^{M} \sum_{n}^{N} [|L(m,n)| - \hat{L}]$$

where

$$\hat{L} = \frac{1}{NM} \sum_{m}^{M} \sum_{n}^{N} |L(m,n)|$$

$Lap\_var$ is the variance of Laplacian, $I$ is the image, $M$ and $N$ are the image dimensions, and $L$ is the Laplacian filter.

Extracting the highest quality frames from the list produced by the first stage of frame selection was a straight forward process. A function of the variance of Laplacian was used to assign a value for each candidate frame and then the values are compared within each set to elect the highest that would represent the best quality for the 3D reconstruction. The final result of the two stages is a list of frames per video to present to the 3D reconstruction algorithm.

## 4.3 3D reconstruction and assessment

Building a model out of the selected frames is the ultimate goal of this research. This steps consumes the most time and computational resources due to its complexity. Several software packages are available to produce a 3D model out of images. In this research *Agisoft Metashape* was used. The models were fed to the software through a custom python code, which also extracted the evaluation metrics after building the model. Two stages of processing were required to build the model; extracting and matching the points, and then aligning the cameras and building sparse point cloud.

## 4.4 Baseline

To evaluate the performance of the frame selection algorithms, a baseline has to be established. An additional frame selection method has to be designed to benchmark the results of evaluation metrics for the suggested methodologies against. Regular interval frame selection would serve as a baseline in this research. Videos are captured in 25 frames per second as mentioned in section 3. Therefore the baseline is established as one frame per second. Once the frames were selected they were passed through the same reconstruction procedures as the two methodologies and metrics where extracted. In total 136 model were produced, 34 per each methodology.

## 4.5 Experiments

In conclusion, the experiments conducted follow the outline of the methodology shown in Fig 7, All the experiments start with frame selection and then 3D reconstruction. They

are organised in terms of selection method as follows:

- **Experiment I**: Features displacement using SIFT and brute force
- **Experiment II**: Features displacement using SIFT and brute force
- **Experiment III**: Visual Odometry
- **Experiment IV**: Baseline regular temporal interval

# 5 Results

Results were collected in the three stages of methodology mentioned in section 4. Here the results will be listed as raw data, and then graphs will be presented to summaries the results in a visual manner. The graphs are presented in two parts; the first stage: the selection process, which includes the initial and final selection and the second stage, which includes the 3D reconstruction process. Most of the analysis is conducted based on the second stage, as its results are more impactful for the purpose of the research.

## 5.1 Raw results

: In the following tables the headers are annotated as follows:

- **M**: Model number
- **IC**: Initial frame selection count
- **FC**: Final frame selection count
- **ST**: Selection time per second
- **RMSE**: RMSE in pixels
- **3DT**: 3D construction time in seconds

### 5.1.1 Experiment I

Results are shown in table 2

### 5.1.2 Experiment II

Results are shown in table 3

Table 2: Results of experiment I

| M | IC | FC | ST | RMSE | 3DT |
|---|---|---|---|---|---|
| 0 | 3794 | 35 | 2204.810438 | 0.469348154 | 31.091 |
| 1 | 1322 | 14 | 763.8734586 | 0.369766436 | 11.624 |
| 2 | 1307 | 32 | 549.9111626 | 0.832856106 | 65.37 |
| 3 | 2539 | 91 | 1571.854882 | 1.072886439 | 175.379 |
| 4 | 2632 | 77 | 1596.180674 | 1.97667193 | 155.036 |
| 5 | 1028 | 34 | 528.5841966 | 1.078884404 | 35.309 |
| 6 | 1288 | 81 | 973.7087777 | 0.979335704 | 183.159 |
| 7 | 431 | 18 | 268.2669909 | 0.677991746 | 13.102 |
| 8 | 1003 | 113 | 689.2272189 | 1.259409212 | 259.59 |
| 9 | 969 | 97 | 596.9551911 | 0.995437275 | 203.174 |
| 10 | 1144 | 152 | 860.4321976 | 1.366044946 | 372.569 |
| 11 | 925 | 66 | 1871.466571 | 1.217360849 | 192.985 |
| 12 | 883 | 93 | 609.360991 | 1.518672685 | 174.753 |
| 13 | 342 | 17 | 178.362628 | 0.637725889 | 17.342 |
| 14 | 180 | 18 | 137.1538548 | 0.762392961 | 24.108 |
| 15 | 207 | 17 | 134.7942116 | 0.727876927 | 22.608 |
| 16 | 1903 | 154 | 1349.188071 | 1.48926708 | 330.915 |
| 17 | 521 | 43 | 347.8898401 | 1.155420936 | 67.277 |
| 18 | 1536 | 186 | 993.844883 | 1.082180614 | 467.316 |
| 19 | 792 | 84 | 469.0325744 | 1.387068788 | 152.911 |
| 20 | 861 | 67 | 501.8410707 | 1.123578303 | 190.876 |
| 21 | 1491 | 169 | 908.6693244 | 1.274615737 | 502.823 |
| 22 | 1297 | 159 | 891.8828392 | 1.221847319 | 505.214 |
| 23 | 1226 | 167 | 660.8921103 | 1.553991599 | 635.407 |
| 24 | 637 | 48 | 344.5258851 | 1.313013317 | 74.995 |
| 25 | 690 | 33 | 302.0446014 | 0.860033814 | 37.247 |
| 26 | 479 | 44 | 216.7628291 | 1.01773818 | 88.32 |
| 27 | 587 | 41 | 255.9699228 | 0.937621563 | 66.48 |
| 28 | 693 | 73 | 377.7189834 | 1.185142056 | 150.896 |
| 29 | 163 | 22 | 93.32226324 | 0.960539009 | 30.576 |
| 30 | 289 | 47 | 135.8174739 | 0.963773911 | 98.556 |
| 31 | 1497 | 148 | 920.7335899 | 0.921672814 | 332.118 |
| 32 | 1952 | 190 | 1027.872786 | 1.4798467 | 529.055 |
| 33 | 410 | 34 | 226.3823059 | 0.909100283 | 54.419 |

Table 3: Results of experiment II

| M | IC | FC | ST | RMSE | 3DT |
|---|---|---|---|---|---|
| 0 | 3749 | 12 | 1522.892436 | 0.399206892 | 9.437 |
| 1 | 1272 | 8 | 476.8708436 | 0.92986638 | 6.14 |
| 2 | 1434 | 22 | 547.0484703 | 0.752930976 | 40.856 |
| 3 | 1550 | 63 | 964.8385553 | 1.079736492 | 98.93 |
| 4 | 2392 | 51 | 1058.99251 | 3.844032076 | 83.665 |
| 5 | 662 | 22 | 345.0923278 | 0.871816469 | 29.327 |
| 6 | 1304 | 63 | 594.7191813 | 0.918960964 | 111.711 |
| 7 | 377 | 9 | 188.6769884 | 1.052192013 | 6.031 |
| 8 | 995 | 115 | 446.8888631 | 1.267140178 | 268.888 |
| 9 | 908 | 91 | 374.1804707 | 1.020882277 | 186.596 |
| 10 | 1238 | 148 | 528.6769485 | 1.33579727 | 432.197 |
| 11 | 941 | 65 | 353.918705 | 1.316831866 | 196.813 |
| 12 | 997 | 91 | 412.5577509 | 1.597714651 | 177.83 |
| 13 | 330 | 16 | 162.0860744 | 0.68029495 | 18.311 |
| 14 | 200 | 19 | 87.58556986 | 0.762121563 | 27.608 |
| 15 | 191 | 18 | 108.8566761 | 0.777719835 | 26.998 |
| 16 | 1893 | 153 | 2170.007775 | 1.29715651 | 366.474 |
| 17 | 519 | 38 | 184.2728052 | 1.159452412 | 57.995 |
| 18 | 1657 | 182 | 661.6171191 | 1.08305611 | 530.477 |
| 19 | 731 | 83 | 315.8605289 | 1.328910499 | 156.88 |
| 20 | 809 | 68 | 341.9919336 | 1.075698827 | 159.505 |
| 21 | 1452 | 165 | 637.3989964 | 1.416321627 | 433.312 |
| 22 | 1268 | 156 | 518.3256543 | 1.245192342 | 414.456 |
| 23 | 1242 | 165 | 528.4316056 | 1.697988023 | 640.359 |
| 24 | 823 | 50 | 324.0326965 | 1.261604862 | 85.229 |
| 25 | 542 | 32 | 238.7649536 | 0.906887003 | 33.795 |
| 26 | 492 | 42 | 169.4953303 | 1.025574044 | 84.915 |
| 27 | 735 | 38 | 251.4898357 | 0.935540553 | 52.246 |
| 28 | 760 | 70 | 276.4254179 | 1.192531259 | 125.257 |
| 29 | 175 | 16 | 82.64479303 | 0.832624054 | 19.828 |
| 30 | 285 | 46 | 136.1156747 | 0.945598991 | 99.087 |
| 31 | 1634 | 140 | 664.36573 | 1.043965264 | 352.693 |
| 32 | 1881 | 195 | 745.7184796 | 1.252474645 | 537.198 |
| 33 | 391 | 34 | 184.9459748 | 0.909839806 | 55.043 |

#### 5.1.3 Experiment III

Results are shown in table 4

#### 5.1.4 Experiment IV

Results are shown in table 5

### 5.2 The selection process

Four selection processes were conducted, two for the features displacement based approach, where two different feature detection and two feature matching algorithms were used, one for the visual odometry based approach and one for the baseline. In the following graphs, each selection process is referred to by a short hand as follows:

- Feature displacement based approach
  - **ORB** refers to the approach using ORB feature detector and FLANN feature matching algorithm
  - **SIFT** refers to the approach using SIFT feature detector and brute force feature matching algorithm
- **VO** refers to the visual odometry based approach
- **25bl** refers to the baseline

This annotation will continue to be constant through out the following sections.

The selection of the baseline was conducted in a single process, while the frame count of the initial selection of the visual odomerty approach is equal to the frame count of the original video due to the nature of the process. The initial selection process in this case resulted in assigning location values for each frame. Therefore the results of the initial selection process are limited to the displacement based approach. Fig 14 shows the distribution of the frame count across the produced models.

The final outcome of frame selection process results in two metrics; the final frame count shown in Fig 15 and the time required for frame selection shown in Fig 16.

### 5.3 The reconstruction process

The second stage of processing is to build the 3D model and check its quality. The two metrics collected in this stage are the processing time of building 3D model shown in

Table 4: Results of experiment III

| M | FC | ST | RMSE | 3DT |
|---|---|---|---|---|
| 0 | 12 | 1415.524 | 0.665233 | 370.265 |
| 1 | 8 | 601.9401 | 4.727103 | 319.702 |
| 2 | 22 | 583.4703 | 0.812812 | 723.263 |
| 3 | 63 | 1059.204 | 1.050505 | 554.15 |
| 4 | 51 | 1346.301 | 0.896075 | 405.281 |
| 5 | 22 | 460.7352 | 0.933952 | 212.09 |
| 6 | 63 | 789.9413 | 0.865885 | 232.39 |
| 7 | 9 | 380.195 | 0.74161 | 43.904 |
| 8 | 115 | 530.1371 | 1.009746 | 146.735 |
| 9 | 91 | 480.2716 | 0.876913 | 94.782 |
| 10 | 148 | 609.4855 | 0.908768 | 150.066 |
| 11 | 65 | 494.3099 | 0.882699 | 197.347 |
| 12 | 91 | 457.6496 | 0.879803 | 156.362 |
| 13 | 16 | 169.3009 | 0.600288 | 24.498 |
| 14 | 19 | 152.2515 | 0.644676 | 16.318 |
| 15 | 18 | 152.0191 | 0.522086 | 25.965 |
| 16 | 153 | 1067.211 | 0.96251 | 345.397 |
| 17 | 38 | 311.3018 | 0.951139 | 86.617 |
| 18 | 182 | 752.108 | 0.720933 | 171.683 |
| 19 | 83 | 375.6617 | 0.902127 | 62.76 |
| 20 | 68 | 441.9334 | 0.772145 | 44.59 |
| 21 | 165 | 727.841 | 0.633781 | 89.232 |
| 22 | 156 | 770.9184 | 0.761945 | 119.761 |
| 23 | 165 | 634.9726 | 1.094538 | 133.029 |
| 24 | 50 | 448.4155 | 0.893109 | 56.362 |
| 25 | 32 | 393.6395 | 0.760137 | 41.716 |
| 26 | 42 | 262.732 | 0.787965 | 96.4 |
| 27 | 38 | 283.485 | 0.789761 | 89.128 |
| 28 | 70 | 375.0798 | 0.769282 | 59.525 |
| 29 | 16 | 110.8536 | 0.685956 | 13.324 |
| 30 | 46 | 149.3294 | 0.626964 | 24.418 |
| 31 | 140 | 968.4221 | 0.926024 | 191.676 |
| 32 | 195 | 986.5528 | 0.824877 | 160.696 |
| 33 | 34 | 409.2698 | 0.727451 | 60.139 |

Table 5: Results of experiment IV

| M | FC | ST | RMSE | 3DT |
|---|---|---|---|---|
| 0 | 654 | 655.3436 | 0.844023 | 6756.582 |
| 1 | 222 | 238.9063 | 0.817088 | 971.822 |
| 2 | 228 | 236.7802 | 0.982941 | 2882.512 |
| 3 | 430 | 478.4447 | 1.138817 | 3651.772 |
| 4 | 440 | 497.7239 | 1.063152 | 3241.205 |
| 5 | 155 | 176.8125 | 1.04498 | 458.069 |
| 6 | 244 | 283.8054 | 1.147954 | 1289.422 |
| 7 | 70 | 84.86067 | 0.943003 | 137.882 |
| 8 | 177 | 199.1821 | 1.502362 | 633.999 |
| 9 | 154 | 171.1294 | 1.294955 | 435.438 |
| 10 | 189 | 209.4976 | 1.49076 | 586.254 |
| 11 | 144 | 164.6527 | 0.967158 | 865.717 |
| 12 | 158 | 182.3293 | 0.982606 | 413.508 |
| 13 | 59 | 65.07438 | 0.888321 | 118.111 |
| 14 | 30 | 35.52643 | 0.960927 | 48.231 |
| 15 | 36 | 41.10601 | 0.964931 | 69.503 |
| 16 | 327 | 369.7578 | 1.272342 | 1437.973 |
| 17 | 83 | 90.97345 | 1.415765 | 216.247 |
| 18 | 265 | 294.9695 | 1.092082 | 652.088 |
| 19 | 128 | 144.6425 | 1.444597 | 329.063 |
| 20 | 134 | 154.5263 | 1.11818 | 556.578 |
| 21 | 236 | 255.7179 | 1.166825 | 695.898 |
| 22 | 217 | 244.05 | 1.326022 | 795.396 |
| 23 | 203 | 223.6443 | 1.399941 | 761.347 |
| 24 | 127 | 137.7184 | 1.035487 | 398.892 |
| 25 | 106 | 112.5986 | 1.092152 | 224.794 |
| 26 | 82 | 87.97258 | 0.928797 | 267.16 |
| 27 | 102 | 107.8285 | 1.054253 | 340.457 |
| 28 | 118 | 130.0769 | 1.202379 | 293.137 |
| 29 | 31 | 34.57043 | 0.938232 | 47.045 |
| 30 | 47 | 53.06896 | 1.02839 | 99.706 |
| 31 | 277 | 298.7459 | 1.207918 | 954.797 |
| 32 | 295 | 316.7193 | 1.258668 | 1022.304 |
| 33 | 68 | 62.21946 | 0.928862 | 182.234 |

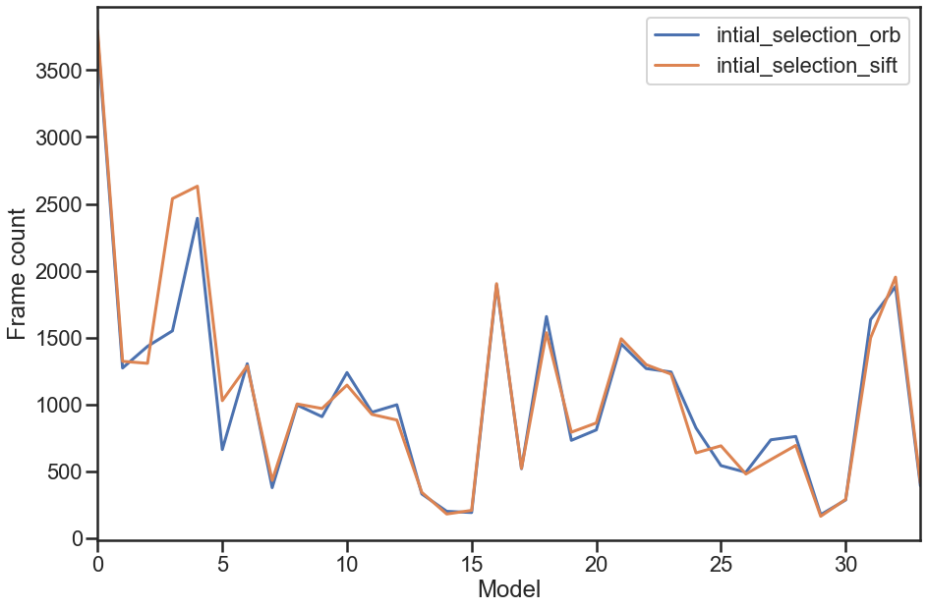


Figure 14: Initial frame selection count

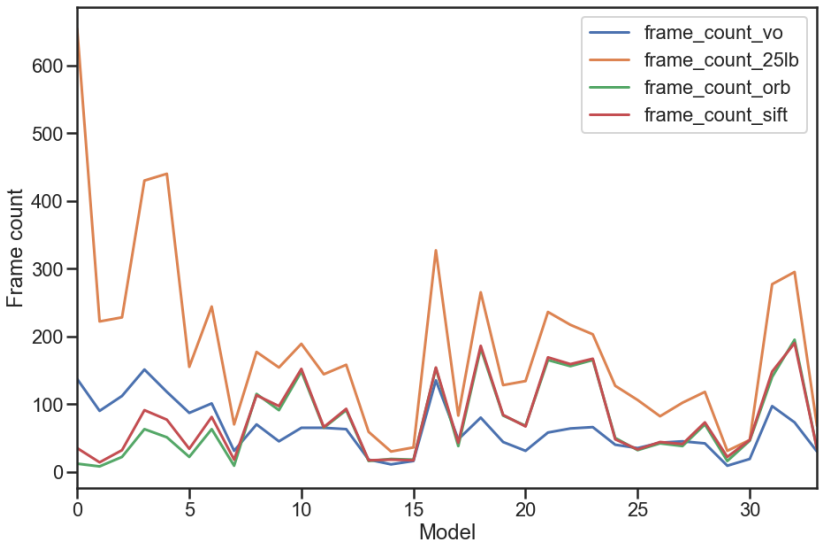


Figure 15: Final frame selection count

Fig 18 and the accuracy of the model presented in Root Mean Square reprojection Error (RMSE) shown in Fig 17.

For each point in the point cloud, a location on at least two images should be found.

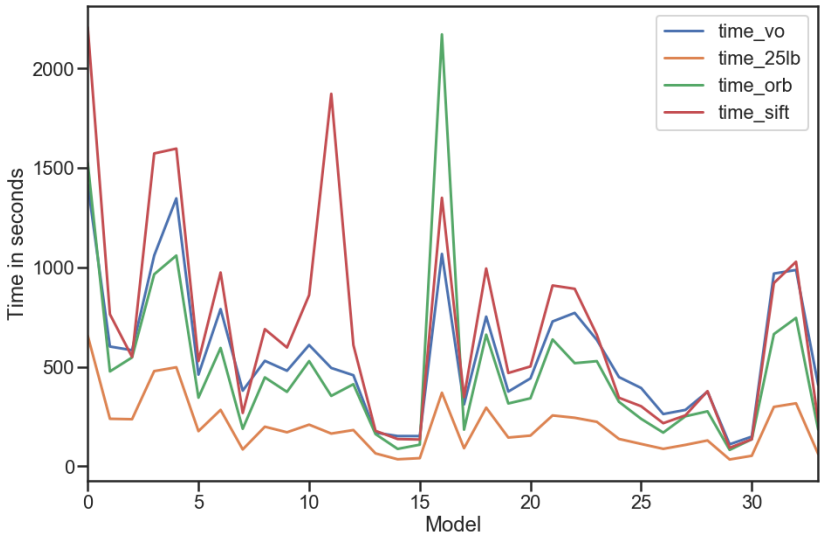


Figure 16: Frame selection time

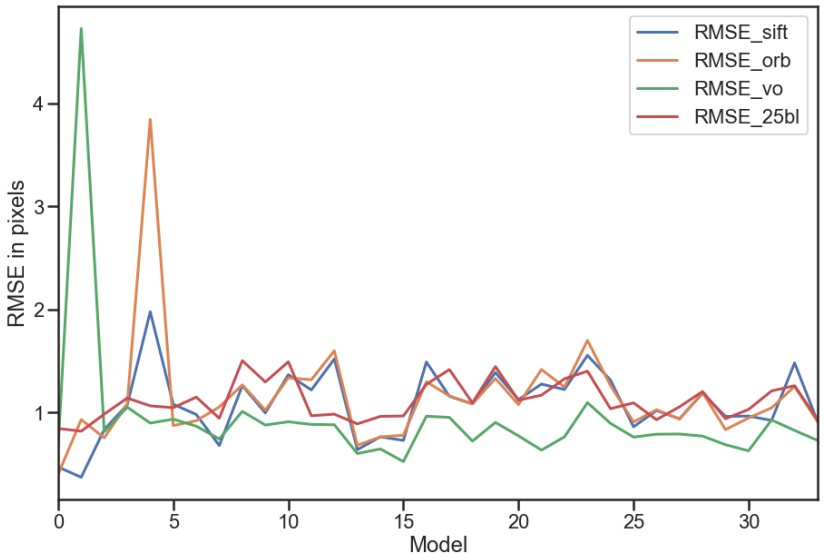


Figure 17: RMSE for 3D reconstruction for all the models

If a point $p$ is estimated to be at location $x, y, z$ in 3D space, then it is possible to reproject the point back to the images, which show it using the camera pose matrix of these images. Since each point used in the point cloud generation has its known location in image 2D coordinates, then the distance between the reprojected position and the actual

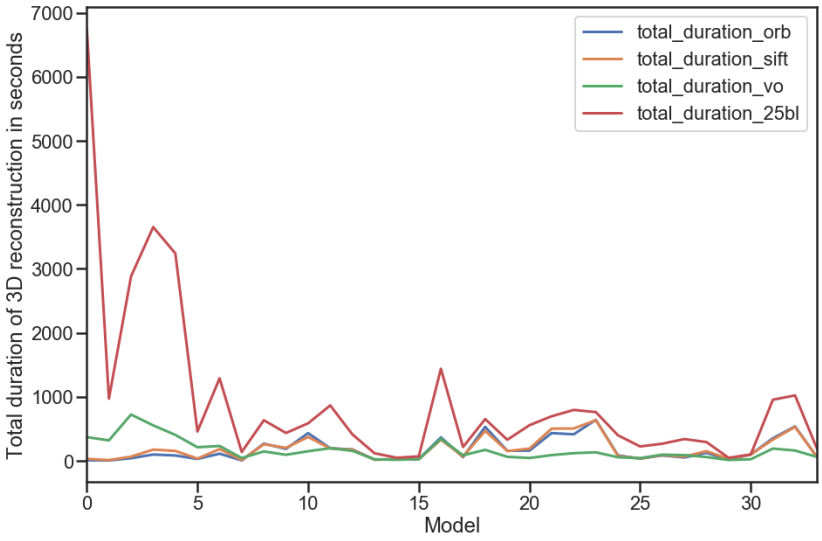


Figure 18: Total time for 3D reconstruction for all the models

position can be calculated. This distance is known as the reprojection error.

Calculating the reprojection error for all the points and corresponding frames in all possible combinations for one model will result in a list of error values. Representation of these values in as single number can be achieved using the RMSE. RMSE represents in this case the standard deviation of the reprojection error. This means that it represents how spread out the estimated points are from their actual locations on the images.

This method of calculation produces values in pixel coordinates which means it lacks the scale of the real world. Another method to calculate the reprojection error is to relate it to actual locations on the scanned objects using real world measurements. This however is not used due to several reasons, but mainly due to the nature of the project and the data, as access to the scanned objects in real world (the buildings) were not provided, and no real life measurements were given with the video data. Additionally, the motivation for this project included using data from the scanning platform alone, which in this case is only videos.

# 6 Discussion

## 6.1 Overall results

### 6.1.1 Frame Count

The frame count of the selected set has a direct impact on the processing time and the storage and processing power required to handle the processing. It is therefore one of the most important variables for evaluating the quality of a selection methodology. The initial frame count of every video is much larger than any of the methods used, as shown in Fig 19.

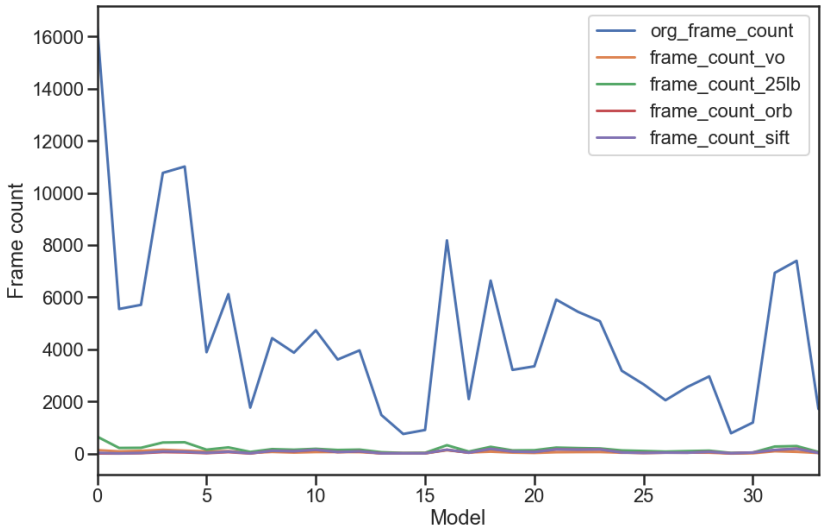


Figure 19: Original frame count of the videos against the frame count of all the methods

The baseline is standardized at 4% of the original frame count, which is consistently higher than the other three methods. When considering VO, SIFT and ORB only as in shown in Fig 20, it is clear that both ORB and SIFT are mostly similar, while VO fluctuates from significantly higher to significantly lower to almost similar in some of the models. This pattern is discussed in more details in section 6.2. The overall performance points to VO as a be marginally a better choices. Table 6 displays the summation of the frame count across all the models, which shows clear separation between the baseline and the three used methods.

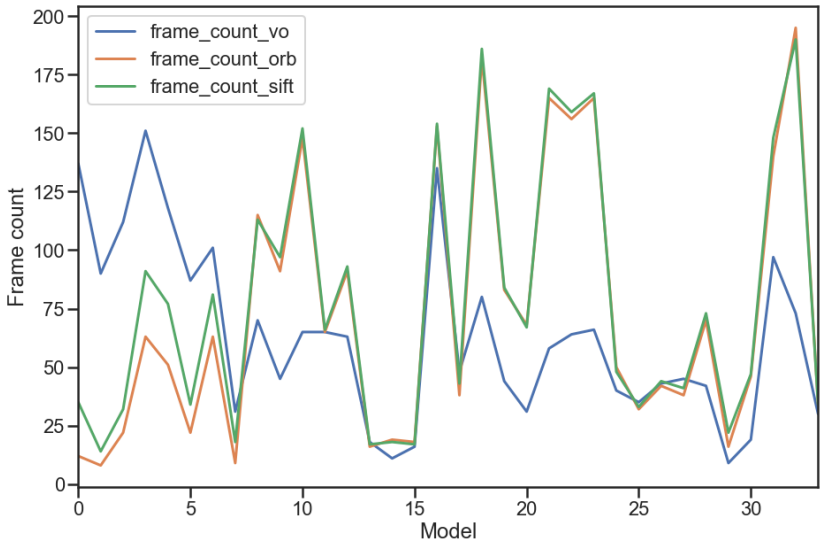


Figure 20: Visual odometry, SIFT and ORB selection frame count

| Method | Sum of frame count |
|---|---|
| Visual Odomerty | 2139 |
| ORB | 2486 |
| SIFT | 2664 |
| Baseline | 6236 |

Table 6: Summation of frame count of all models

### 6.1.2 Time

Processing time of both the selection and the 3D reconstruction processes are important factors in identifying an appropriate method of selecting frames.

**Selection time** : As shown in Fig 16, the selection time for the baseline is the lowest of all the methods due to two factors. Firstly, the selection process of the baseline is straight forward, and only includes checking the frame number and then saving or passing the frame. Secondly: the method of selection requires streaming the video only once, in comparison the other three methods requires streaming the video twice, which requires more time to process.

In most cases SIFT requires more time for selection than ORB and VO, which is expected for the comparison with ORB given the slower processing of both the feature

detection and description method used (SIFT itself), and the feature matching algorithm used (brute force). By contrast both ORB and VO use faster feature detection algorithms. VO requires mostly more time to process than ORB and less than SIFT, although it peaks to over 2000 seconds in one anomaly for the model number 16, which seems to be challenging for the other methods too, despite not having the highest initial frame count. Table 7 shows the summation of selection time across all models.

| Method | Sum of selection time in seconds |
|---|---|
| Visual Odomerty | 19152 |
| ORB | 16605 |
| SIFT | 23559 |
| Baseline | 6840 |

Table 7: Summation of selection time in seconds of all models

**3D reconstruction time** : The reconstruction time depends greatly on the number of selected frames. Fig 18 shows that the time required in case of the baseline is much higher in most cases than that of any of the other methods. Fig 21 shows the reconstruction time of the three selection methods. The disparity between VO on one side and ORB and SIFT on the other is displayed. VO is generally faster which is expected due to the lower overall frame count as in shown in table 6. Table 8 shows the summation of time required for 3D reconstruction.

| Method | Sum of 3D reconstruction time in seconds |
|---|---|
| Visual Odomerty | 5519 |
| ORB | 5926 |
| SIFT | 6252 |
| Baseline | 31835 |

Table 8: Summation of 3D reconstruction time in seconds of all models

The time of selection and 3D reconstruction are not summed together due to the differences in processing power of the computers used in each of the processes. The selection was conducted on a 2.2 GH processor while the reconstruction was implemented on a 3.3 GH processor. This means that the time needed for selection will most likely be shorter on the second machine, and will add more disproportional weight to the selection time in case of summing it with the processing time. Therefore, they are analysed separately.

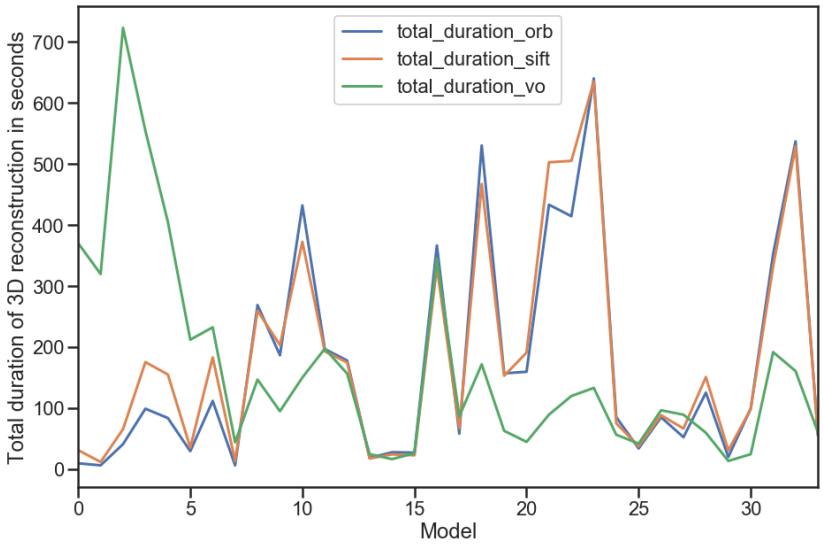


Figure 21: Visual odometry, SIFT and ORB 3D reconstruction time

### 6.1.3 RMSE

The quality of the produced mode is evaluated using the RMSE. Fig 17 shows that the VO had a consistently lower RMSE than the other methods including the baseline. One anomaly is recorded at model number 1 for VO and another at model number 4 for ORB and SIFT. This figure shows that VO presents a better solution in terms of quality in most cases.

## 6.2 Grouped results

In most of the displayed figures, the values change rapidly across the models. For example in Fig 21, VO changes from being considerably higher in the first few models to similar to ORB and SIFT in the middle and then to lower at the 18th model onward. This change happening in groups of models which reflect the video groups. The following discussion addresses this phenomenon in greater detail.

By examining the videos, it can be observed that the video groups differ in video count, video length, and, most importantly, the UAS flying pattern. As the VO, ORB and SIFT depend largely on the movement of features within the frame, the different movements of the UAS is expected to have an impact how the different selection methods perform. The flying patterns can be classified roughly to circular and linear. Two groups are mainly recorded in circular motion around the building, while three are recorded in linear move-

ment parallel to the facade, or with the camera directed downwards to capture the roof. Table 9 shows the video count of the different groups. Group A and group B are classified under circular flying pattern while group C, D and E are classified under linear flying pattern.

| Group symbol | Video count | Flying pattern |
|---|---|---|
| A | 3 | Circular |
| B | 5 | circular |
| C | 13 | Linear |
| D | 10 | Linear |
| E | 3 | Linear |

Table 9: Video count of video groups and flying patterns

### 6.2.1 Circular flying pattern

Both groups A and B are showing the same building with two years of separation between the two scanning exercises. The main differences are in the flying style and the color of the building. By examining the produced models closely, more variables could be noted.

The list of frames provided to the 3D reconstruction software pass through to stages as mentioned in section 4.3. The first is extracting and matching the points and the second stage is using these points to align the frames on 3D space. The alignment of the frames is dependent on the availability of overlapping matching points between frames. The software might fail to align some of the frames and therefore may exclude some of the data necessary to build the model. A complex flying pattern may result in the selection algorithm selecting insufficient frames for building the model. In such cases the model might fail partially or completely.

Group A initially contained five videos, two of which were not used due to the quality of the shots. The three videos used in selection showed complex flying patterns which included moving towards and away from the building, as well as tilting in the three axes, and changing of elevation. Figures 22, 23, 24 show the resulting model for ORB, SIFT and VO respectively. It is clear that ORB did not perform well as only one side of the building is modeled, while SIFT modeled three sides and lost one, and VO mapped them all. The baseline was not collected, as it had complete coverage with more frames, which caused the program to crash on display. Fig 25 shows the movement of the camera on the $x$ and $z$ axes as mapped by VO. Each dot represents a frame, and the movement can be traced from near the center with the blue color to the edges with the red colors.

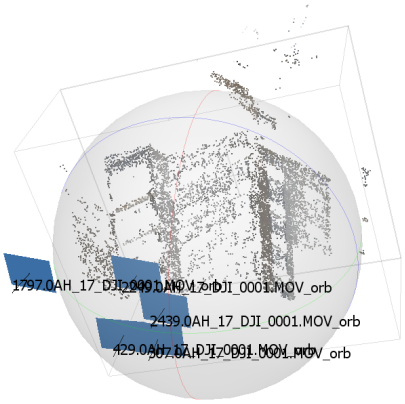


Figure 22: 3D reconstruction of ORB selected mode of model no 0

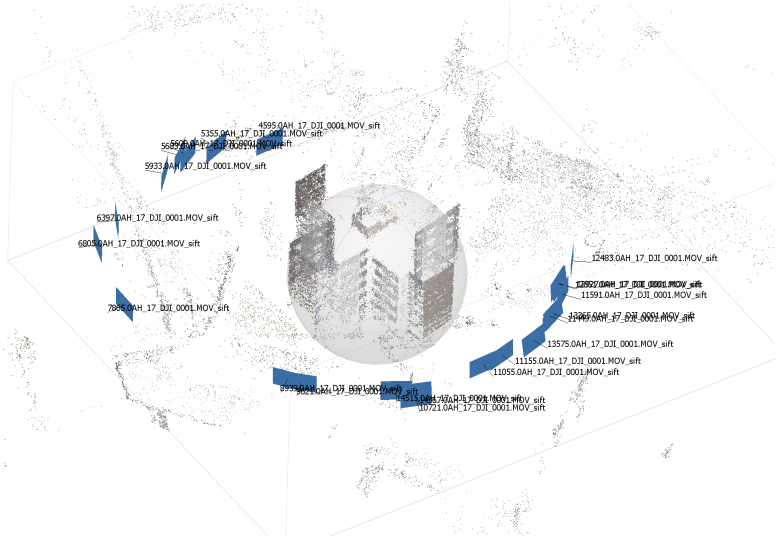


Figure 23: 3D reconstruction of SIFT selected mode of model no 0

Group B shows similar behaviour of the three methods as shown in Figures 26, 27, 28 where ORB and SIFT are not able to build a full model of the target as missing and not aligned frames cause the model to fail.

This is most likely due to the nature of the flying pattern which might be more dif-

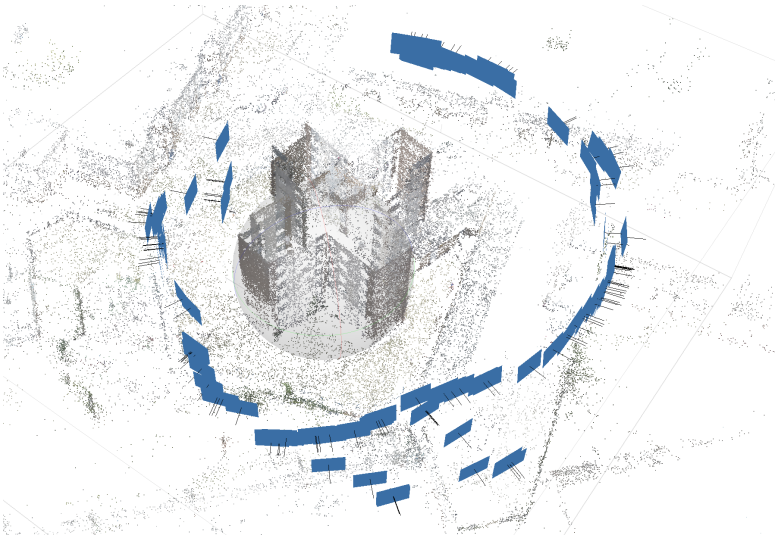

Figure 24: 3D reconstruction of VO selected mode of model no 0

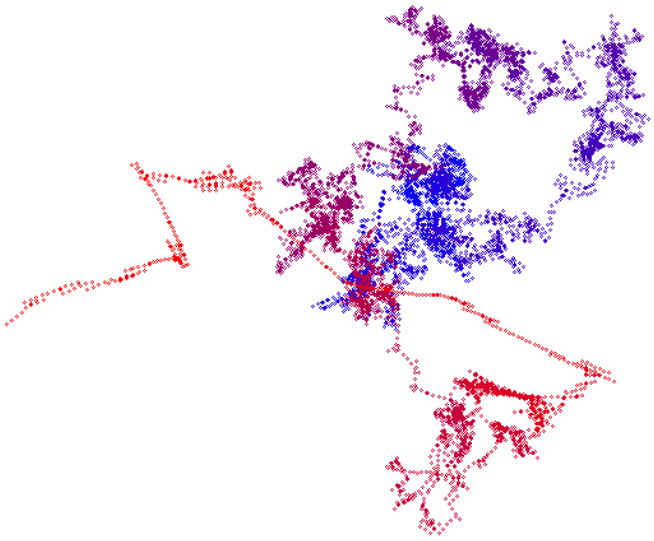

Figure 25: The movement of the camera on the $(x, z)$ axes mapped by VO for model no 1

ficult to track using the features displacement based approach as it largely assumes a movement of the camera across a 2D plane. Additionally, the target building has four similar faces, which might have caused some confusion for the selection algorithm. The marginal better performance of SIFT than ORB might be due to the better performance of SIFT in feature detection, description and the brute force matching algorithm.

The visual odometry based method mapped the movement of the camera in both the $(x, y)$ and $(x, z)$ planes. Only $(x, z)$ was used in selecting frames due to the flying pattern which did not include pure vertical movement. As long as the movement of the camera has a horizontal component, the frames will be mapped to different spatial grid cells and therefore get selected for 3D reconstruction. The code can easily be adopted in cases when a vertical movement is used in the scanning.

This shows that more metrics than RMSE might be required to evaluate the quality of the produced model which could be an aspect of further research.

Figure 26: 3D reconstruction of ORB selected mode of model no 4

### 6.2.2 Linear flying pattern

Groups C, D and E show different buildings but share the same pattern of linear to semi-linear flying, with a few exceptions of flying around a chimney. The model quality for

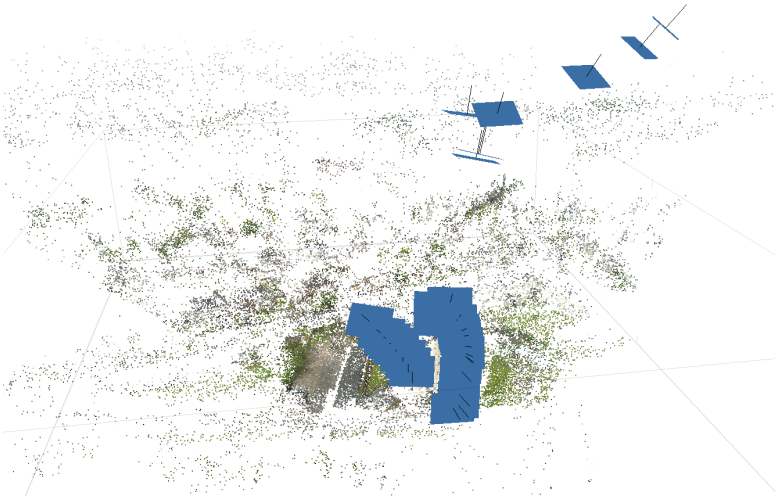

Figure 27: 3D reconstruction of SIFT selected mode of model no 4

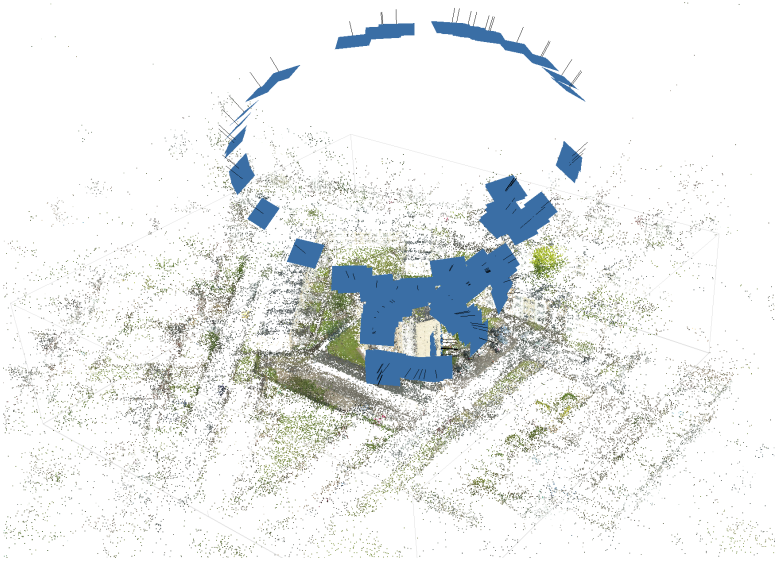

Figure 28: 3D reconstruction of VO selected mode of model no 4

each of the groups is shown in Figures 29, 30, 31 represented in RMSE values. VO is consistently lower in value than the other methods.

As the flying pattern is linear, it is possible for both SIFT and ORB to produce complete

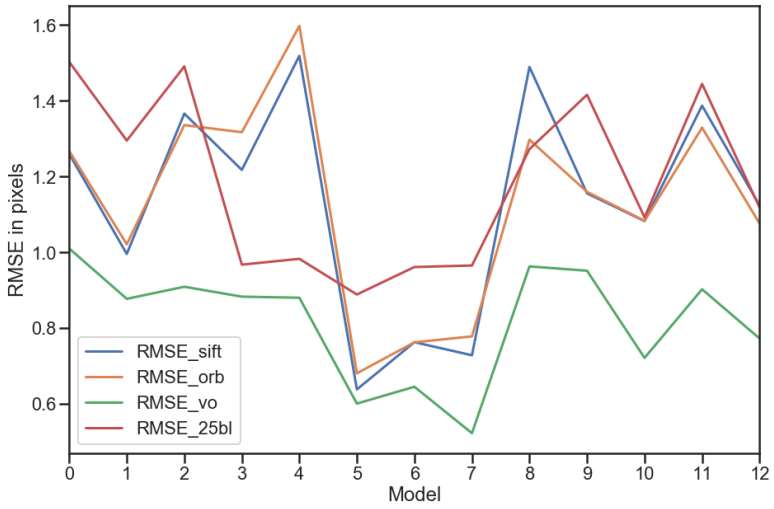


Figure 29: RMSE for group C

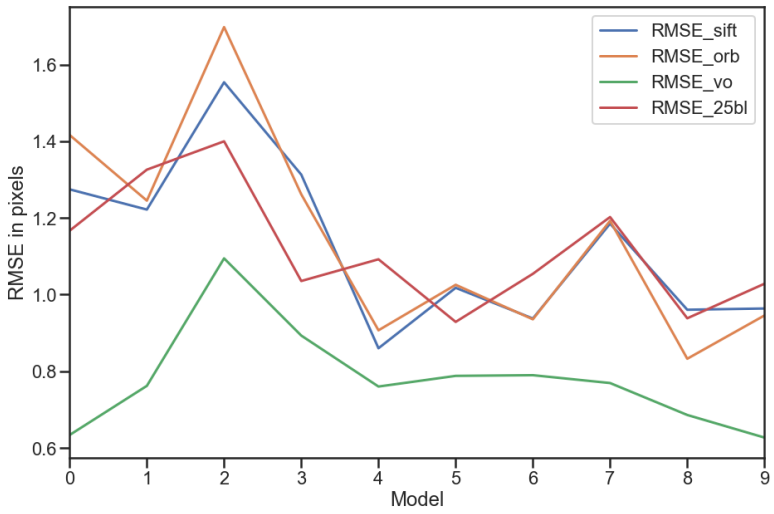


Figure 30: RMSE for group D

models. Figures 32, 33, 34, 35 show the 3D reconstruction of model number 27 for the baseline, ORB, SIFT and VO respectively. The simple linear movement of the camera can be inferred from the distribution of the frames of the baseline model. In this model, VO still exhibits better performance in terms of RMSE as shown in table 10.

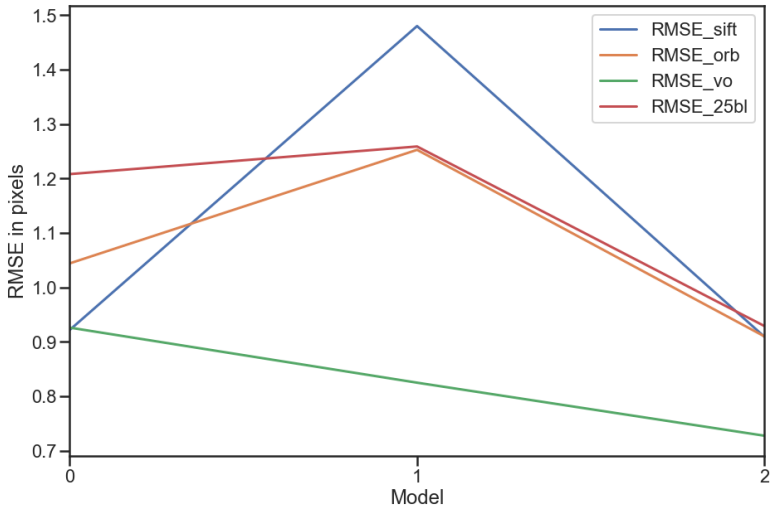


Figure 31: RMSE for group E

| Frame selection method | RMSE |
|---|---|
| Baseline | 1.054253 |
| ORB | 0.935541 |
| SIFT | 0.937622 |
| VO | 0.789761 |

Table 10: RMSE for model no. 27

## 6.3 Image quality assessment

The method used for image quality assessment proved to be adequate. A risk resulting from including a hazy image in the 3D reconstruction is that the modeling software might locate the features in shifted positions due to the effect of motion blur or lack of focus. The assessment algorithm eliminated images with lower amounts of clear edges. These included not just hazy and out of focus images, but also those where the camera is pointing towards the sky or horizon, where detectable features are low. Figures 36 and 37 show examples of images eliminated by the image quality assessment algorithm for haziness and pointing towards the sky.

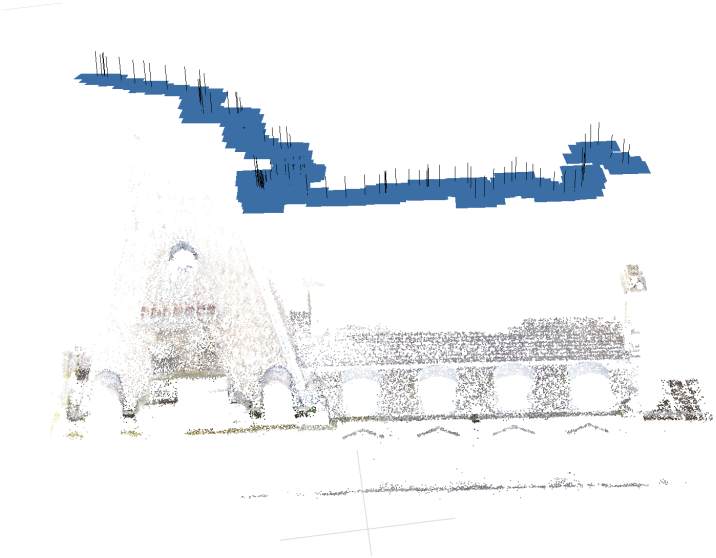

Figure 32: 3D reconstruction of baseline selected mode of model no 27

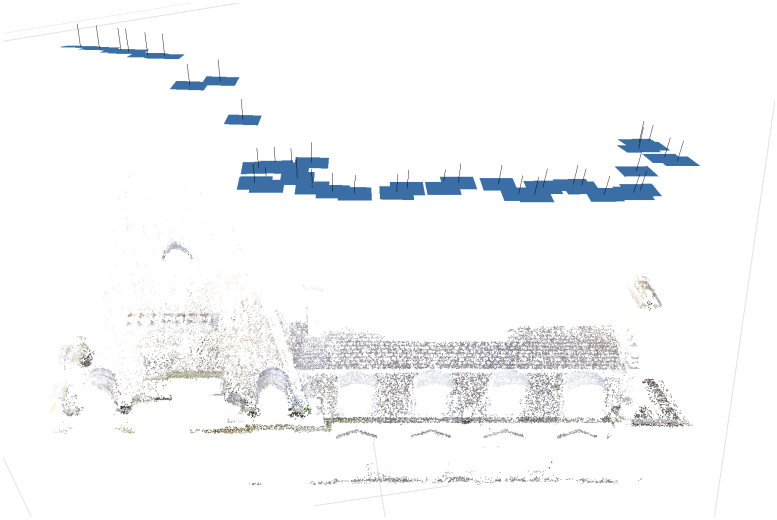

Figure 33: 3D reconstruction of ORB selected mode of model no 27

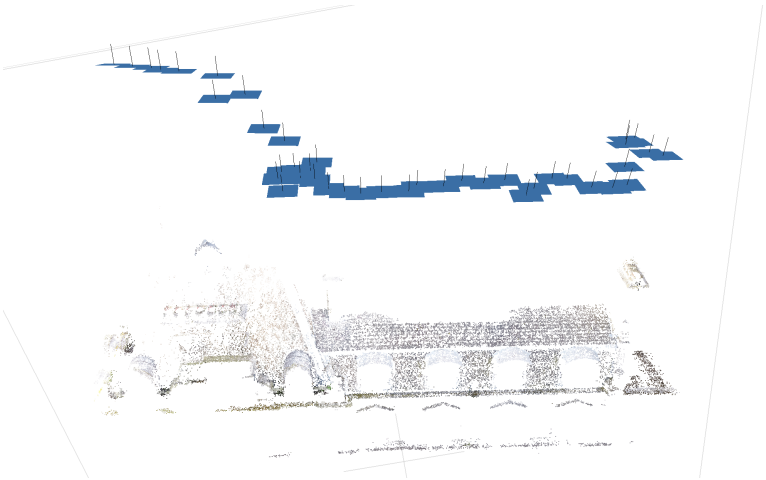

Figure 34: 3D reconstruction of SIFT selected mode of model no 27

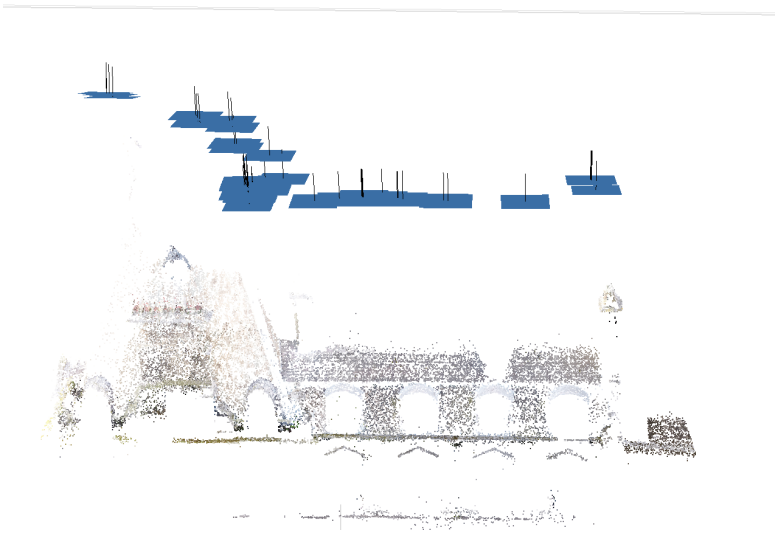

Figure 35: 3D reconstruction of VO selected mode of model no 27

# 7 Conclusions and recommendations

## 7.1 Conclusions

This work tested the possibility of using two different methods to select frames out of videos to use them for 3D reconstruction. Features displacement based methods were found to be better suited to extract frames from videos where the camera is moving

Figure 36: Hazy frame which selected out by the IQA stage


Figure 37: sky looking frame which selected out by the IQA stage

linearly rather than in a complex flying pattern. Visual odeometry based method was found to be more robust against the rapid complex movement of the camera as shown in section 6.2.1. Both approaches proved to extract less frames, and require less time for 3D reconstruction than the baseline method, while maintaining similar or better quality of the 3D model.

## 7.2 Limitations of work and areas for future research

This work opens possible avenues for several aspects to be explored; the following are some suggestions for potential areas of future research:

- **Feature detection, and matching**: other feature detector can be used including

those which realises on deep learning

- **Camera movement**: Using of more advanced methods for estimating camera movement can improve the results of frames selection. Those methods include variations of (Simultaneous Localization and Mapping) SLAM, such as (Large-Scale Direct SLAM) LSD-SLAM, or ORB-SLAM.
- **Camera model**: The model used in this work is a simple pinhole camera, which might have impacted the accuracy of estimating the movement of the camera and affected the selection by consequence. Studies with other models can yield further useful results.
- **Image Quality Assessment**: As the current method measures only the edges in the image, it is not robust against the background interference. A deep learning algorithm can be used to limit this effect and to add more details to the selection process.
- **Code optimization**: Several methods to improve the processing speed can be implemented, such as saving the positions estimated by VO to use them in the initial step of the 3D reconstruction, and implementing the code in a faster language such as C++.

# References


Bay, H., Tuytelaars, T. & Van Gool, L. (2006), SURF: Speeded up robust features, Technical report, ETH Zurich.

Beauchemin, S. S. & Barron, J. L. (1995), 'The Computation of Optical Flow', *ACM Computing Surveys (CSUR)* **27**(3), 433–466.

Bianco, S., Ciocca, G. & Marelli, D. (2018), 'Evaluating the performance of structure from motion pipelines', *Journal of Imaging* **4**(8).

Bouguet, J.-Y. (2000), Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm, Technical Report 2, Intel Corporation, Microprocessor Research Labs.

Calonder, M., Lepetit, V., Strecha, C. & Fua, P. (2010), BRIEF: Binary robust independent elementary features, Technical Report PART 4, CVLab.

Csurka, G., Dance, C. R. & Humenberger, M. (2018), From handcrafted to deep local features, Technical report, naverlab.
**URL:** *http://arxiv.org/abs/1807.10254*

Dornhege, C. & Kleiner, A. (2006), Visual Odometry for Tracked Vehicles, *in* 'Proc. of the IEEE Int. Workshop on Safety, Security and Rescue Robotics (SSRR)'.
**URL:** *http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-72546*

*FLANN Fast Library for Approximate Nearest Neighbors* (2019).
**URL:** *https://www.cs.ubc.ca/research/flann/*

Glocker, B., Komodakis, N., Tziritas, G., Navab, N. & Paragios, N. (2008), 'Dense image registration through MRFs and efficient linear programming', *Medical Image Analysis* **12**(6), 731–741.

Gu, X., Abdel-Aty, M., Xiang, Q., Cai, Q. & Yuan, J. (2019), 'Utilizing UAV video data for in-depth analysis of drivers' crash risk at interchange merging areas', *Accident Analysis & Prevention* **123**, 159–169.
**URL:** *https://www.sciencedirect.com/science/article/pii/S0001457518309631*

Harris, C., Harris, C. & Stephens, M. (1988), 'A combined corner and edge detector', *IN PROC. OF FOURTH ALVEY VISION CONFERENCE* pp. 147–151.
**URL:** *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.231.1604*

Horn, B. K. & Schunck, B. G. (1981), 'Determining optical flow', *Artificial Intelligence* **17**(1-3), 185–203.
**URL:** *https://linkinghub.elsevier.com/retrieve/pii/0004370281900242*

Humphreys, G. & Bruce, V. (1989), *Visual Cognition: Computational, Experiemental and Neuropsychological Perspectives*, Erlbaum Hove (U.K.).

Kamate, S. & Yilmazer, N. (2015), 'Application of Object Detection and Tracking Techniques for Unmanned Aerial Vehicles', *Procedia Computer Science* **61**, 436–441.
**URL:** *https://www.sciencedirect.com/science/article/pii/S1877050915030136*

Konecny, G. (2014), *Geoinformation: Remote Sensing, Photogrammetry and Geographic Information Systems*, 2 edition edn, CRC Press, Boca Raton, FL.

Linder, W. (2016), *Digital Photogrammetry*, 3rd ed. 20 edn, Springer, Berlin ; London.

Longuet-Higgins, H. (1987), A computer algorithm for reconstructing a scene from two projections, *in* 'Readings in Computer Vision', Elsevier, pp. 61–62.
**URL:** *https://linkinghub.elsevier.com/retrieve/pii/B978008051581650012X*

Lowe, D. G. (2004), 'Distinctive image features from scale-invariant keypoints', *International Journal of Computer Vision* **60**(2), 91–110.
**URL:** *http://link.springer.com/10.1023/B:VISI.0000029664.99615.94*

Lucas, B. D. & Kanade, T. (1981), Iterative Image Registration Technique With an Application To Stereo Vision, *in* 'Proceedings of the 7th International Joint Conference on Artificial Intelligence', Vol. 2, pp. 674–679.

Luhmann, T., Robson, S., Kyle, S. & Boehm, J. (2014), *Close-range photogrammetry and 3D imaging*, De Gruyter textbook, 2nd edn, De Gruyter.

Lv, Q., Josephson, W., Wang, Z., Charikar, M. & Li, K. (2007), Multi-probe LSH: Efficient indexing for high-dimensional similarity search, *in* '33rd International Conference on Very Large Data Bases, VLDB 2007 - Conference Proceedings', pp. 950–961.

Muja, M. & Lowe, D. G. (2009), Fast approximate nearest neighbors with automatic algorithm configuration, *in* 'VISAPP 2009 - Proceedings of the 4th International Conference on Computer Vision Theory and Applications', Vol. 1, pp. 331–340.

Nistér, D. (2004), 'An efficient solution to the five-point relative pose problem', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(6), 756–770.

Nister, D., Naroditsky, O. & Bergen, J. (2004), Visual odometry, *in* 'Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE.

Pérez-Alvárez, J. A., Gonçalves, G. R. & Cerrillo-Cuenca, E. (2019), 'A protocol for mapping archaeological sites through aerial 4k videos', *Digital Applications in Archaeology and Cultural Heritage* **13**(April), 2–11.

Pertuz, S., Puig, D. & Garcia, M. A. (2013), 'Analysis of focus measure operators for shape-from-focus', *Pattern Recognition* **46**(5), 1415–1432.
**URL:** *https://linkinghub.elsevier.com/retrieve/pii/S0031320312004736*

Rakha, T. & Gorodetsky, A. (2018), 'Review of Unmanned Aerial System (UAS) applications in the built environment: Towards automated building inspection procedures using drones', *Automation in Construction* **93**, 252–264.
**URL:** *http://www.sciencedirect.com/science/article/pii/S0926580518300165*

Rao, B., Gopi, A. G. & Maione, R. (2016), 'The societal impact of commercial drones', *Technology in Society* **45**, 83–90.
**URL:** *http://www.sciencedirect.com/science/article/pii/S0160791X15300828*

Rosten, E. & Drummond, T. (2006), Machine learning for high-speed corner detection, *in* 'Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)', Vol. 3951 LNCS, Springer Verlag, pp. 430–443.

Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. (2011), ORB: An efficient alternative to SIFT or SURF, *in* 'Proceedings of the IEEE International Conference on Computer Vision', pp. 2564–2571.

Sandau, R., Beisl, U., Braunecker, B., Cramer, M., Driescher, H., Eckardt, A., Fricker, P., Gruber, M., Hilbert, S., Jacobsen, K., Jagschitz, W., Jahn, H., Kirchhofer, W., Leberl, F., Neumann, K. J., Von Schönermark, M. & Tempelmann, U. (2010), *Digital airborne camera: Introduction and technology*, Springer.

Series, B. T. (2012), 'Methodology for the subjective assessment of the quality of television pictures', *Recommendation ITU-R BT* pp. 500–513.

Shi, J., Shi, J. & Tomasi, C. (1994), 'Good Features to Track', pp. 593–600.
**URL:** *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.2669*

Srinivasan, M., Zhang, S., Lehrer, M. & Collett, T. (1996), 'Honeybee navigation en route to the goal: visual flight control and odometry', *Journal of Experimental Biology* **199**(1).

Szeliski, R. (2011), 'Computer vision: algorithms and applications', *Choice Reviews Online* **48**(09), 48–5140.

Talukder, A., Goldberg, S., Matthies, L. & Ansar, A. (n.d.), Real-time detection of moving objects in a dynamic scene from moving robotic vehicles, *in* 'Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)', Vol. 2, IEEE, pp. 1308–1313.
**URL:** *http://ieeexplore.ieee.org/document/1248826/*

Thevara, D. J. & Vasanth Kumar, C. H. (2018), Application of photogrammetry to automated finishing operations, *in* 'IOP Conference Series: Materials Science and Engineering', Vol. 402, Institute of Physics Publishing.

Thrun, S., Burgard, W. & Fox, D. (1999), *Probabilistic robotics*, MIT Press.

*TRIK - Drone mapping and 3D reporting software for structural inspection* (n.d.).
**URL:** *https://gettrik.com/*

*uoip/monoVO-python: A simple monocular visual odometry project in Python* (n.d.).
**URL:** *https://github.com/uoip/monoVO-python*

Wang, Z. & Bovik, A. C. (2005), 'Modern image quality assessment', *Synthesis Lectures on Image, Video, and Multimedia Processing* **3**(1), 1–156.
**URL:** *http://www.morganclaypool.com/doi/abs/10.2200/S00010ED1V01Y200508IVM003*

Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004), 'Image quality assessment: From error visibility to structural similarity', *IEEE Transactions on Image Processing* **13**(4), 600–612.
**URL:** *http://ieeexplore.ieee.org/document/1284395/*

*What Is Camera Calibration* (2019).
**URL:** *https://www.mathworks.com/help/vision/ug/camera-calibration.html*

Wikipedia, F. (2012), 'K-D Tree'.
**URL:** *https://en.wikipedia.org/wiki/K-d*$_{tree}$

Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R. (2015), 'An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics', *Intelligent Industrial Systems* **1**(4), 289–311.
**URL:** *http://link.springer.com/10.1007/s40903-015-0032-7*

Zhang, T., Liu, X., Kühnlenz, K. & Buss, M. (2009), Visual odometry for the autonomous city explorer, *in* '2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009', pp. 3513–3518.

# A Python Code

## A.1 Frame selection

### A.1.1

SIFT

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:

import cv2
```

```
from matplotlib import pyplot as plt
import time
import numpy as np
from operator import itemgetter
import os
import pandas as pd

# In[ ]:

path = "src"
videos_names = []
files_path = []
for file in os.listdir(path):
    if file.endswith(".MOV"):
        videos_names.append(file)
        files_path.append(os.path.join(path,file))

# In[ ]:

def select_frames (vid, vid_name):
    # read the video
    vidcap = cv2.VideoCapture(vid)

    # get the video dimensions
    if vidcap.isOpened():
        width = vidcap.get(3)
        height = vidcap.get(4)
        frame_count = vidcap.get(7)
        print (frame_count)

    scalar = 5
    # set video dimensions
    width = int(width/scalar)
    height = int(height/scalar)
    frame_size = width * height

    # set the feature exctraction method and the matcher
    sift = cv2.xfeatures2d.SIFT_create()
    bf = cv2.BFMatcher()

    # initiate timer
    start = time.time()

    # initiate counter
    count = 0

```

```
    # initiate accumulators
    x_distance_median_accumulator = 0
    y_distance_median_accumulator = 0

    # initiate frame lists
    selected_video_frames = []
    selected_section_frames = []
    intial_selection_count = 0
    while True:
            # read the video and convert image to gray
            success, image = vidcap.read()
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            gray = cv2.resize(gray, (width, height))

            # initiate the check by reading the first frame
            if count < 1:
                    currentframe = gray.copy()
                    kp2, des2 = sift.detectAndCompute(currentframe,None)

            # use the next frames based on the condition below
            if count % 2 == 0:

                    # copy the parameters of the previous frame
                    pastframe = currentframe.copy()
                    kp1, des1 = kp2.copy(), des2.copy()
                    currentframe = gray

                    # find the keypoints and descriptors with SIFT
                    kp2, des2 = sift.detectAndCompute(currentframe,None)

                    # apply bf knn matcher
                    matches = bf.knnMatch(des1,des2,k=2)

                    # ratio test as per Lowe's paper
                    good_matches = []
                    for i, match_pair in enumerate(matches):
                        #check if the matches consists of a pair of matches
     and not just one
                        if len(match_pair) == 2:
                            if match_pair[0].distance < 0.7 * match_pair
    [1].distance:
                                good_matches += [match_pair[0], match_pair
    [1]]

                    # initialize distances lists
                    x_distance = []
```

```
                    y_distance = []

                    # append distances to their lists
                    for mat in good_matches:
                        x_distance.append((kp1[mat.queryIdx].pt)[0] - (kp2[
mat.trainIdx].pt)[0])
                        y_distance.append((kp1[mat.queryIdx].pt)[1] - (kp2[
mat.trainIdx].pt)[1])

                    # accumulate the distance
                    x_distance_median_accumulator += np.median(x_distance)
                    y_distance_median_accumulator += np.median(y_distance)

                    # calculate the intersection ratio
                    intersection_pixels = (width - abs(
x_distance_median_accumulator)) * (height - abs(
y_distance_median_accumulator))
                    intersection_percent = (intersection_pixels /
frame_size) * 100

                    # add frames which has intersection ratios between 80%
and 85% to the section
                    if 90 < intersection_percent < 95:
                            selected_section_frames.append((vidcap.get(1),
cv2.Laplacian(gray, cv2.CV_64F).var()))
                            intial_selection_count += 1


                    # if intersection goes below 80% reset the intersection
 calculation and add selected frames to the main video selected frames
                    elif intersection_percent < 90:
                            print ("len of selected_section_frames", len(
selected_section_frames))
                            x_distance_median_accumulator = 0
                            y_distance_median_accumulator = 0
                            selected_video_frames.append(
selected_section_frames)
                            selected_section_frames = []


                    print ("count: ", count)
                    print ( "len of selected_video_frames", len (
selected_video_frames))

            #count frames
            count += 1
```

```
131
132             # check if video is fully read
133             if count >= frame_count:
134                 break
135
136     vidcap.release()
137
138     # select best quality frames
139     final_frames_ids = []
140     for frame_list in selected_video_frames:
141         final_frames_ids.append(max(frame_list,key=itemgetter(1))[0])
142
143
144     #create folder for the selected frames
145     if not os.path.exists(vid+'_sift'):
146         os.mkdir(vid+'_sift')
147
148
149     # save images
150     vidcap = cv2.VideoCapture(vid)
151     count = 0
152     while True:
153         success, image = vidcap.read()
154         if (vidcap.get(1)) in final_frames_ids:
155             cv2.imwrite(vid+'_sift/'+str(vidcap.get(1))+vid_name+"_sift.tif
    ",image)
156          #count frames
157         count += 1
158
159         # check if video is fully read
160         if count >= frame_count:
161                 break
162     vidcap.release()
163
164     end = time.time() - start
165     print ('time: ', end)
166     return (end, len (final_frames_ids), intial_selection_count )
167
168 # In[ ]:
169
170 time_list = []
171 frame_count_list = []
172 intial_selection_list = []
173
174 for count in range(len(files_path)):
175     process_time, frame_count, intial_selection = select_frames (files_path
```

```
    [count+2], videos_names[count+2])
    time_list.append(process_time)
    frame_count_list.append(frame_count)
    intial_selection_list.append(intial_selection)

# In[ ]:

df = pd.DataFrame({
    "video name" : videos_names,
    "time" : time_list,
    "frame_count" : frame_count_list,
    "intial_selection" : intial_selection_list
})

df.to_csv("src/SIFT.csv")

# In[ ]:

print ("Done!")
```

### A.1.2 ORB

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:

import cv2
from matplotlib import pyplot as plt
import time
import numpy as np
from operator import itemgetter
import os
import pandas as pd

# In[ ]:

path = "src"
videos_names = []
files_path = []
for file in os.listdir(path):
    if file.endswith(".MOV"):
        videos_names.append(file)
        files_path.append(os.path.join(path,file))

```

```
# In[ ]:

def select_frames (vid, vid_name):
    # read the video
    vidcap = cv2.VideoCapture(vid)

    # get the video dimensions
    if vidcap.isOpened():
        width = vidcap.get(3)
        height = vidcap.get(4)
        frame_count = vidcap.get(7)
        print ("intial frame count", frame_count)

    scalar = 5
    # set video dimensions
    width = int(width/scalar)
    height = int(height/scalar)
    frame_size = width * height

    # set the feature exctraction method and the matcher
    orb = cv2.ORB_create()
    FLANN_INDEX_LSH = 6
    index_params= dict(algorithm = FLANN_INDEX_LSH,
                       table_number = 6, # 12
                       key_size = 12,     # 20
                       multi_probe_level = 1) #2
    search_params = dict(checks=100)
    flann = cv2.FlannBasedMatcher(index_params,search_params)

    # initiate timer
    start = time.time()

    # initiate counter
    count = 0

    # initiate accumulators
    x_distance_median_accumulator = 0
    y_distance_median_accumulator = 0

    # initiate frame lists
    selected_video_frames = []
    selected_section_frames = []
    intial_selection_count = 0
    while True:
            # read the video and convert image to gray
            success, image = vidcap.read()
```

```
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            gray = cv2.resize(gray, (width, height))

            # initiate the check by reading the first frame
            if count < 1:
                    currentframe = gray.copy()
                    kp2, des2 = orb.detectAndCompute(currentframe,None)

            # use the next frames based on the condition below
            if count % 2 == 0:

                    # copy the parameters of the previous frame
                    pastframe = currentframe.copy()
                    kp1, des1 = kp2.copy(), des2.copy()
                    currentframe = gray

                    # find the keypoints and descriptors with orb
                    kp2, des2 = orb.detectAndCompute(currentframe,None)

                    # apply flann knn matcher
                    matches = flann.knnMatch(des1,des2,k=2)

                    # ratio test as per Lowe's paper
                    good_matches = []
                    for i, match_pair in enumerate(matches):
                        #check if the matches consists of a pair of matches
 and not just one
                        if len(match_pair) == 2:
                            if match_pair[0].distance < 0.7 * match_pair
[1].distance:
                                good_matches += [match_pair[0], match_pair
[1]]

                    # initialize distances lists
                    x_distance = []
                    y_distance = []

                    # append distances to their lists
                    for mat in good_matches:
                        x_distance.append((kp1[mat.queryIdx].pt)[0] - (kp2[
mat.trainIdx].pt)[0])
                        y_distance.append((kp1[mat.queryIdx].pt)[1] - (kp2[
mat.trainIdx].pt)[1])

                    # accumulate the distance
                    x_distance_median_accumulator += np.median(x_distance)
```

```
                    y_distance_median_accumulator += np.median(y_distance)

                    # calculate the intersection ratio
                    intersection_pixels = (width - abs(
x_distance_median_accumulator)) * (height - abs(
y_distance_median_accumulator))
                    intersection_percent = (intersection_pixels /
frame_size) * 100

                    # add frames which has intersection ratios between 80%
and 85% to the section
                    if 90 < intersection_percent < 95:
                            selected_section_frames.append((vidcap.get(1),
cv2.Laplacian(gray, cv2.CV_64F).var()))
                            intial_selection_count += 1


                    # if intersection goes below 80% reset the intersection
 calculation and add selected frames to the main video selected frames
                    elif intersection_percent < 90:
                            print ("len of selected_section_frames", len(
selected_section_frames))
                            x_distance_median_accumulator = 0
                            y_distance_median_accumulator = 0
                            selected_video_frames.append(
selected_section_frames)
                            selected_section_frames = []


                    print ("count: ", count)
                    print ( "len of selected_video_frames", len (
selected_video_frames))

            #count frames
            count += 1

            # check if video is fully read
            if count >= frame_count:
                break

    vidcap.release()

    # select best quality frames
    final_frames_ids = []
    for frame_list in selected_video_frames:
        final_frames_ids.append(max(frame_list,key=itemgetter(1))[0])
```

```
148
149
150     #create folder for the selected frames
151     if not os.path.exists(vid+'_orb'):
152         os.mkdir(vid+'_orb')
153
154
155     # save images
156     vidcap = cv2.VideoCapture(vid)
157     count = 0
158     while True:
159         success, image = vidcap.read()
160         if (vidcap.get(1)) in final_frames_ids:
161             img_name = vid+'_orb/'+str(vidcap.get(1))+vid_name+"_orb.tif"
162             cv2.imwrite(img_name,image)
163          #count frames
164         count += 1
165
166         # check if video is fully read
167         if count >= frame_count:
168                 break
169     vidcap.release()
170
171     end = time.time() - start
172     print ('time: ', end)
173     return (end, len (final_frames_ids), intial_selection_count )
174
175 # In[ ]:
176
177 time_list = []
178 frame_count_list = []
179 intial_selection_list = []
180
181 for count in range(len(files_path)):
182     process_time, frame_count, intial_selection = select_frames (files_path
      [count], videos_names[count])
183     time_list.append(process_time)
184     frame_count_list.append(frame_count)
185     intial_selection_list.append(intial_selection)
186
187 # In[ ]:
188
189 df = pd.DataFrame({
190     "video name" : videos_names,
191     "time" : time_list,
192     "frame_count" : frame_count_list,
```

```
    "intial_selection" : intial_selection_list
})

df.to_csv("src/ORB.csv")

# In[ ]:

print ("Done!")
```

### A.1.3 VO

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import cv2
from matplotlib import pyplot as plt
import time
import numpy as np
from operator import itemgetter
import os
import pandas as pd

# In[2]:

path = "src"
videos_names = []
files_path = []
for file in os.listdir(path):
    if file.endswith(".MOV"):
        videos_names.append(file)
        files_path.append(os.path.join(path,file))

# In[3]:

len(files_path)

# In[4]:

STAGE_FIRST_FRAME = 0
STAGE_SECOND_FRAME = 1
STAGE_DEFAULT_FRAME = 2
kMinNumFeature = 1000

# params for ShiTomasi corner detection
```

```
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict(winSize  = (21, 21),
                 maxLevel = 3,
                 criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT
    , 30, 0.01))

def featureTracking(image_ref, image_cur, kp_ref):
    kp2, st, err = cv2.calcOpticalFlowPyrLK(image_ref, image_cur, kp_ref,
    None, **lk_params)  #shape: [k,2] [k,1] [k,1]
    st = st.reshape(st.shape[0])
    kp1 = kp_ref[st == 1]
    kp2 = kp2[st == 1]

    return kp1, kp2

class PinholeCamera:
    def __init__(self, width, height, fx, fy, cx, cy,
                 k1=0.0, k2=0.0, p1=0.0, p2=0.0, k3=0.0):
        self.width = width
        self.height = height
        self.fx = fx
        self.fy = fy
        self.cx = cx
        self.cy = cy
        self.distortion = (abs(k1) > 0.0000001)
        self.d = [k1, k2, p1, p2, k3]

class VisualOdometry:
    def __init__(self, cam):
        self.frame_stage = 0
        self.cam = cam
        self.new_frame = None
        self.last_frame = None
        self.cur_R = None
        self.cur_t = None
        self.kp_ref = None
        self.kp_cur = None
        self.focal = cam.fx
        self.pp = (cam.cx, cam.cy)
        self.detector = cv2.FastFeatureDetector_create(threshold=10,
    nonmaxSuppression=False)
```

```
79
80      def processFirstFrame(self):
81          self.kp_ref = self.detector.detect(self.new_frame)
82          self.kp_ref = np.array([x.pt for x in self.kp_ref], dtype=np.
    float32)
83          self.frame_stage = STAGE_SECOND_FRAME
84
85      def processSecondFrame(self):
86          self.kp_ref, self.kp_cur = featureTracking(self.last_frame, self.
    new_frame, self.kp_ref)
87          E, mask = cv2.findEssentialMat(self.kp_cur, self.kp_ref, focal=self
    .focal, pp=self.pp, method=cv2.RANSAC, prob=0.999, threshold=3.0)
88          _, self.cur_R, self.cur_t, mask = cv2.recoverPose(E, self.kp_cur,
    self.kp_ref, focal=self.focal, pp = self.pp)
89          self.frame_stage = STAGE_DEFAULT_FRAME
90          self.kp_ref = self.kp_cur
91
92      def processFrame(self, frame_id):
93          self.kp_ref, self.kp_cur = featureTracking(self.last_frame, self.
    new_frame, self.kp_ref)
94          E, mask = cv2.findEssentialMat(self.kp_cur, self.kp_ref, focal=self
    .focal, pp=self.pp, method=cv2.RANSAC, prob=0.999, threshold=1.0)
95          _, R, t, mask = cv2.recoverPose(E, self.kp_cur, self.kp_ref, focal=
    self.focal, pp = self.pp)
96          self.cur_t = self.cur_t + self.cur_R.dot(t)
97          self.cur_R = self.cur_R.dot(R)
98
99
100         if(self.kp_ref.shape[0] < kMinNumFeature):
101             self.kp_cur = self.detector.detect(self.new_frame)
102             self.kp_cur = np.array([x.pt for x in self.kp_cur], dtype=np.
    float32)
103
104         self.kp_ref = self.kp_cur
105
106     def update(self, img, frame_id):
107         assert(img.ndim==2 and img.shape[0]==self.cam.height and img.shape
    [1]==self.cam.width), "Frame: provided image has not the same size as
    the camera model or image is not grayscale"
108         self.new_frame = img
109
110         if(self.frame_stage == STAGE_DEFAULT_FRAME):
111             self.processFrame(frame_id)
112         elif(self.frame_stage == STAGE_SECOND_FRAME):
113             self.processSecondFrame()
114         elif(self.frame_stage == STAGE_FIRST_FRAME):
```

```
            self.processFirstFrame()
        self.last_frame = self.new_frame

# In[5]:

def select_frames (vid, vid_name):

    start = time.time()
    print (vid_name)
    vidcap = cv2.VideoCapture(vid)

    width = vidcap.get(3)
    height = vidcap.get(4)
    org_frame_count = vidcap.get(7)


    resize_factor = 5
    width = int(width/resize_factor)
    height = int(height/resize_factor)
    frame_size = width * height

    x_list, y_list, z_list , sharp_list, frame_list  = [], [], [], [], []

    # selection base
    base = 5

    count = -1
    focal_length = 2709.06 # / resize_factor
    distortion_coefficient = -0.00681111

    cam = PinholeCamera(float(width), float(height), focal_length,
    focal_length, float(width)/2, float(height)/2)

    vo = VisualOdometry(cam)

    traj_amplification = 5

    traj_width = 1200
    traj_height = 1200

    # list coordinations
    traj_z = np.full((traj_width,traj_height,3),255, dtype=np.uint8)
    traj_y = np.full((traj_width,traj_height,3),255, dtype=np.uint8)


    #create folder for the selected frames
```

```
    if not os.path.exists(vid+'_vo'):
        os.mkdir(vid+'_vo')

    print ('initial frame_count is ', org_frame_count)

    #undistoration
    distCoeff = np.zeros((4,1),np.float64)

    # TODO: add your coefficients here!
    k1 = distortion_coefficient; # negative to remove barrel distortion
    k2 = 0.0;
    p1 = 0.0;
    p2 = 0.0;

    distCoeff[0,0] = k1;
    distCoeff[1,0] = k2;
    distCoeff[2,0] = p1;
    distCoeff[3,0] = p2;

    # assume unit matrix for camera
    cam_mat = np.eye(3,dtype=np.float32)

    cam_mat[0,2] = width/2.0  # define center x
    cam_mat[1,2] = height/2.0 # define center y
    cam_mat[0,0] = focal_length        # define focal length x
    cam_mat[1,1] = focal_length        # define focal length y

    while True:
            success, image = vidcap.read()
            count += 1
            if count >= org_frame_count:
                break
            if count % 2 != 0:
                continue
            # read the video and convert image to gray
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            gray = cv2.resize(gray, (width, height))
            sharp = cv2.Laplacian(gray, cv2.CV_64F).var()
            dst = cv2.undistort(gray,cam_mat,distCoeff)
            vo.update(dst, count)

            cur_t = vo.cur_t
            if(count > 2):
                x, y, z = cur_t[0], cur_t[1], cur_t[2]
            else:
                x, y, z = 0., 0., 0.
```

```
206
207             try:
208                 x_list.append(int(base * round(x[0]/base)))
209                 y_list.append(int(base * round(y[0]/base)))
210                 z_list.append(int(base * round(z[0]/base)))
211                 sharp_list.append(sharp)
212                 frame_list.append(vidcap.get(1))
213             except:
214                 pass
215
216             draw_x, draw_y = int((x*traj_amplification)+(traj_width/2)),
    int((z*traj_amplification)+(traj_height/2))
217             cv2.circle(traj_z, (draw_x,draw_y), 1, (255-((vidcap.get(1)/
    org_frame_count)*255),0,((vidcap.get(1)/org_frame_count)*255)), 1)
218             cv2.rectangle(traj_z, (10, 20), (600, 60), (255,255,255), -1)
219             text = "Coordinates: x=%2fpx y=%2fpx z=%2fpx"%(x,y,z)
220             cv2.putText(traj_z, text, (20,40), cv2.FONT_HERSHEY_PLAIN, 1,
    (0,0,0), 1, 8)
221 #             cv2.imshow('Trajectory_Z', traj_z)
222             cv2.imwrite(vid+'_vo/'+vid_name+"traj_z.png",traj_z)
223
224             # draw movement on the trajectory window
225             draw_x, draw_y = int((x*traj_amplification)+(traj_width/2)),
    int((y*traj_amplification)+(traj_height/2))
226             cv2.circle(traj_y, (draw_x,draw_y), 1, (0,255-((vidcap.get(1)/
    org_frame_count)*255),((vidcap.get(1)/org_frame_count)*255)), 1)
227             cv2.rectangle(traj_y, (10, 20), (600, 60), (255,255,255), -1)
228             text = "Coordinates: x=%2fpx y=%2fpx z=%2fpx"%(x,y,z)
229             cv2.putText(traj_y, text, (20,40), cv2.FONT_HERSHEY_PLAIN, 1,
    (0,0,0), 1, 8)
230 #             cv2.imshow('Trajectory_Y', traj_y)
231             cv2.imwrite(vid+'_vo/'+vid_name+"traj_y.png",traj_y)
232
233 #             cv2.imshow('Drone Camera', gray)
234             cv2.waitKey(1)
235             # check if video is fully read
236
237             if count >= org_frame_count:
238                 break
239
240     vidcap.release()
241     cv2.destroyAllWindows()
242
243     #create the grid
244     df = pd.DataFrame({
245     "ID" : frame_list,
```

```
    "x" : x_list,
    "y" : y_list,
    "z" : z_list,
    "sharp" : sharp_list
    })

    df_group = df.groupby(['x', 'z'])

    final_list = []
    for i, j in df_group:
        final_list.append(j.ID[j.idxmax(axis= 0)['sharp']])



    # save images
    vidcap = cv2.VideoCapture(vid)
    count = 0
    while True:
        success, image = vidcap.read()
        if (vidcap.get(1)) in final_list:
            cv2.imwrite(vid+'_vo/'+str(vidcap.get(1))+vid_name+"_vo.tif",
    image)
         #count frames
        count += 1

        # check if video is fully read
        if count >= org_frame_count:
                break
    vidcap.release()

    end = time.time() - start
    print ('time: ', end)
    return (end, len (final_list), org_frame_count )

# In[6]:

def wrtie_header(path):
    str_r = 'video name;time;frame_count;org_frame_count'
    f = open(os.path.join(path,"VO_fast_selection.txt"), "w")
    f.write(str_r)
    f.close()

# In[7]:

def write_output(path, video_name, time, frame_count, org_frame_count):
    str_r = '\n' + str(video_name) + ';' +  str(time) + ';' + str(
```

```
    frame_count) + ';' + str (org_frame_count)
    f = open(os.path.join(path,"VO_fast_selection.txt"), "a")
    f.write(str_r)
    f.close()

# In[8]:

wrtie_header(path)
for count in range(len(files_path)):
    process_time, frame_count, org_frame_count = select_frames (files_path[
    count], videos_names[count])
    write_output(path, videos_names[count], process_time, frame_count,
    org_frame_count)

print ("Done!")
```

## A.2 3D reconstruction and results collection

```
import Metashape
import os
import math
import statistics

def get_files_and_folders (path):
    folders = get_folders_lists(path)
    folders_files = {}
    for folder in folders:
        files =[]
        for file in os.listdir(os.path.join(path,folder)):
            if file.endswith(".tif"):
                files.append(os.path.join(os.path.join(path,folder),file))
        folders_files[folder] = files
    return (folders_files)

def get_folders_lists(path):
    folders = []
    for (dirpath, dirnames, filenames) in os.walk(path):
        for folder in dirnames:
            if folder.endswith(target):
                folders.append(folder)
    return (folders)

def wrtie_header(path):
    str_r = 'Model;camera_count;aligned_cameras;MSRE;count_of_points;
    std_error;max_error;match_photos_duration;align_cameras_duration'
    f = open(os.path.join(path,"result.txt"), "w")
```

```
    f.write(str_r)
    f.close()

def reconstrct(path, dic):
        for key, value in dic.items():
                pointcloud, camera_count, aligned_cameras, MSR,
    count_of_points, std_error, max_error, match_photos_duration,
    align_cameras_duration = None, None, None, None, None, None, None, None
    , None
                chunk = doc.addChunk()
                chunk.addPhotos(value)
                chunk.matchPhotos(accuracy=Metashape.HighAccuracy,
    generic_preselection=True, reference_preselection=False, keypoint_limit
    =80000, tiepoint_limit=8000 )
                chunk.alignCameras()

                pointcloud = chunk.point_cloud
                points = pointcloud.points
                error, tie_points = [], []
                for camera in [cam for cam in chunk.cameras if cam.
    transform]:
                    point_index = 0
                    photo_num = 0
                    for proj in pointcloud.projections[camera]:
                        track_id = proj.track_id
                        while point_index < len(points) and points[
    point_index].track_id < track_id:
                            point_index += 1
                        if point_index < len(points) and points[point_index
    ].track_id == track_id:
                            if not points[point_index].valid:
                                continue

                            dist = camera.error(points[point_index].coord,
    proj.coord).norm() ** 2
                            error.append(dist)
                            photo_num += 1
                    tie_points.append(photo_num)

                MSR = math.sqrt(sum(error) / len(error))
                aligned_cameras = len(tie_points)
                count_of_points = sum(tie_points) / len(tie_points)
                std_error = statistics.stdev(error)
                max_error = max(error)

                meta_chunk = chunk.meta
```

```
                meta_pointcloud = pointcloud.meta

                match_photos_duration = meta_pointcloud['MatchPhotos/duration']
                align_cameras_duration = meta_chunk['AlignCameras/duration']

                # MSR, aligned_cameras, count_of_points, std_error, max_error = get_RMSE(pointcloud)
                camera_count = len(value)
                str_r = '\n' + str(key) + ';' +  str(camera_count) + ';' + str(aligned_cameras) + ';' + str (MSR) + ';' + str(count_of_points) + ';' + str(std_error) + ';' + str(max_error) + ';' + str(match_photos_duration) + ';' + str(align_cameras_duration)
                f = open(os.path.join(path,"result.txt"), "a")
                f.write(str_r)
                f.close()
                # write_output (path, key, camera_count, aligned_cameras, MSR, count_of_points, std_error, max_error, match_photos_duration, align_cameras_duration)

path = "src"
wrtie_header(path)
dic = get_files_and_folders(path)
doc = Metashape.app.document
reconstrct(path, dic)
```