AN ANALYSIS OF REAL-TIME SHAPE FROM MOTION

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2020

Student id: 10459380

Department of Computer Science

Contents

Word Count: 16245

List of Tables

List of Figures

Abstract

AN ANALYSIS OF REAL-TIME SHAPE FROM MOTION
Chuanhai Luo
A dissertation submitted to The University of Manchester
for the degree of Master of Science, 2020

This thesis presents an analysis of a realtime Shape from Motion algorithm used to create a sparse 3D shape model of the scene in real-time from an image stream provided by a monocular hand-held camera. To fully understand the realtime Shape from Motion algorithm, foundations concerning camera models and camera calibration are presented and followed by image processing techniques including features detection and features matching. Before the depth searching procedure for each feature point, the camera's pose information at each viewpoint has to be estimated. Thus, after the image processing chapter, the Epipolar Geometry is introduced, based on which the rotation and translation between two consecutive viewpoints are extracted. Via triangulation and with a depth searching range, the depth having the least Reprojection Error is used to construct a local point cloud. Also, a direct solution based on the Least-Squares Method for computing the depth is proposed and is twenty times faster than the depth searching method. Then local point clouds are stitched together to form a comprehensive 3D shape model. An implementation using OpenCV on iPhone XR is used to evaluate the performance of the discussed methods and achieves a speed of 0.8 second per reconstruction for around 500 feature points. What's more, as an application of shape from motion and a future promising research field, the Visual Mono-SLAM is also discussed.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank Dr. Tim Morris for the proofreading and helpful suggestions during the implementation and the evaluation; Wenbo Mi, Yufei Yan and Johannes Rohn for emotional support during the COVID-19 quarantine and last but not the least my parents for their support over the years.

Chapter 1

Introduction

Constructing the three dimensional model for a scene from its projections, images, has been a fundamental task in Computer Vision. Even though there are a lot of hardware-based solutions for accurate scene reconstruction, such as LIDAR, the vision-based approach still has attracted researchers' attention because cameras are cheap and ubiquitous compared with LIDAR [46]. Besides, cameras also provide very rich and high resolution of information about the environment. There are generally two types of vision-based 3D reconstruction schemes. One is called Stereo [52] in which two or more relatively fixed cameras probe the environment simultaneously. Because the cameras are relatively fixed, the baselines and orientation differences among them are known. By calculating the disparity information among images captured at the same time, the depth information of a scene point relative to the cameras can be accurately computed. Stereo is a good solution for vision-based scene reconstruction, but it needs an arrangement of cameras. To obtain a reliable and stable three dimensional model of the environment, the arrangement among cameras has to be stable and shootings need to be synchronized. Another solution is called Shape from Motion (SfM) [46], also known as structure from motion, which uses one camera and reconstructs the environment based on an image stream produced by the camera. SfM treats each pair of consecutive images as a Stereo problem with two cameras. However, the baseline in SfM can't be accurately measured except the direction of the translation of the camera between two viewpoints. Thus, SfM can only reconstruct the shape model of a scene in which the orientation and uniform scaling of the scene are removed. The setup of SfM for scene reconstruction is simple. However, due to the imperfections of the camera system, the high computational complexity of features detection/matching among images, etc., it's still a hard task to reconstruct the scene which makes SfM remain at

the forefront of the Computer Vision research [46].

This thesis performs an analysis of the realtime SfM along with an implementation of the algorithm, an iOS reconstruction App, to evaluate its effectiveness.

1.1 Shape from motion

SfM is believed to be started with the seminal paper by Longuet-Higgins [32]. A typical pipeline of SfM includes camera calibration, features detection and matching, estimating the camera's poses and locations of world points, and optional bundle adjustment which refines the shape model by setting a global optimization and minimizing the total reprojection error function [60]. According to the way of processing images, SfM can be divided into two categories: the online SfM and the offline SfM. Offline SfM processes the whole images at the same time and reconstructs the scene by solving an optimization problem, such as the bundle adjustment [46]. Unfortunately, the bundle adjustment is a non-linear optimization procedure and non-convex, so that its optimization in reality usually converges to an undesired local minimum. Therefore, it's crucial to feed bundle adjustment an initial estimation of camera motions and the shape model which is as close to the true solution as possible. The online SfM adopts an incremental strategy that constructs the shape model gradually by processing images sequentially and stitching local point clouds together. As realtime SfM is an online reconstruction, this project would follow the pathway of the online SfM. The development of techniques used in the online SfM algorithm will be briefly introduced in the following subsections. Visual Mono-SLAM as an application and a future work of SfM will be described in section 1.2. The outline of this thesis will be listed in section 1.3.

1.1.1 Features detection & matching

Features detection and matching are the fundament for various computer vision problems such as camera calibration, object recognition, and scene reconstruction. The features approach is a bottom-up approach to dealing with computer vision problems, as the projection of a scene can be regarded as a collection of features. The research of finding correspondences between images can be divided into three stages. The first one is feature detection in which distinctive locations (feature points) in images are extracted by a *detector* with the assumption that the more distinctive they are, the less

ambiguous to be matched in other images. Apart from the preferred uniqueness of features, the detector itself should have high repeatability, so that the same feature under different viewing conditions can be reliably identified [42]. Once a feature point is extracted, it needs an ID which can hopefully uniquely represent the feature point. In the feature matching process, two features will be classified as the same one if they have the same ID or their ID are very similar. In computer vision, the ID is represented by a feature vector called *descriptor* in which there are descriptions of the feature point computed from its neighborhood. Finally, the features are matched between images based on a distance measurement between the descriptors such as Euclidian distance [43]. The dimension of descriptors has a direct impact on the accuracy of features matching. A higher dimension generally achieves better matching results but is less desirable for fast matching.

The first widely used detector is probably the Harris corner detector back in 1988 [24]. However, the Harris detector is not scale-invariant so that it performs poorly between images with even only a scale difference. Later, Mikolajczyk and Schmid refined the Harris corner detector and proposed two robust and scale-invariant detectors called Harris-Laplace and Hessian-Laplace [37]. They used the determinant of the Hessian matrix for locating features and the Laplacian of Gaussians for scale analysis. To speed up the scale analysis, Lowe proposed to use the Difference of Gaussians as an approximation of the Laplacian of Gaussians [33] and proposed a scale- and rotation-invariant feature detector and descriptor called Scale Invariant Feature Transform (SIFT) [34] in 2004 which has been proven remarkably successful in computer vision. However, it is patented and imposes a large computational burden which is less desirable for realtime systems. Later, a faster rotation- and scale-invariant detector called Speeded-Up Robust Features (SURF) was proposed in 2006 [4]. It provides a comparable performance with SIFT at a faster speed. But it's also patented. Therefore, the OpenCV community proposed an efficient alternative to SIFT and SURF called Oriented Fast and Rotated BRIEF (ORB) in 2011 [51] which is also rotation- and scale-invariant and almost two orders of magnitude faster than SIFT and SURF while providing comparable performance.

1.1.2 Pose estimation

The pose difference estimation for a pair of images in shape from motion problem probably starts with the eight-point algorithm [32] proposed in 1981 – a linear method

based on point correspondences. Given that the relative rotation and translation between two viewpoints is encapsulated in the fundamental matrix, which will be introduced in detail in section 4.1, and that the fundamental matrix can be calculated with at least 8 feature matches between images, the eight-point algorithm computes the fundamental matrix by solving an equation system formed by feature matches and factorizes the rotation matrix and the translation vector out from the fundamental matrix. However, the original eight-point algorithm is very sensitive to noise and performs with errors as large as 10 pixels [25]. To address this problem, the normalized eight-point algorithm [25] was proposed in 1997, in which the coordinates of feature matches are centered and normalized before being used for calculating the fundamental matrix. What's more, due to the outliers in feature matches, the normalized eight-point algorithm alone is not sufficient for computing the fundamental matrix. A better solution is to integrate RANSAC [19] into the algorithm so that the outliers can be excluded. The later five-point algorithm for pose estimation [45] can be regarded as a variant of the eight-point algorithm.

1.1.3 Point cloud construction & registration

Once the pose change between two viewpoints is estimated, the depth information for each feature match can be searched using the triangulation and the depth with a minimum reprojection error is returned. A local point cloud for a pair of images can be constructed by searching depths for all feature matches. Then, all local point clouds are stitched together to form a comprehensive three-dimensional shape model of the scene using the Procrustes Transformation [17] in which the translation, rotation, and uniform scaling components of objects are removed and what remain, objects' shape, are compared with.

1.2 Application: Visual Mono-SLAM

The applications of SfM can be classified into up to ten categories, ranging from augmented reality, autonomous navigation/guidance, motion capture, vision-based 3D modeling to recognition, etc [60]. Particularly in this thesis, the ability of SfM to perform Visual Mono-SLAM as a way to achieve autonomous navigation/guidance will be explored. Even though there are already some solutions for localization and navigation such as Global Positioning System (GPS), there are still some scenarios where

the GPS signal may not be strong enough because of occlusions e.g., indoors [60]. Besides, there is always a demand for map creation. To avoid the tedious map creation by hand, several automatic map creation techniques have emerged such as SLAM short for Simultaneous Localization And Mapping. SLAM is designed to solve the map creation problem and the localization problem in that map simultaneously and is regarded as one of the most popular at the moment [2].

SLAM can be classified into two categories according to the type of sensor employed: Range finder based approaches and Vision-based approaches [2]. In the first one, the distance of the scene relative to the SLAM equipment is measured by laser light, sonar, or infra-red sensors, among which the laser range finders generally provides better accuracy and higher frame rates [2]. Cameras have also attracted researchers' attention as a choice for SLAM because they are noninvasive, well-understood, cheap, and ubiquitous [11]. In the vision-based approaches for SLAM, the SfM algorithm can be used to compute the distance information. However, due to the high computational complexity of image processing, only a sparse point cloud of the environment can be generated for realtime applications. Luckily, a sparse point cloud is enough for map creation and navigation purposes. Once the distance information is gained, the map can be generated by employing probabilistic methods like the Extended Kalman Filter [11].

1.3 Thesis outline

Chapter 1: A brief introduction of SfM, the development of techniques used in it, and Visual Mono-SLAM as one of its applications. Furthermore, an outline of the thesis.

Chapter 2: An introduction of a basic camera model, a distortion model, and the Camera Calibration which calculates values of the parameters in the camera model and the distortion model.

Chapter 3: A description of two feature detectors called SURF and ORB and some common matching schemes for feature matching.

Chapter 4: The Epipolar Geometry between two views and the eight-point algorithm for estimating pose change between two viewpoints are introduced.

Chapter 5: An introduction of methods for constructing local point clouds. Besides, Procrustes Transformation is introduced in this chapter for stitching local point clouds into a comprehensive three-dimensional shape model of the scene.

Chapter 6: The realtime SfM algorithm was implemented on iPhone XR. The user interface of the iOS reconstructing App and the reconstruction results are presented and evaluated in this chapter.

Chapter 7: This chapter concludes this thesis and explores Visual Mono-SLAM as an application and a future work of SfM.

Chapter 2

Camera Model

As stated in chapter 1, the only sensor information used in the SfM algorithm is an image stream from a camera. So before exploring the algorithm itself, the pinhole camera model, a simple and commonly used camera model in Computer Vision, used in SfM and how it can be improved by incorporating a distortion model to address the imperfections found in real camera lenses will be introduced. The remainder of this chapter would present Camera Calibration that calculates the intrinsic and the extrinsic parameters of a camera, through which a 3D world point can be mapped to a 2D image point, and vice versa.

2.1 The pinhole camera model

2.1.1 The basic model

The pinhole camera model presented in figure 2.1 describes the mathematical relationship between the coordinate of a point in the three-dimensional scene and its projection onto a two-dimensional image, where the camera aperture is expressed as a point [16]. Although the pinhole model requires some assumptions, such as the aperture point, which are lacking in real cameras, it's still widely used in the Computer Vision community because of its mathematical convenience and simplicity and it does give a reasonable first approximation [26].

In the figure 2.1, O is defined as the center of projection, and the optical axis is parallel with the Z axis. f is the focal length. In a real camera, the image plane is behind O, in the negative area of the Z axis, and the image is upside down compared with the scene. However, in the pinhole camera model, the image plane is moved to

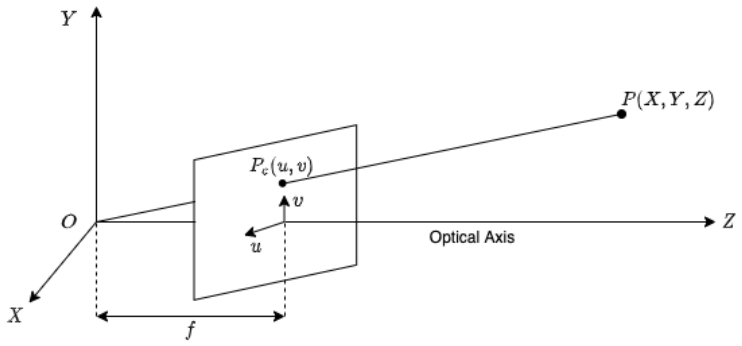


Figure 2.1: Pinhole Camera Model

in front of O, so that the image could be the identical up to a scale projection of the scene, which simplifies the model and doesn't sacrifice the validity of the model at the same time. P is a point in the scene whose coordinate in the camera coordinate system is (X,Y,Z). P_c is the projection of point P onto the image plane whose coordinate in the image plane is (u,v). Using similar triangles, the equations behind the model can be established as:

$$\frac{x}{X} = \frac{y}{Y} = \frac{f}{Z} \tag{2.1}$$

where (x,y,f) is the coordinate of P_c in the camera coordinate system. Equation 2.1 can be rewritten as:

$$x = \frac{fX}{Z}, \quad y = \frac{fY}{Z} \tag{2.2}$$

Using homogenous coordinates, the equation 2.2 can be combined as:

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

The location of P_c in the image coordinate is described by the number of pixels to the left-up corner of the image. So there is a transformation for P_c between the camera coordinate system (x,y,f) and the image coordinate system (u,v). Assume that the physical size of a pixel are d_x for width along the x-axis direction and d_y for height along the y-axis direction, and the center of the image in the image coordinate system

is (u_o, v_o). Then, point $P_c(u,v)$ can be formulated as:

$$u = \frac{x}{d_x} + u_o, \quad v = \frac{y}{d_y} + v_o \tag{2.4}$$

which can be rewritten in homogenous coordinate as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_o \\ 0 & 1/d_y & v_o \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.5}$$

So integrating equation 2.3 and equation 2.5, the pinhole camera model can be expressed in a matrix form as:

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_o \\ 0 & 1/d_y & v_o \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f/d_x & 0 & u_o & 0 \\ 0 & f/d_y & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

If the coordinate of point P in world coordinate system is (X_w, Y_w, Z_w), equation 2.6 can be extended as:

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_o & 0 \\ 0 & f_y & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_o & 0 \\ 0 & f_y & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.7}$$

where f_x and f_y are focal length in pixel units, $R_{3\times3}$ and $t_{3\times1}$ are the rotation matrix and translation vector between the camera coordinate system and the world coordinate system. Note that the Z is the depth information of point P in the camera coordinate system. The parameters in the equation 2.7 can be classified into two categories [27]:

- Intrinsic parameters including f_x, f_y, u_o and v_o.
- Extrinsic parameters including R and t.

2.1.2 An adaption of the basic model with lenses

Note that in the pinhole model the aperture is assumed to be a single point. However, in reality, the aperture can not be infinitely small. Besides, the aperture size has a

big influence on the formation of images. As the aperture size increases, the number of light rays reflected from the scene passing through the aperture increases, which results in a brighter but more blurred image. Because each point on the sensor can be affected by light rays from multiple points in the scene, which blurs the image. On the contrary, decreasing the aperture size would reduce the number of light rays passing through, leading to a sharper but darker image. The effect of the aperture size on the formation of images is showed in figure 2.2 [26].

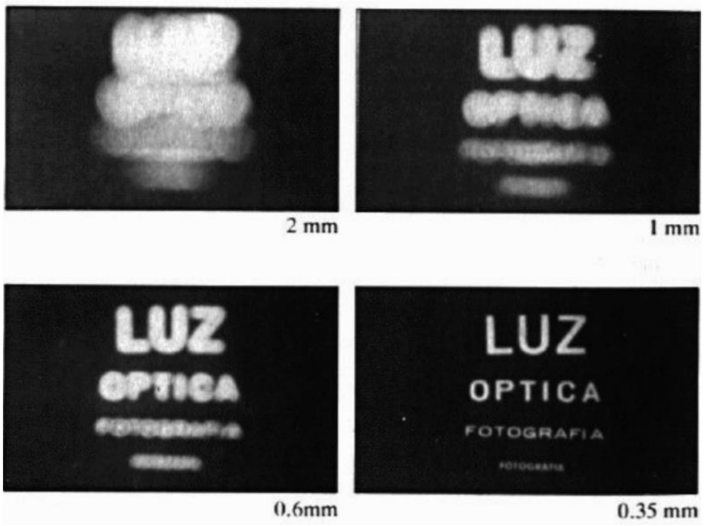


Figure 2.2: Effects of varying aperture size

To mitigate the conflict between the brightness and the sharpness, modern cameras use photographic lenses that can focus and disperse lights instead of a pinhole. The lenses satisfy the following property: rays of light reflected by a point P in the scene are refracted by lenses so that the rays can converge to a single point on the sensor (refer to figure 2.3 [26]). However, this property doesn't hold for all points in the scene. Points that are closer or further than point P to the image plane will be blurred because of out of focus [26]. Each lens has a specific distance for the scene to be in focus. This property is also related to a concept in photography called depth of field [26]. Only an interval of distance in which the scene can be captured in acceptable sharpness.

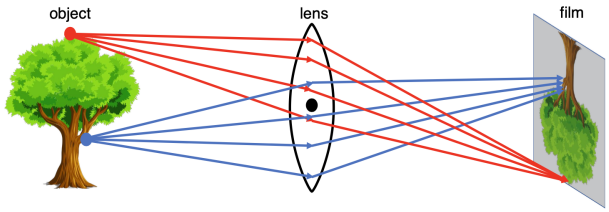


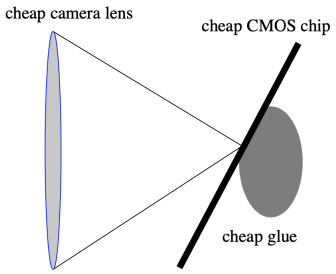
Figure 2.3: A setup of a simple lens model

2.2 The brown distortion model

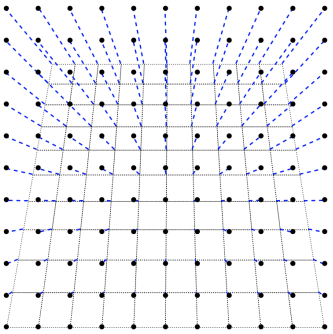
During the formation of images, there are typically two types of distortions. One is called optical distortions, and the other one is called perspective distortions [47]. It's important to distinguish those two different kinds of distortions and to understand the principles leading to them. Optical distortions are brought in by the imperfections of lenses, referring to an optical aberration by which physical straight lines appear curvy in images. It can be mitigated by refined design and production or post-processing softwares such as Lightroom and Photoshop [47]. However, perspective distortions are caused by the position of objects in the scene relative to the camera [47]. For example, the object close to the camera appears to be disproportionately large or distorted compared with objects in the background, which is a very normal occurrence. Therefore, only optical distortions need to be modeled and removed.

According to the work by Brown [7], a significant degree of optical distortions can be divided into two categories: tangential distortions and radial distortions, and all of them are caused by individually manufacturing flows. Tangential distortion occurs when the image plane isn't parallel to the lens. An example of tangential distortion is presented in figure 2.4 [2]. In figure 2.4a an image sensor that detects and converts optical information to images might be positioned behind a lens with cheap glue. As time goes by, the sensor loosens. The grid in figure 2.4b displays an image of a grid with tangential distortion. The black points indicate the correct positions the grid's corners should be, and the blue dashed lines between correspondences imply the tangential distortion.

As for radial distortion, it's caused because of individually manufacturing flaws of lenses. If given a camera with perfect lenses, the images captured would have no radial distortions. Physically straight lines in the scene shall appear straight in images if it's placed in front of the camera and is parallel with the image plane. When straight lines



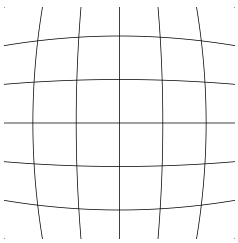
(a) Exemplary cheap camera schematics



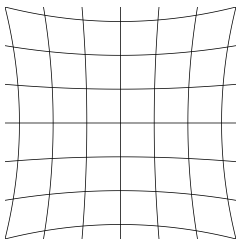
(b) Tangential Distortion

Figure 2.4: An example of tangential distortion

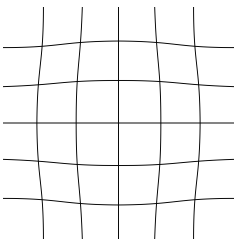
appear in images to be curved inwards in a shape like a barrel – the image magnification decreases, this kind of distortion is called barrel distortion (refer to figure 2.5a [14]) which is commonly seen on wide-angle lenses [47]. On the contrary, if the straight lines appear in images to be curved outwards from the center – the image magnification increases, this type of distortion is called pincushion distortion (refer to figure 2.5b [14]) which is commonly seen on telephoto lenses [47]. The optical magnification is defined as the ratio between the apparent size of an object in an image and its true size [22]. There is also another type of distortion called mustache distortion (refer to figure 2.5c [14]) which is a mixture of pincushion distortion and barrel distortions [47]. All of those distortions are called radial distortion which causes the image magnification to increase or decrease as a function of the distance to the optical axis. And radial distortion is caused by the imperfection of lenses – different portions of a lens have different focal lengths [26].



(a) Barrel Distortion



(b) Pincushion Distortion



(c) Mustache Distortion

Figure 2.5: Common cases of radial distortions

To achieve 3D scene reconstruction with high precision from images, an accurate camera distortion correction is the very first step. So the Brown-Conrady-Model [7]

used to model those distortions and to undistort images will be introduced in the following. Let (u_u, v_u) denote the coordinate of an undistorted point and (u_d, v_d) denote the coordinate of a distorted point. Then, the distortion can be modeled by function f:

$$u_d = f_u(u_u, v_u) \qquad v_d = f_v(u_u, v_u). \tag{2.8}$$

The undistortion can be modeled by function g:

$$u_u = g_u(u_d, v_d) \qquad v_u = g_v(u_d, v_d). \tag{2.9}$$

Based on the observation that radial distortions are symmetry relative to the principal point (u_o, v_o), the distortion radius can be defined as $r = \sqrt{(\bar{u}_d/f_x)^2 + (\bar{v}_d/f_y)^2}$ [2], where f_x and f_y are focal length in pixel units, $\bar{u}_d = u_d - u_o$ and $\bar{v}_d = v_d - v_o$. In the Brown-Conrady-Model, radial distortion is modeled by using 3 radial distortion coefficients (k_1, k_2, k_3) and the distortion radius. The correction function g_r [2] for radial distortion is defined as:

$$\begin{bmatrix} \hat{u}_u \\ \hat{v}_u \end{bmatrix} = \begin{bmatrix} \bar{u}_d \\ \bar{v}_d \end{bmatrix} (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.10}$$

To compensate for the effect of tangential distortion, two tangential distortion coefficients (p_1, p_2) are proposed and the correction function g_t [2] for tangential distortion is defined as:

$$\begin{bmatrix} \tilde{u}_u \\ \tilde{v}_u \end{bmatrix} = \begin{bmatrix} f_x(2p_1\bar{u}_d\bar{v}_d + p_2(r^2 + 2\bar{u}_d^2)) \\ f_y(2p_2\bar{u}_d\bar{v}_d + p_1(r^2 + 2\bar{v}_d^2)) \end{bmatrix} \tag{2.11}$$

Combining the function g_r in equation 2.10 and the function g_t in equation 2.11, the complete undistortion function g [2] is defined as:

$$\begin{bmatrix} u_u \\ v_u \end{bmatrix} = \begin{bmatrix} \bar{u}_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + f_x(2p_1\bar{u}_d\bar{v}_d + p_2(r^2 + 2\bar{u}_d^2)) \\ \bar{v}_d(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + f_y(2p_2\bar{u}_d\bar{v}_d + p_1(r^2 + 2\bar{v}_d^2)) \end{bmatrix} + \begin{bmatrix} u_o \\ v_o \end{bmatrix} \tag{2.12}$$

2.3 Camera calibration

Camera calibration is the process of obtaining the intrinsic and the extrinsic parameters of a camera which include [27]:

- f_x and f_y are focal length in pixel units with pixel size along the X and Y axes.
- Principal point (u_o, v_o) is usually at the center of an image.

- Skew coefficient defines the angle between the u and the v axes.

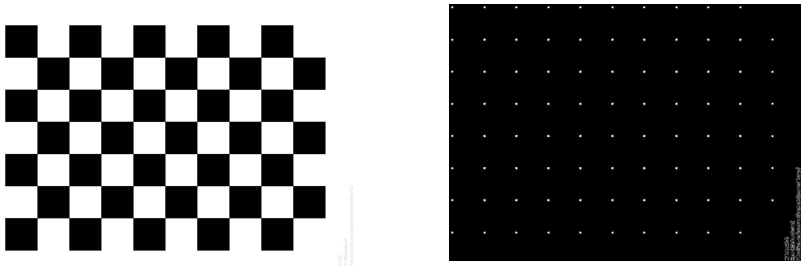
- Distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$ define the distortions introduced by lenses including radial and tangential distortions.

2.3.1 Distortion coefficients calculation

The first step of the calibration process is to determine the distortion coefficients so that the distorted images can be transformed to true perspective images – straight lines in the world are projected as straight lines in images. The chessboard pattern[1] (refer to the figure 2.6a) is one of the best choices, where its corners and interior intersection points can be easily localized in several orientations (horizontal, vertical and diagonals) by convolving images with the filter in table 2.1 which gives strong response (positive or negative depending on the type of corner) when centered at a chessboard corner [12]. Taking the absolute value of the filtered chessboard images gives an image where corners appear as white dots (refer to figure 2.6b).

-1	-1	-1	0	1	1	1
-1	-1	-1	0	1	1	1
-1	-1	-1	0	1	1	1
0	0	0	0	0	0	0
1	1	1	0	-1	-1	-1
1	1	1	0	-1	-1	-1
1	1	1	0	-1	-1	-1

Table 2.1: A 7×7 convolution filter that detects corners of the chessboard pattern



(a) A chessboard pattern image

(b) Convoluted chessboard pattern image with the filter in table 2.1

Figure 2.6: An example of convolution of the chessboard pattern

[1] You can find a chessboard pattern by OpenCV at https://github.com/opencv/opencv/blob/master/doc/pattern.png

Localizing a particular chessboard corner can be achieved by locating the point with a maximum filter response. Sub-pixel accuracy can be obtained by fitting it and its nearby responses with an upside-down paraboloid, and choosing the location with a maximum response of the paraboloid as the final position of that corner [12].

The chessboard corners in the world space can provide sets of points that are collinear vertically, horizontally, or diagonally. However, due to the optical distortions, those points being collinear in the world space might not be collinear in images. So for any set of such points, one can fit a straight line to them and calculate its summed residual error. Summing all the residual errors for each set of points, one can construct an error function E_d which measures the extent of distortion of images captured by that lens. Applying the undistortion function in equation 2.12 with an initial guess of parameters $(u_o, v_o, k_1, k_2, k_3, p_1, p_2)$, one can obtain the optimal value of those parameters with which the error function E_d is at its global minimum [12]. Then, the distorted images can be corrected by equation 2.12 with optimal distortion coefficients and be used for the further calibration process.

2.3.2 Intrinsic & extrinsic parameters calculation

Inputted with images of a chessboard shot from various angles and distances (refer to the figure 2.7), the calibration algorithm can calculate the values of the eleven-degree-of-freedom projection matrix P in equation 2.13 which can further be factored into the camera's six extrinsic parameters (rotation and translation of the camera relative to the chessboard) and five intrinsic parameters [12]. Normally, the calibration won't obtain a satisfying result unless more than 20 chessboard images are used. One of the most common way to calibrate a camera is the Direct Linear Transformation method [1, 35].

$$P = \begin{bmatrix} f_x & 0 & u_o & 0 \\ 0 & f_y & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

Assuming that the coordinate of scene points $\mathbf{X}_i$ and the coordinate of their projections $\mathbf{x}_i$ are known, and according to equation 2.7 and equation 2.13, camera calibration is to solve the following equations:

$$\lambda_i \mathbf{x}_i = P\mathbf{X}_i, \qquad i = 1, \ldots, n \tag{2.14}$$

where λ_i and P are the unknowns. The matrix P has 11 degrees of freedom, and each

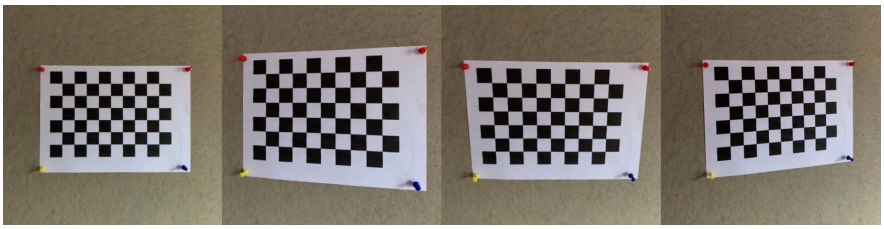


Figure 2.7: Examples of chessboard image

point projection introduce one additional unknown λ_i. Therefore, at least 6 ($3n \geq 11+n \rightarrow n \geq 6$) point-projections are needed to solve equation 2.14. To solve it, the Direct Linear Transformation method formulates a homogeneous linear system of equations and solves it by finding an approximate null space of the system matrix. If let p_i, $i = 1,2,3$, be 4×1 vectors containing the rows of matrix P, equation 2.14 can be rewritten as:

$$\begin{aligned} \mathbf{X}_i^T p_1 - \lambda_i x_i &= 0, \\ \mathbf{X}_i^T p_2 - \lambda_i y_i &= 0, \\ \mathbf{X}_i^T p_3 - \lambda_i &= 0, \end{aligned} \tag{2.15}$$

where $\mathbf{x}_i = (x_i, y_i, 1)$. In a matrix form, equation 2.15 can be rewritten as:

$$\begin{bmatrix} \mathbf{X}_i^T & \mathbf{0} & \mathbf{0} & -x_i \\ \mathbf{0} & \mathbf{X}_i^T & \mathbf{0} & -y_i \\ \mathbf{0} & \mathbf{0} & \mathbf{X}_i^T & -1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \lambda_i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.16}$$

Note that in equation 2.16, $\mathbf{X}_i$ are 4×1 vectors, and $\mathbf{0}$ is a 1×4 vector of zeros. Therefore, the left hand side is a 3×13 matrix. If all point projections are included, equation 2.16 would be extended as:

$$\underbrace{\begin{bmatrix} \mathbf{X}_i^T & \mathbf{0} & \mathbf{0} & -x_1 & 0 & \dots \\ \mathbf{0} & \mathbf{X}_i^T & \mathbf{0} & -y_1 & 0 & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{X}_i^T & -1 & 0 & \dots \\ \mathbf{X}_i^T & \mathbf{0} & \mathbf{0} & 0 & -x_2 & \dots \\ \mathbf{0} & \mathbf{X}_i^T & \mathbf{0} & 0 & -y_2 & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{X}_i^T & 0 & -1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \lambda_1 \\ \lambda_2 \\ \vdots \end{bmatrix}}_{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{2.17}$$

Here, M is the system matrix, and v is a vector containing unknowns. Solving this linear system equals to finding a non-zero vector in the nullspace of M. Since the scale of the solution is arbitrary, let it be constrained to $\| v \|^2 = 1$. Besides, due to the noise in the measurement of point projections, the linear system will not have an exact solution normally. Thus, finding the non-zero vector in the nullspace of M is converted to an optimization problem:

$$\underset{v}{arg\,min}_{\|v\|^2=1} \| Mv \|^2 \tag{2.18}$$

Note there are always at least two solutions to equation 2.18, because of $\| Mv \| = \| M(-v) \|$ and $\| v \| = \| -v \|$. So the solution which makes λ_i are all positive shall be selected. If let $f(v) = v^T M^T M v$ and $g(v) = v^T v$, equation 2.18 can be rewritten as:

$$\underset{v}{arg\,min}_{g(v)=1} f(v) \tag{2.19}$$

According to Lagrange Multiplier, the solution to equation 2.19 must fulfill

$$\bigtriangledown f(v) = \gamma \bigtriangledown g(v) \quad \leftrightarrow \quad 2M^T M v = \gamma 2v \quad \leftrightarrow \quad M^T M v = \gamma v \tag{2.20}$$

which means that v is an eigenvector of $M^T M$. Assume that $v_\star$ is an eigenvector with eigenvalue $\gamma_\star$, and insert equation 2.20 into $f(v)$.

$$f(v_\star) = v_\star^T M^T M v_\star = \gamma_\star v_\star^T v_\star \tag{2.21}$$

Since $\| v \| = 1$, the minimum of $f(v)$ equals to the smallest eigenvalue of $M^T M$, and the solution to equation 2.17 is the eigenvector of $M^T M$ with the smallest eigenvalue, which can be solved by Singular Value Decomposition.

Then, matrix P can be factorized by RQ-factorization into:

$$P = K \begin{bmatrix} R & t \end{bmatrix} \tag{2.22}$$

while K is the right triangular camera matrix, R is the rotation matrix and t is the translation vector.

Theorem 1 (RQ Factorization [35]) *If A is an $n \times n$ matrix then there is an orthogonal matrix Q and a right triangular matrix R such that*

$$A = RQ. \tag{2.23}$$

(If A is invertible and the diagonal elements are chosen positive then the factorization is unique.)

If let $P = \begin{bmatrix} A_{3\times 3} & \mathbf{a}_{3\times 1} \end{bmatrix}$, the camera matrix K and the rotation matrix R can be obtained by RQ-factorizing A. If

$$K = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix}, \quad A = \begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \end{bmatrix}, \quad \textit{and} \quad R = \begin{bmatrix} R_1^T \\ R_2^T \\ R_3^T \end{bmatrix} \tag{2.24}$$

while R_1, R_2, R_3 and A_1, A_2, A_3 are 3×1 vectors containing rows of R and A respectively. Then, we get

$$\begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \end{bmatrix} = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} R_1^T \\ R_2^T \\ R_3^T \end{bmatrix} = \begin{bmatrix} aR_1^T + bR_2^T + cR_3^T \\ dR_2^T + eR_3^T \\ fR_3^T \end{bmatrix} \tag{2.25}$$

Since rotation matrix R is orthogonal, and $A_3 = fR_3$ from the third row in equation 2.25, we get

$$\| R_3 \| = 1 \quad , f = \| A_3 \|, \quad \textit{and} \quad R_3 = \frac{1}{\| A_3 \|} A_3. \tag{2.26}$$

Now f and R_3 are known. The second row of equation 2.25 indicates that A_2 is a linear combination of two orthonormal vectors of R_2 and R_3. Therefore, the coefficient e is the length of projection of A_2 in the direction of R_3, being computed from

$$e = A_2^T R_3. \tag{2.27}$$

When e is known, coefficient d and R_2 can be computed from

$$dR_2 = A_2 - eR_3. \tag{2.28}$$

Same principle can be applied to the first row of equation 2.25, computing the coefficients a, b, c and R_1. Dividing each element in K by element $K_{3,3}$ obtains the resulting K. Translation t can be computed from

$$t = K^{-1}\mathbf{a} \tag{2.29}$$

Chapter 3

Features Detection & Matching

Chapter 2 has presented a camera model, an optical distortion model, and a camera calibration method which recovers the mapping between world points and their projections in images. During the calibration process, the coordinates of world points and their projections are assumed to be known. In this chapter, the techniques needed to detect those projections and to obtain their correspondences in different images will be presented, as the same scene is captured by a camera at various viewpoints, which is the essence of SfM which reconstructs based on traces of features.

The projection of a world point in an image is one pixel which is a scalar for a grayscale image or a vector for any other color models such as RGB, BGR, and HSV. Refer to [21] for more information about color models, their pros & cons, and the conversion methods among them. However, a single-pixel itself provides little information for effective detection and satisfying matching repeatability. Therefore, an aggregation of adjacent pixels and the pixel itself are used to denote the projection of a world point. Such aggregation is usually referred to as a feature. Features above a uniqueness threshold are called good features which lead to unambiguous matchings among images. The following sections would discuss the detection of features, the construction of descriptors for each feature, and features matching according to features' descriptors in detail. Two features detection methods, SURF, and ORB are introduced in this thesis.

3.1 SURF

Speeded-Up Robust Features (SURF) is a scale- and rotation-invariant detector and descriptor [4]. Before introducing SURF, some background information and development history of features detecting shall be introduced, which is helpful for the understanding of SURF.

The earlier widely used detector is probably the Harris Corner Detector [24] which is based on the eigenvalues of the second-order moment matrix. However, it's not scale-invariant. Later, Mikolajczyk and Schmid [37] refined it and proposed the Harris-Laplace Detector and the Hessian-Laplace Detector which detects features by Harris measure or the determinant of the Hessian matrix and selects scale by Laplacian of Gaussians. To speed up the calculation of the Laplacian of Gaussians, Lowe [33] proposed to approximate the Laplacian of Gaussians by the difference of Gaussians. Besides, according to the studying and comparing the existing detectors [38, 39], the Hessian-based detectors are more stable and repeatable than their Harris-based counterparts, and using the determinant of the Hessian matrix is better than its trace (the Laplacian).

Apart from detecting features, the descriptor of each feature is also needed for further matching process. Among all the descriptors proposed by far such as Gaussian derivatives [20] and moment invariants [40], the one introduced by Lowe [34] outperforms others [39]. The descriptor in [34] is called SIFT (Scale-Invariant Feature Transform) which calculates a histogram of locally oriented gradients around a feature point and stores the bins in a 128-dimensional vector. In SIFT, the further matching process is based on those 128-dimensional vectors. Using high dimensional descriptors, features can be more likely uniquely defined and, thus, producing more true-positive matchings. However, the high-dimensionality of descriptors is also a drawback of SIFT at the matching step, which slows down the matching speed.

Based on all the background information above, SURF was proposed to achieve more stable and repeatable detection by using the determinant of the Hessian matrix and faster matching speed by using 64-dimensional descriptors and the trace of the Hessian matrix.

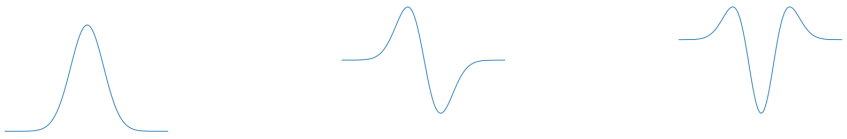
3.1.1 Detector

Hessian Matrix-based Detector

SURF [4] detects features at locations where the determinant of the Hessian matrix is maximum. The Hessian matrix is a square matrix of second-order partial derivatives of a scalar-valued function. Given a point $\mathbf{x} = (x, y)$ in an image I, the Hessian matrix $\mathcal{H}(\mathbf{x}, \sigma)$ of $\mathbf{x}$ at scale σ is defined as:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{3.1}$$

where $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of the Gaussian second derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ of point $\mathbf{x}$ in image I, and same principle to $L_{xy}(\mathbf{x}, \sigma)$ and $L_{yy}(\mathbf{x}, \sigma)$. The reason for selecting Gaussians as the convolution function for scale selection is that Gaussians are optimal for scale-space analysis [30, 31]. Refer to figure 3.1 for one-dimensional Gaussian distribution and its first and second derivatives. Figure 3.2 [4] presents discretized and cropped Gaussian second order partial derivatives in $y - (L_{yy})$ and $xy - L_{xy}$ directions, respectively, and their approximations, box filters, used in SURF.



(a) One-dimensional gaussian distribution

(b) First derivative of one-dimensional gaussian distribution

(c) Second derivative of one-dimensional gaussian distribution

Figure 3.1: One-dimensional Gaussian distribution and its first and second derivatives

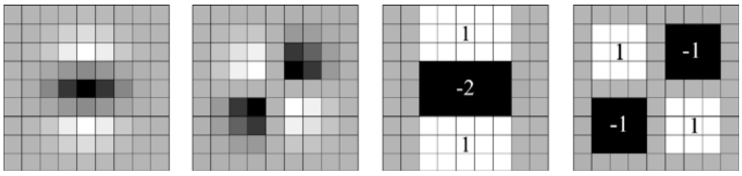


Figure 3.2: 9×9 box filters – approximations of a Gaussian with $\sigma = 1.2$

To speed up the convolution process of box filters, integral images [4] are computed before the convolution, allowing fast computation of any box type convolution. The

value for each element at $\mathbf{x} = (x, y)$ in an integral image I_Σ is the sum of intensities of all pixels in a rectangular area in the original image I formed by its top-left corner and $\mathbf{x}$.

$$I_\Sigma(\mathbf{x}) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j) \tag{3.2}$$

Once the integral image is calculated, the sum of intensities over any rectangular area can be calculated by three additions (refer to figure 3.3 [4]). Therefore, the calculation time is independent of the rectangle size.

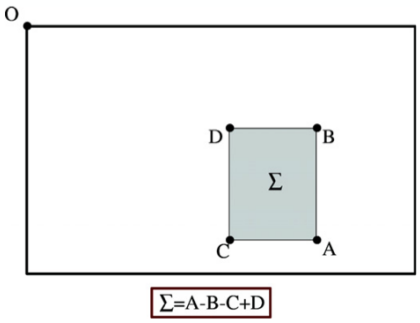


Figure 3.3: Calculation the sum of intensities over a rectangle area using integral image

If denote the response of box filters as D_{xx}, D_{yy} and D_{xy}, the determinant of approximated Hessian matrix is

$$det(\mathcal{H}_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2. \tag{3.3}$$

where weight w is used to balance the expression between the Gaussians and the approximated box filters,

$$w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} = 0.912\cdots \simeq 0.9, \tag{3.4}$$

where $|x|_F$ is the Frobenius norm [4]. The weight parameter changes according to the scale. But in practice, this parameter can be set as a constant because of its limited influence on the results. To guarantee a constant Frobenius norm for any filter size, the filter responses are normalized with respect to the filters' size, which is crucial for the scale space analysis. The approximated determinant of the Hessian represents the response of SURF detector in image at location $\mathbf{x}$ and is stored in a response map over different scales.

Features in the original image are extracted from response maps through a non-maximum suppression scheme (refer to figure 3.4) [44]. Each sample point is compared with its neighbors in the current response map and neighbors in the response map at the scale above and bellow. It is selected only when its response value is larger than all of its neighbors. The true maximum of response can be found by interpolating the response maps in scale and image space with the method proposed by Brown and Lowe [8].

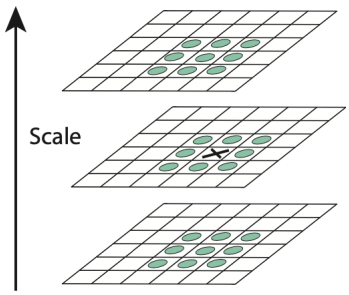


Figure 3.4: Non-maximum suppression as a local maximum search method

Scale Space Representation

The scale-space in SURF is analyzed by increasing the size of box filters, which saves the computation of integral images for the original image at different scales [4]. For example, the construction of the scale-space could start with 9×9 box filters. Then, filters with size 15×15, 21×21 et al. can be applied. For each new octave, the filter size increase is doubled. For example, the filter sizes in the second octave are 15, 27, 39, et al. The filter sizes in the third octave are 27, 51, 75, et al. Figure 3.5 presents an overview of the filter sizes in the first three octaves.

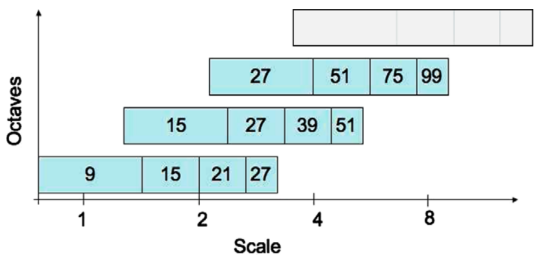


Figure 3.5: An overview of the filter sizes in the first three octaves

3.1.2 Descriptor

Orientation Assignment

Once SURF finished feature points' detection, it's ready to construct a descriptor for each feature point. The first step of constructing a descriptor is to calculate the feature's orientation so that features can be compared after orientation alignment, which is crucial for SURF's rotation-invariant property.

In SURF, the orientation of a feature point is obtained by calculating the Haar wavelet responses in x and y direction over a circular neighborhood of radius $6s$ around the feature point (refer to figure 3.6 [28]), where s is the scale at which the feature point is detected [4]. Figure 3.7 [4] presents the Haar wavelet filters whose side lengths are set to be $4s$.

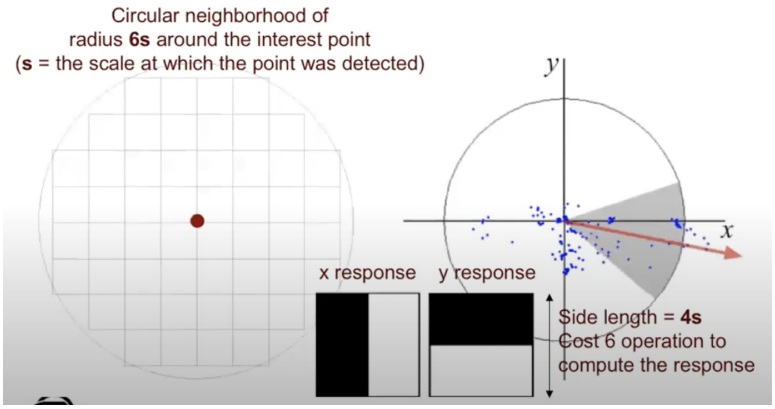


Figure 3.6: Orientation assignment

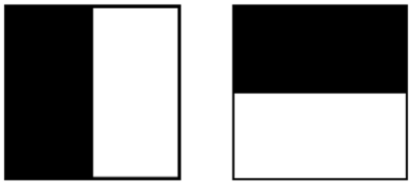


Figure 3.7: Haar wavelet filters in x and y direction. The dark parts indicate values of -1 and the bright parts indicate value of 1

Once the responses of Haar wavelet filters are calculated and weighted with a Gaussian ($\sigma = 2s$) centered at feature points, the responses are represented as points in a 2D coordinate system where the x-axis and the y-axis indicate responses in X direction and

Y direction, respectively (refer to figure 3.6) [4]. The orientation of a feature point is obtained by calculating the sum of responses, additions of vectors, in a rotating sector window of size $\frac{\pi}{3}$ and picking the longest such sum vector over all windows [4].

Descriptor Construction

To construct the descriptor of a feature point, a square region centered at the feature point and orientated along the feature's orientation has to be constructed. The region is further divided into 4×4 square sub-regions (refer to figure 3.8) [4]. Each sub-region has 5×5 regularly spaced pixels. Then, the Haar wavelet responses of each sub-region are calculated with a filter size of $2s$ and weighted with a Gaussian ($\sigma = 3.3s$) centered at the feature point. The responses of Haar wavelet filters and their absolute values are summed to form entries in the feature vector of a sub-region, $(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$, where d_x and d_y are responses in the x and y direction, respectively [4]. Figure 3.9 [4] presents an example of feature vectors for patches with different intensity landscape. Since each feature point has 4×4 such sub-regions, concatenate those feature vectors and the descriptor for a feature point is a 64 dimensional vector. The wavelet responses are invariant to illumination and invariance to contrast can be achieved by normalizing the descriptor into a unit vector.

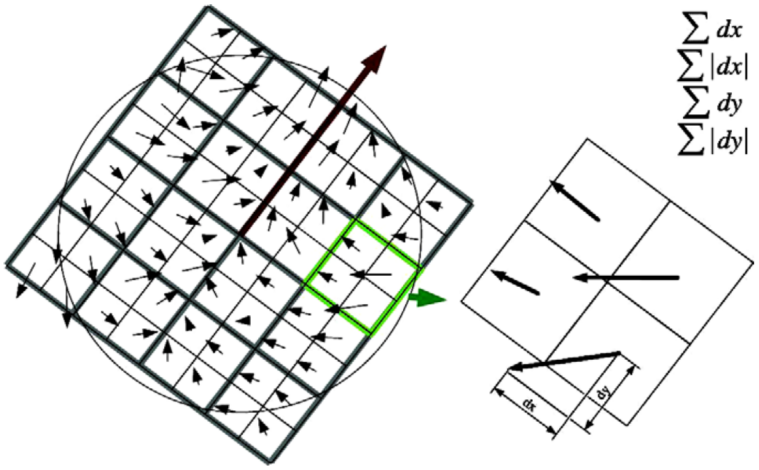


Figure 3.8: Descriptor construction

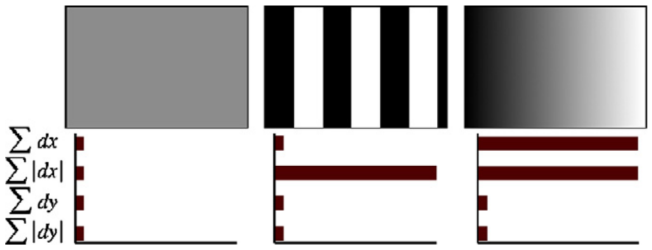


Figure 3.9: Feature vector

3.1.3 Fast indexing

To speed up the matching process, SURF has introduced fast indexing which compares the sign of the Laplacian, the trace of the Hessian matrix, of feature points. As typically features are found with blob-type structures and the sign of the Laplacian distinguishes bright blobs with the dark background from dark blobs with the bright background (refer to figure 3.10) [4], only those features that have the same type of contrast are further compared, which saves the computation for complex comparison [4].

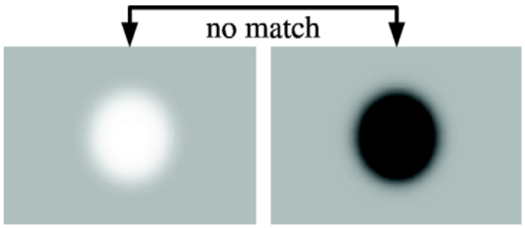


Figure 3.10: Fast indexing

3.2 ORB

Orientated FAST and Rotated BRIEF (ORB) is another scale- and rotation-invariant detector and descriptor which has a similar matching performance to SURF but is more computationally efficient [51]. Therefore, ORB is more capable of being used in real-time scenarios as this project intends to be. What's more, ORB is not patented like SIFT and SURF. It's free to use. The performance analysis of SURF and ORB will be presented in chapter 6.

3.2.1 Detector

The detector in ORB is built on the FAST feature detector [50]. However, FAST itself doesn't produce multi-scale features and isn't rotation invariant. So in ORB, FAST is augmented with image pyramid schemes for scale-space analysis [29] and Intensity Centroid [49] for features' orientation assignment.

oFAST

oFAST (FAST in ORB) detects corners (also known as features) by performing a segment test that considers a circle of pixels with a radius of 9 around the candidate point p. A candidate point would be picked as a corner if there are n pixels on the circle which are all brighter or all darker than point p plus a threshold t. For example, the original FAST detector considers a circle of sixteen pixels centered at p and classifies p as a corner if there are 12 contiguous pixels which are all brighter or all darker than point p plus a threshold t [50] (refer to figure 3.11).

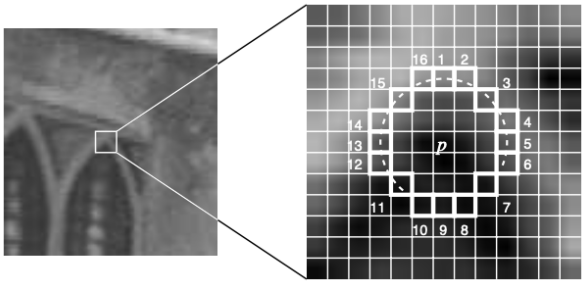


Figure 3.11: 12 point segment test corner detection in an image patch

As the FAST doesn't produce a measure of cornerness, the second step of oFAST is to order detected corners by employing a Harris corner measure [24], so that oFAST can produce a target N corners by setting a low enough threshold, and pick the top N points. To achieve scale-invariant, a scale pyramid of the input image is constructed and oFAST is performed at each level in the pyramid.

Orientation assignment by Intensity Centroid

A feature's orientation in ORB is assigned by the offset of its intensity centroid from its center (remember that a feature is a small image patch centered at the feature point and in ORB the patch size equals to the radius in oFAST). The intensity centroid of a

feature is defined by Rosin as [49]:

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{3.5}$$

where m_{pq} are moments of the feature:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \qquad p,q \in \{0,1\}. \tag{3.6}$$

The vector from the feature's center O to its centroid C, $\vec{OC}$, points toward the orientation of that feature. The angle of rotation can be computed by

$$\theta = atan2(m_{01}, m_{10}), \tag{3.7}$$

where $atan2$ is the quadrant-aware version of $arctan$.

3.2.2 Descriptor

The descriptor in ORB, rBRIEF, is built on the BRIEF descriptor [9]. To integrate orientation extracted by the ORB detector, ORB computes descriptors based on a rotated set of binary intensity tests that are selected by a greedy search algorithm [51]. In the ORB paper [51], BRIEF combined with the rotation is named steered BRIEF in which the binary tests are picked randomly according to Gaussian distributions. If the binary tests are constructed by the greedy search algorithm instead of random selection, the steered BRIEF is called rBRIEF.

Brief overview of BRIEF

The BRIEF descriptor [9] is a bit string description of an image patch constructed from binary intensity tests of a predefined set of n point pairs. Given a smoothed image patch p and a predefined set of point pairs $\{\{p_{1,1}, p_{1,2}\}, \ldots, \{p_{n,1}, p_{n,2}\}\}$, the feature vector of image patch p is defined as a vector of n binary tests:

$$f_n(p) := \sum_{1 \le i \le n} 2^{i-1} \tau(p; p_{i,1}, p_{i,2}) \tag{3.8}$$

where a binary test τ is defined as:

$$\tau(p; p_{i,1}, p_{i,2}) = \begin{cases} 0, & if\ p(p_{i,1}) < p(p_{i,2}) \\ 1, & if\ p(p_{i,1}) \geq p(p_{i,2}) \end{cases} \tag{3.9}$$

and $p(p_{i,1})$ is the intensity of p at point $p_{i,1}$. Points $p_{i,1}$ and $p_{i,2}$ can be selected randomly according to a Gaussian distribution centered at center of a patch. Refer to [9] for more spatial arrangement of the binary tests. Note that the spatial arrangement is only constructed once and applies to all features.

In ORB, the length of a feature vector is set to $n = 256$ and the patch size is set to 31×31. Each test point is selected from center points of smoothed 5×5 windows in that patch [9].

steered BRIEF

Given a set of point pairs $\{\{p_{1,1}, p_{1,2}\}, \ldots, \{p_{n,1}, p_{n,2}\}\}$, its spatial arrangement S can be defined as:

$$S = \begin{bmatrix} x_1, \ldots, x_{2n} \\ y_1, \ldots, y_{2n} \end{bmatrix} \tag{3.10}$$

To steer BRIEF according to the orientation of feature points, the spatial arrangement S is transformed by the corresponding rotation matrix of θ in equation 3.7:

$$S_\theta = R_\theta S \tag{3.11}$$

The feature vector of steered BRIEF is defined as:

$$g_n(p) := f_n(p) | (p_{i,1}, p_{i,2}) \in S_\theta \tag{3.12}$$

rBRIEF

Compared with BRIEF, steered BRIEF faces a variance loss problem [9]. As high variance makes descriptor more discriminative, the authors of ORB has devised a greedy search algorithm for the creation of spatial arrangement to recover the variance loss in steered BRIEF and reduce the correlation among binary tests.

The authors of ORB has set up a training set of $300k$ feature points from images in the PASCAL 2006 set [18]. The image patch size is 31×31 and each test point comes from a smoothed 5×5 window of that patch. Thus, there are $\binom{(31-5)^2}{2}$ possible

point pairs in total. The binary test for each point pair has been performed $300k$ times so that there is a bit string for each point pair from which the mean and variance of a point pair can be computed. Based on the bit strings of all point pairs, the optimal spatial arrangement can be constructed by selecting an optimal set of point pairs using the greedy search algorithm [9]:

1. Run test against all point pairs in all training patches.

2. Order the point pairs by their distance from the mean of their bit strings to 0.5, forming a vector T.

3. Greedy search:

   (a) Put the first point pair into the result vector R and remove it from T.

   (b) Take the next point pair from T, and compare its bit string against those of all point pairs in R. If its absolute correlation is greater than a threshold, discard it; else add it into R.

   (c) Repeat the previous step until there are 256 point pairs in R. If the length of the final R is less than 256, increase the threshold, and try again.

3.3 Matching schemes

Once features are extracted by a feature detection method and represented by their descriptors, it's ready to find matches of features between different images which is the fundament for many computer vision problems including scene reconstruction. Features matching problem can be defined as finding the closest neighbor for a descriptor vector in a high-dimensional space [43]: given a set of descriptors $D = \{d_1, \ldots, d_n\}$ in a vector space R^d, those descriptors can be pre-processed so that for a query descriptor $q \in R^d$, its closest neighbor in R^d can be found efficiently. Normally, R^d is assumed to be a Euclidean vector space which is suitable for most problems in computer vision. The distance measurement commonly used includes *NORM_L1*, *NORM_L2*, and *NORM_HAMMING* [41]. *NORM_HAMMING* is used for binary bit string based descriptors like ORB. Several matching schemes would be presented in this section, including the brute force linear search method and variations of KD-trees.

3.3.1 Linear search

For finding the nearest neighbor of query descriptor q, linear search checks each element in set D and return the closet [41]. The maximum computational complexity for each linear search is $O(n)$ where n is the size of set D, so that linear search is suboptimal for a descriptor set D having a big size.

3.3.2 KD-tree

As linear search is too costly for big size descriptor sets, algorithms that perform approximate nearest neighbor search and return near-optimal neighbors gain more attention [43]. Compared with linear search, such approximate algorithms can be orders of magnitude faster, while returning near-optimal accuracy. One such algorithm is called KD-tree [5].

KD-tree is a balanced binary search tree. The elements stored in the KD-tree are descriptors in high-dimensional space R^d. At each level, the dataset is split into halves by a hyperplane orthogonal to a chosen dimension at a threshold value. Usually, the split is at the median of a dimension on which the dataset has the greatest variance. By comparing the query vector with the partitioning value at a node, it can be determined which partition the query vector belongs to. Each leaf represents a single descriptor in the dataset. Depending on the implementation of the KD-tree, the leaf node may contain more than one descriptor. The height of the tree will be $log_2 n$, where n is the size of the descriptor dataset. Thus, given a query vector, a descent down the tree to a leaf node needs only $log_2 n$ comparisons. It should be noted that each node in the tree represents a cell in space R^d, as it represents a collection of descriptors that are stored in its children nodes.

However, the first leaf encountered in the searching process might not be the true nearest neighbor to the query vector, so that several backtracking processes are needed to search for better candidates. For example, in figure 3.12 [54], for query vector q, the first leaf it will encounter in the KD-tree is the descriptor in cell 1, as the query vector is classified into cell 1. However, its true closet neighbor is the descriptor in the cell 3. Thus, priority search is proposed, in which cells are searched according to their distance to the query vector. The searching process terminates when there are no more cells left not being searched within a predefined distance to the query vector.

The true nearest neighbor to the query vector can only be assured when all descriptors in dataset D have been searched. Thus for the true nearest neighbor in KD-tree,

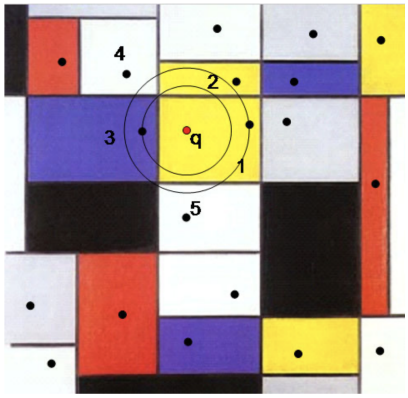


Figure 3.12: Priority search of a KD-tree

the predefined distance must be large enough to cover all cells. However, the searching time for a large size dataset D might be unacceptable under some scenarios, so that setting a small searching distance is fine as long as the returned neighbor is the exact nearest neighbor within an acceptable probability.

One problem of the KD-tree with the increasing searching distance is its diminishing returns [54]. The accuracy increment to find the true nearest neighbor decreases with the increasing searching distance. To address this problem, the following strategy was proposed:

1. Create m KD-trees with different structures so that the searches in different KD-trees are independent.

2. Limit the maximum of nodes n to be searched and break the search into simultaneous searches among all m KD-trees. On average, n/m searches would be performed in each tree.

3. Perform Principal Component Analysis on the dataset and align the dataset to the axes represented by its eigenvectors. The dataset would be split along the principal axes.

3.3.3 NKD-tree

NKD-tree [54] is similar to the previous multiple KD-trees method. In NKD-tree, multiple searching trees are created using rotated datasets RD. The principal (regular) KD-tree is constructed without any rotation $R = I$. The NKD-tree method is based on

this assumption that by rotating the dataset, the resulting searching trees have different structures and cover a different set of dimensions compared with the principal tree [54]. For searching trees constructed with rotated datasets, the query vector must be also rotated with the corresponding rotation matrix. The distance in the principal tree is defined as $\| q - d_i \|$, while the distance in the other trees is defined as $\| Rq - Rd_i \|$. The best one from the all searching trees is selected as the final nearest neighbor to the query vector q .

3.3.4 RKD-tree

In the standard KD-tree, splittings are conducted at dimensions where the dataset has large variances. However, in reality, data variances are similar in many dimensions [54]. Thus, in RKD-tree, multiple searching trees are created by splitting the dataset at a random dimension among those that have large variances at each level. In this way, the dataset stays in the original space, and the computation to calculate the projections of the descriptor dataset is saved. The authors of RKD-tree claim that RKD-tree performs as well as NKD-tree [54].

3.3.5 PKD-tree

In PKD-tree [54], the descriptor dataset is projected to a lower-dimensional space spanned by its top k eigenvectors. The searching trees are constructed based on the descriptors' projections in the lower dimensional space. Thus, the depth of each searching tree is k. Multiple searching trees are created by rotating those projections. Once the searching trees are created, the projection data can be discarded. The distance between the query vector and each descriptor is still measured in the original space.

3.3.6 Lowe's distance ratio test

Once the nearest neighbor of the query vector q is found in the descriptor dataset D, one can claim that the best match for q in D has been found. However, it might not be the true match for q due to the low uniqueness of the feature of q or the descriptor dataset has no records of feature q. To filter the matches, Lowe [34] has proposed the distance ratio test to eliminate the false matches. The distance ratio is computed by dividing the distance of the closet neighbor by the distance of the second closet neighbor. It's a good match when the ratio is below a threshold. Figure 3.13 [34]

illustrate the probability that a match is correct based on the distance ratio test. False matches can be filtered out by lowering the threshold value of the distance ratio.

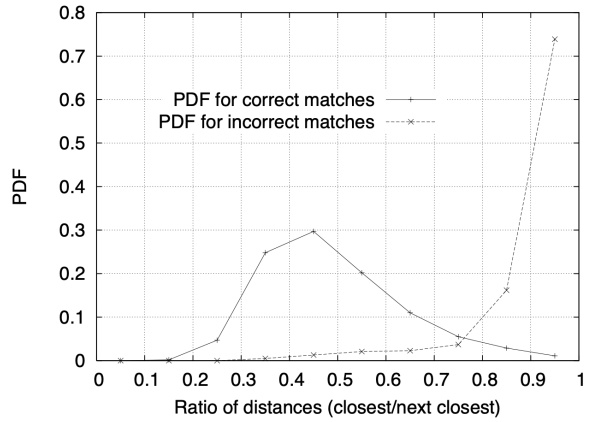


Figure 3.13: Probability density function for a correct match and a incorrect match based on the distance ratio test

Chapter 4

Pose Estimation

After introducing the prerequisite knowledge, camera model in chapter 2 and features detection and matching in chapter 3, for scene reconstruction, it's time to explore the scene reconstruction algorithm itself – SfM in this thesis.

SfM generates the shape model of an object by taking a sequence of images of that object. For example, in the figure 4.1 [23], a calibrated camera takes pictures of a cube at pose $P1$, then at poses $P2$ and $P3$. The red point on the cube appears in those three images. By constructing a triangle between two out of the three images, the red point's depth information relative to a camera pose can be calculated. This chapter would focus on the first step of SfM – pose estimation, estimating the rotation and translation between two poses of a camera such as from pose $P1$ to pose $P2$ and from pose $P2$ to pose $P3$ which is required for triangulation, as the shape of a triangle can be determined by two known internal angles recovered by the rotation between poses and a triangle can be fully determined by two known internal angles with a length-known edge between them recovered by the translation between poses. Method for calculating depths of features, forming a local point cloud, relative to a camera pose, and registration of local point clouds into one will be presented in section 5.1 and section 5.2 respectively.

4.1 Fundamental matrix

To estimate the pose of the camera at each point [45], one can treat each consecutive pair of images as a stereo problem. If given a world point P, denote its coordinate in the camera coordinate system at pose 1 and pose 2 as X_L and X_R respectively. Assuming that the rotation and translation between pose 1 and pose 2 is R and t, the relationship

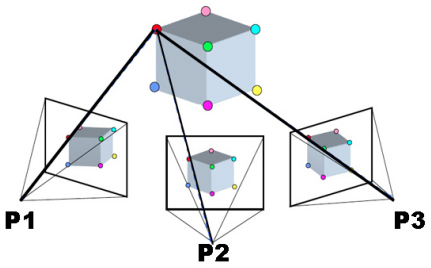


Figure 4.1: An example of image sequence used in shape from motion

between X_L and X_R can be defined as:

$$X_R = RX_L + t \tag{4.1}$$

Multiplying both side with t and X_R consecutively using the cross product and the dot product:

$$\begin{aligned} t \times X_R &= t \times RX_L + t \times t \\ X_R \cdot (t \times X_R) &= X_R \cdot (t \times RX_L + 0) \\ 0 &= X_R \cdot (t \times RX_L) \\ 0 &= X_R^T [t]_\times RX_L \end{aligned} \tag{4.2}$$

while $[t]_\times$ is the matrix representation of the cross product with t. Denote $[t]_\times R$ as E, the last row in equation 4.2 becomes

$$X_R^T E X_L = 0 \tag{4.3}$$

where E is called the essential matrix. Use intrinsic parameters matrix K in equation 2.6 and integrate it into equation 4.3:

$$\begin{aligned} (K^{-1}\lambda_r x_r)^T E (K^{-1}\lambda_l x_l) &= 0 \\ \lambda_r \lambda_l x_r^T K^{-T} E K^{-1} x_l &= 0 \\ x_r^T K^{-T} E K^{-1} x_l &= 0 \end{aligned} \tag{4.4}$$

while x_l and x_r are projections of X_L and X_R in images, λ_l and λ_r are depths of X_L and X_R relative to the camera at two poses, and K^{-T} means the transpose of the inverse of

matrix K. Denote $K^{-T}EK^{-1}$ as F, the last row in equation 4.4 becomes

$$x_r^T F x_l = 0 \tag{4.5}$$

where F is called the Fundamental Matrix which can be calculated by using some feature matches between images. Then, the rotation matrix and translation vector between the corresponding poses can be estimated from the fundamental matrix, as

$$F = K^{-T}[t]_\times R K^{-1}. \tag{4.6}$$

4.2 Epipolar geometry

Equation 4.5 reveals the Epipolar Geometry [15] that describes the relationship between the three-dimensional world points and their projections onto two-dimensional images which leads to constraints between the image points. For example, in figure 4.2 [15], the point X in the scene has two projections x_l into the left view and x_r into the right view. e_l and e_r are epipoles, the projection of the other camera's optical center. The Epipolar Geometry tells us that each projection x_l in the left view maps to a line $e_r x_r$ in the right view and vice versa.

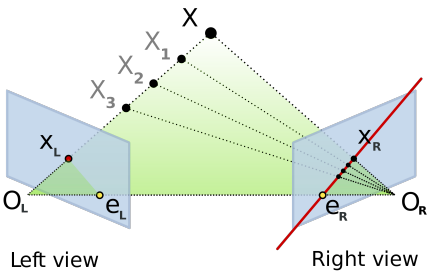


Figure 4.2: Epipolar Geometry

If given the fundamental matrix F, any feature matches between two images can be expressed as:

$$x_r^T F x_l = 0 \;\rightarrow\; \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix}^T F \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = 0 \tag{4.7}$$

If given F and x_l, the equation 4.7 can be rewritten as:

$$au_r + bv_r + c = 0 \tag{4.8}$$

where a, b and c are some constants. Even though the exact coordinate of x_r can't be recovered, it does return a line where x_l's correspondence must lie on.

4.3 8-point algorithm

The fundamental matrix F is a 3×3 matrix. So, theoretically, 8 point correspondences at least [25] are needed to obtain the fundamental matrix, if don't care about the scaling. If given a feature mach $x_l = (u, v, 1)$, $x_r = (u', v', 1)$ and the fundamental matrix F, equation 4.5 can be rewritten as:

$$uu'F_{11} + uv'F_{21} + uF_{31} + vu'F_{12} + vv'F_{22} + vF_{32} + u'F_{13} + v'F_{23} + F_{33} = 0 \tag{4.9}$$

With a set of feature matches, formulating each match into equation 4.9 resulting a system of equations:

$$Af = 0 \tag{4.10}$$

where each row in A is a vector of

$$(uu', uv', u, vu', vv', v, u', v', 1) \tag{4.11}$$

and vector f is

$$(F_{11}, F_{21}, F_{31}, F_{12}, F_{22}, F_{32}, F_{13}, F_{23}, F_{33}) \tag{4.12}$$

As the fundamental matrix only needs to be determined up to a scale, the 8-point algorithm can be defined as

$$\underset{f}{arg} \min_{\|f\|=1} \| Af \| \tag{4.13}$$

where $\| f \|$ is the norm of f[1]. Because of the constraint on f, matrix A should have rank 8 ideally. However, due to the inaccuracies of the measurement of feature points, matrix A has rank 9 in reality. According to Lagrange Multiplier, the solution to this problem is the eigenvector corresponding to the least eigenvalue of $A^T A$, which can be solved by the Singular Value Decomposition (SVD) [3].

[1] An alternative constraint is to set $F_{33} = 1$.

One important property of the fundamental matrix is that it is singular and has rank 2, because of the fact that the left and right null-spaces of matrix F are two epipoles in homogeneous coordinate. However, the matrix F obtained from A^TA will not have rank 2 in general. To enforce this constraint, matrix F is replaced by F' that minimizes the Frobenius norm $\| F - F' \|$ with the constraint of $det(F') = 0$, which can be solved by the Singular Value Decomposition. If let $F = UDV^T$, matrix D is a diagonal matrix $D = diag(\sigma_1, \sigma_2, \sigma_3)$ where $\sigma_1 \geq \sigma_2 \geq \sigma_3$. Then, let $F' = UD'V^T$ where $D' = diag(\sigma_1, \sigma_2, 0)$. This method was proposed by Tsai and Huang [58] and has been proven to minimize $\| F - F' \|$.

4.4 Normalized 8-point algorithm

In practice, the standard 8-point algorithm is not precise. The distance between point x_r and its corresponding epipolar line $Fx_l = 0$ can be up to 10 pixels. To reduce the error, a refined version of the 8-point algorithm was proposed by Hartley [25] and called the Normalized 8-point Algorithm which performs as well as other best iterative algorithms, runs 20 times faster, and is easier to implement.

The reason for low precision of the standard 8-point algorithm is that matrix A^TA is ill-conditioned for SVD, which can be solved by applying a transformation of feature points' coordinates before the 8-point algorithm for calculating the fundamental matrix [25]. The transformation is summarized as follows:

1. Feature points are translated relative to the center of the image, as generally the coordinate of a feature point is relative to the top-left corner of the image.

2. Feature points are then scaled so that the average distance from them to the origin is equal to $\sqrt{2}$.

3. The transformation is applied to each image independently.

If given feature point $x = (u, v, 1)$, the transformation can be defined as

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c_u \\ 0 & 1 & -c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = T \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.14}$$

where c_u and c_v are the mean coordinates. In this way, the fundamental matrix F_n is calculated from normalized coordinates. To make it usable in the original coordinates,

the fundamental matrix F_n obtained from normalized features has to be de-normalized:

$$F = T_r^{-T} F_n T_l \tag{4.15}$$

where T_l and T_r are transformations for feature points in the left and the right views respectively. Ultimately, the new fundamental matrix F gives a good encapsulation of the epipolar geometry.

4.5 RANSAC

The fundamental matrix is computed from feature matches between two images which are unlikely to be perfect matches. Due to the outliers, false matches, or inaccurate matches, in the feature matches, the least-squares regression alone is not sufficient for computing the fundamental matrix. A better solution is to integrate RANSAC [19] into the calculation of the fundamental matrix in which the outliers are excluded.

In this scheme, some number of feature matches (8 or a number slightly greater than 8) are iteratively selected from all matches to compute a fundamental matrix using the normalized 8-point algorithm. After each computation of the fundamental matrix, the number of inliers is calculated, which "agree" with the computed fundamental matrix. A match is defined as an inlier if the distance from the feature point x_r to its epipolar line $Fx_l = 0$, vice versa, is less than a threshold. The fundamental matrix with the largest number of inliers is returned as the best estimation.

4.6 Rotation & translation

Once the fundamental matrix is gained, the rotation matrix and translation vector between two poses can be estimated by [55] using equation 4.6 as

$$F = K^{-T}[t]_{\times} R K^{-1} \tag{4.16}$$

where K is the camera matrix obtained by camera calibration, K^{-T} means the transpose of the inverse of the matrix K, t is the translation vector and R is the rotation matrix. Left multiplying F with K^T and right multiplying F with K leads to the essential matrix

$$E = [t]_{\times} R \tag{4.17}$$

Taking EE^T yields

$$EE^T = [t]_\times RR^T [t]_\times^T \tag{4.18}$$

Since rotation matrix R is a orthonormal matrix, $RR^T = I$. If given $t = [t_1, t_2, t_3]$,

$$[t]_x = \begin{bmatrix} 0 & -t_3 & t_2 \\ b_3 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{bmatrix} \tag{4.19}$$

and

$$EE^T = [t]_\times [t]_\times^T = \begin{bmatrix} t_2^2 + t_3^2 & -t_1 t_2 & -t_1 t_3 \\ -t_1 t_2 & t_1^2 + t_3^2 & -t_2 t_3 \\ -t_1 t_3 & -t_2 t_3 & t_1^2 + t_2^2 \end{bmatrix} \tag{4.20}$$

Then, $Tr(EE^T) = 2(t_1^2 + t_2^2 + t_3^2) = 2 \parallel t \parallel^2$.

4.6.1 Translation

To obtain the rotation matrix R and the unit length translation vector t, the essential matrix needs to be normalized first by dividing it with $\parallel t \parallel$:

$$E_n = \frac{E}{\parallel t \parallel} = \frac{E}{\sqrt{\frac{Tr(EE^T)}{2}}} \tag{4.21}$$

Using equation 4.20, the absolute value of entries in the translation vector can be computed by

$$\begin{aligned} |t_1| &= \sqrt{1 - [E_n E_n^T]_{11}} \\ |t_2| &= \sqrt{1 - [E_n E_n^T]_{22}} \\ |t_3| &= \sqrt{1 - [E_n E_n^T]_{33}} \end{aligned} \tag{4.22}$$

where $[E_n E_n^T]_{ij}$ denotes the entry of $E_n E_n^T$ at the i^{th} row and j^{th} column. The sign of the other two entries in t is inferred by the sign of the entry which has the largest absolute value using the off-diagonal entries of EE^T in equation 4.20. Suppose t_1 has the largest absolute value. Then, $t_1 = \pm\sqrt{1 - [E_n E_n^T]_{11}}$, $t_2 = -[E_n E_n^T]_{12}/t_1$ and $t_3 = -[E_n E_n^T]_{13}/t_1$.

4.6.2 Rotation

If given E_n, the cofactor matrix of E_n is

$$C_n = \begin{bmatrix} \begin{vmatrix} e_{22} & e_{23} \\ e_{32} & e_{33} \end{vmatrix} & -\begin{vmatrix} e_{21} & e_{23} \\ e_{31} & e_{33} \end{vmatrix} & \begin{vmatrix} e_{21} & e_{22} \\ e_{31} & e_{32} \end{vmatrix} \\ -\begin{vmatrix} e_{12} & e_{13} \\ e_{32} & e_{33} \end{vmatrix} & \begin{vmatrix} e_{11} & e_{13} \\ e_{31} & e_{33} \end{vmatrix} & -\begin{vmatrix} e_{11} & e_{12} \\ e_{31} & e_{32} \end{vmatrix} \\ \begin{vmatrix} e_{12} & e_{13} \\ e_{22} & e_{23} \end{vmatrix} & -\begin{vmatrix} e_{11} & e_{13} \\ e_{21} & e_{23} \end{vmatrix} & \begin{vmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{vmatrix} \end{bmatrix} \tag{4.23}$$

where e_{ij} denotes the entry of E_n in the i^{th} row and in the j^{th} column. If let e^i be the i^{th} column of E_n, the cofactor matrix of C_n can be rewritten as

$$C_n = \begin{bmatrix} e^2 \times e^3 & e^3 \times e^1 & e^1 \times e^2 \end{bmatrix} \tag{4.24}$$

E_n can also be expressed in terms of cross-products as:

$$E_n = [t]_\times R = \begin{bmatrix} [t]_\times r^1 & [t]_\times r^2 & [t]_\times r^3 \end{bmatrix} = \begin{bmatrix} t \times r^1 & t \times r^2 & t \times r^3 \end{bmatrix} \tag{4.25}$$

where r^i is the i^{th} column of rotation matrix R. Substituting from equation 4.25 into equation 4.24 yields

$$\begin{aligned} C_n &= \begin{bmatrix} (t \times r^2) \times (t \times r^3) & (t \times r^3) \times (t \times r^1) & (t \times r^1) \times (t \times r^2) \end{bmatrix} \\ &= \begin{bmatrix} (t \cdot (r^2 \times r^3))t & (t \cdot (r^3 \times r^1))t & (t \cdot (r^1 \times r^2))t \end{bmatrix} \\ &= \begin{bmatrix} (t \cdot r^1)t & (t \cdot r^2)t & (t \cdot r^3)t \end{bmatrix} \\ &= t \begin{bmatrix} (t \cdot r^1) & (t \cdot r^2) & (t \cdot r^3) \end{bmatrix} \\ &= tt^T \begin{bmatrix} r^1 & r^2 & r^3 \end{bmatrix} \\ &= tt^T R \end{aligned} \tag{4.26}$$

Since a well-known property of skew symmetric matrix,

$$tt^T = I_3 + [t]_\times^2. \tag{4.27}$$

Substituting equation 4.27 into equation 4.26 yields

$$C_n = (I_3 + [t]_\times^2)R = R + [t]_\times^2 R = R - [t]_\times^T [t]_\times R = R - [t]_\times^T E \iff R = C_n + [t]_\times^T E \quad (4.28)$$

Since translation vector t is sign-ambiguous, there are two possible solutions of rotation matrix R by flipping the sign of $[t]_\times^T$:

$$R = C_n \pm [t]_\times^T E \quad (4.29)$$

4.6.3 Cheirality constraint

Because of the sign-ambiguity of the translation vector, there are four combinations of the rotation matrix and the translation vector, while only one of them is the true configuration. To identify which one is the true one, cheirality constraint [61] is imposed in which the scene points should be in front of the cameras.

Given one feature match and the rotation and the translation between two poses, a triangle of two feature points and their corresponding world point can be constructed. Only the pose estimation through which the world point has a positive depth relative to the camera at two poses is the correct estimation like the left subfigure in figure 4.3 [55]. The four possible pose estimations recovered from the essential matrix are presented in figure 4.3 and figure 4.4 [55].

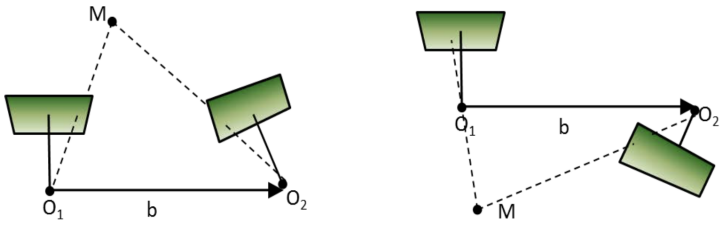


Figure 4.3: Camera orientation with respect to the scene point for “positive” translation vector

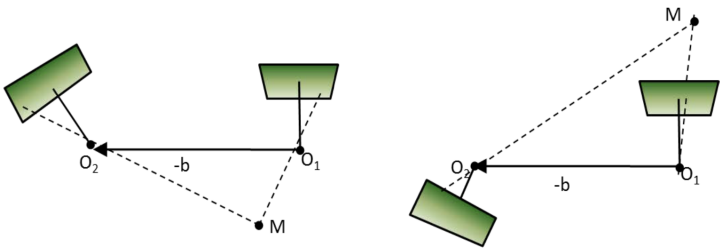


Figure 4.4: Camera orientation with respect to the scene point for “negative” translation vector

Chapter 5

Point Cloud Construction & Registration

5.1 Point cloud construction

Chapter 4 has discussed the method to estimate the pose change of a camera between two poses, based on which the triangulation for each feature match can be constructed and, then, the depth information for a feature point can be obtained by depth searching according to the reprojection error and by the least-squares method proposed in this thesis which is almost twenty times faster than the depth searching method while providing comparable reconstruction results.

The following subsections will start from the cornerstone, triangulation, for both methods and then introduce two methods individually.

5.1.1 Triangulation

The shape from motion algorithm can be regarded as a set of consecutive stereo visions. In each stereo vision, the triangulation can be constructed for each feature match, if given the rotation and translation between camera poses, as figure 5.1.

In figure 5.1, P_L and P_R are coordinates of scene point P in the camera coordinate system at two poses and p_l and p_r are projections of scene point P in images captured by the camera at two poses. The orientation difference between two poses is expressed by the rotation matrix R and the translation vector T which can be estimated using the method presented in chapter 4. Points p_l, p_r and P form a triangle which will be used

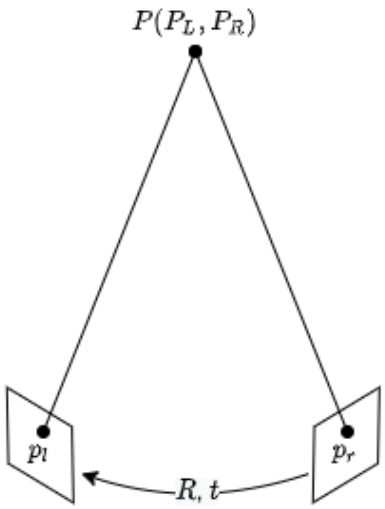


Figure 5.1: Stereo vision

for depth searching. The relationship between point P_L and point P_R is formulated as

$$P_L = RP_R + T \tag{5.1}$$

As the translation vector t extracted from the essential matrix is a unit-length vector, the true translation vector equals to the unit-length vector multiplied by a scalar k. Equation 5.1 can be rewritten as

$$P_L = RP_R + kt \tag{5.2}$$

Using the camera model expressed by equation 2.6, the relationship between point p_l and p_r is formulated as

$$K^{-1}\lambda_l p_l = R(K^{-1}\lambda_r p_r) + kt \tag{5.3}$$

where K is the camera matrix which transform P_L and P_R to p_l and p_r, and λ_l and λ_r are depths of P_L and P_R relative to the camera at two poses.

Note that the translation vector recovered from the essential matrix is only a unit-length vector instead of the true translation between poses, so that the triangle constructed by the scene point and its projections in the camera at two poses, like the one in figure 5.1, is not fully determined. Only the shape of the triangle is determined. Thus, the depths of feature points computed in shape from motion are only up to a scale. Only the shape of the scene is reconstructed through shape from motion.

5.1.2 Reprojection error

The reprojection error method for depth searching is a local stereo algorithm [48] which is generally much faster than its global counterpart. The local stereo algorithm identifies corresponding pixels only based on the correlation of local image patches. Given the coordinate of a feature point, x_l in the left image and all the possible depth values $\mathcal{D}$ for x_l, we can calculate all the possible coordinates of the corresponding feature point x_r in the right image. Then, the correlation coefficients of x_l and all its candidates x_rs can be calculated. The candidate with the smallest Zero-mean Normalized Cross-Correlation (ZNCC) [13] score value would be selected as the true corresponding point, and the depth value $d \in \mathcal{D}$ used to calculate that ZNCC score would be the depth information for feature point x_l.

Let f be the function which projects a three dimensional point in the scene to a pixel in an image. If given function f, coordinate of point x_l in the left view, and a depth value d, the coordinate of the corresponding point in the right image can be computed by

$$I'_p(x_l) = I'(f(\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} f^{-1}(x_l, d))) \quad \forall x_l \in I_p \tag{5.4}$$

where I_p is a square patch in the left image center at pixel p, and I'_p is the projection of I_p in the right image according to depth d. Function $f^{-1}(x_l, d) = dK^{-1}x_l$ converts pixel x_l into the three dimensional scene point according to depth d, where K is the intrinsic matrix of the camera. R and t represent the rotation and the translation between two shooting poses.

The ZNCC score between two patches can be calculated by

$$C(x_l, d) = -\sum_{x_l \in I_p} \frac{(I_p(x_l) - \bar{I}_p)(I'_p(x_l) - \bar{I}'_p)}{\sigma(I_p)\sigma(I'_p)} \tag{5.5}$$

where $I_p(x_l)$ returns the intensity value at pixel x_l in patch I_p, and $I'_p(x_l)$ returns the intensity value at the corresponding pixel of x_l in patch I'_p. $\bar{I}_p$ and $\bar{I}'_p$ denote the mean in patch I_p and patch I'_p. $\sigma(I_p)$ and $\sigma(I'_p)$ denote the standard deviation in patch I_p and patch I'_p.

The depth for pixel x_l would be the one with the smallest ZNCC score value:

$$d_{x_l} = \underset{d \in \mathcal{D}}{arg\,\min} C(x_l, d) \tag{5.6}$$

5.1.3 Direct solutions to depth

A new method for depth computation is proposed in this thesis, which runs faster and achieves higher precision theoretically than the reprojection error method. In the reprojection error method, if higher precision wants to be achieved, more candidates of depths have to be searched, in which the computation complexity is $O(n)$. However, the method presented in this section is essentially performing Singular Value Decomposition of a 3×3 matrix for each feature match, whose computation complexity is O(1).

This method is trying to solve equation 5.3 as

$$K^{-1}\lambda_l p_l = R(K^{-1}\lambda_r p_r) + kt \tag{5.7}$$

where there are three linear equations and three unknowns: λ_l and λ_r, depths of a scene point relative to camera at two poses, and the scale factor k for the translation vector.

SVD Approach

Slightly rearranging equation 5.7 yields

$$\begin{gathered} K^{-1}\lambda_l p_l - R(K^{-1}\lambda_r p_r) - kt = 0 \\ \begin{bmatrix} K^{-1}p_l & -RK^{-1}p_r & -t \end{bmatrix} \begin{bmatrix} \lambda_l \\ \lambda_r \\ k \end{bmatrix} = 0 \end{gathered} \tag{5.8}$$

in the form like $Af = 0$ where $A = \begin{bmatrix} K^{-1}p_l & -RK^{-1}p_r & -t \end{bmatrix}$ and $f = \begin{bmatrix} \lambda_l & \lambda_r & k \end{bmatrix}^T$. Therefore, the solution vector f can be computed up to a scale, which doesn't degenerate the shape from motion algorithm at all as it only reconstructs the shape of the scene. So this depth estimation method can be defined as

$$\underset{f}{arg\,\min}_{\|f\|=1} \| Af \| \tag{5.9}$$

where $\| f \|$ is the norm of f. According to Lagrange Multiplier, the solution to this problem is the eigenvector corresponding to the least eigenvalue of $A^T A$, which can be solved by the Singular Value Decomposition (SVD) [3].

Once solutions of depths for all feature points have been obtained, scale all solutions so that the third entry equals to 1, as all feature matches from the same pair of

images share the same translation vector, i.e., sharing the same scale factor k. After scaling, all the first entries in the solution vectors form the depth estimation of feature points in the left view and all the second entries in the solution vectors form the depth estimation of feature points in the right view.

Least Squares Method

An alternative constraint is to set $k = 1$. So equation 5.7 is rearranged into

$$\begin{aligned} K^{-1}\lambda_l p_l - R(K^{-1}\lambda_r p_r) &= kt \\ \begin{bmatrix} K^{-1}p_l & -RK^{-1}p_r \end{bmatrix} \begin{bmatrix} \lambda_l \\ \lambda_r \end{bmatrix} &= t \end{aligned} \tag{5.10}$$

in the form like $Af = t$ where $A = \begin{bmatrix} K^{-1}p_l & -RK^{-1}p_r \end{bmatrix}$ and $f = \begin{bmatrix} \lambda_l & \lambda_r \end{bmatrix}^T$, which is a overdetermined linear system of equations. So the least squares method is employed, the solution vector resulting in the least residual, $\| t - Af \|$, is returned. All the first entries in the solution vectors form the depth estimation of feature points in the left view and all the second entries in the solution vectors form the depth estimation of feature points in the right view. By setting a threshold of the residual, the outliers can be filtered out.

5.2 Point cloud registration

By now, a local point cloud for each pair of images can be computed using the depth calculation methods presented in section 5.1. Each point cloud describes a local part of the scene. To construct a completed shape model of the scene, all the local point clouds have to be stitched together.

The methodology behind the techniques used in this thesis to perform point cloud registration is to utilize the shared feature points between two image pairs, thus sharing world points in two point clouds. By adjusting the scales and orientations of shared reconstructed world points in two local point clouds to fit each other well, the point cloud size ratio, rotation, and translation between them can be computed. Then, one point cloud is scaled and reorientated to fit the other one, by which two consecutive point clouds are fused. Repeating this process whenever a new local point cloud is generated yields a complete shape model which shares the scale and orientation of the last local point cloud.

The techniques used to implement this methodology will be discussed in the following subsections.

5.2.1 Extracting the world points matches

The point cloud size ratio, rotation, and translation between two point clouds can be computed by adjusting and fitting the share world points, which converts the problem to compute the point cloud size ratio, rotation, and translation between two shared world point sets. A scene point would appear in both point clouds if there are feature matches of it in two image pairs, which is a feature tracking problem. The trace of a feature point can be obtained by connecting its matches among the image sequence. If it's failed to find a match of a feature between any two consecutive images, its trace is terminated. Therefore, the shared world points can be obtained if there are continuous matches of their corresponding feature points.

5.2.2 Normalization

The first step to fit the shared point clouds is to normalize them [17], so that they can compared at the same scale. Scaling a point cloud is easier if it has been centered. Thus, centering is presented first in this section and scaling comes later. A point cloud is centered if subtract the mean of the point cloud from all its points. If given a point cloud $X = \{X_1, X_2, \ldots, X_n\}$ where $X_i \in \mathcal{R}^3$, the point cloud X_i can be centered by

$$X_i^c = X_i - \bar{X} \tag{5.11}$$

where $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then, the size of point cloud X is computed by

$$S(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \parallel X_i^c \parallel^2}. \tag{5.12}$$

The point cloud X is normalized by further dividing each wold point by the cloud size

$$X_i^n = \frac{X_i^c}{S(X)} \tag{5.13}$$

5.2.3 Procrustes transformation

In statistics, Procrustes Transformation [17] is a form of statistical shape analysis used to compare the shapes of two or more objects, which is performed by optimally translating, scaling, and rotating the objects so that objects can be superimposed onto each other. Removing the translation and the scale differences between point clouds in this thesis is achieved by normalizing them using the methods described in subsection 5.2.2.

Removing the rotation difference is time consuming, but is still quite straightforward. At first, a shape difference metric has to be defined, by which if two shapes are identical, their shape difference should be zero. The square root of the Sum of Squared Differences (SSD) between corresponding world points can be used to measure the shape difference

$$d = \sqrt{\sum_{i}^{n} \parallel X_i^{n1} - X_i^{n2} \parallel^2} \tag{5.14}$$

where X_i^{n1} and X_i^{n2} represent a match of normalized world points in cloud 1 and in cloud 2. This measure is also often called procrustes distance.

Then, fix one point cloud as the reference and rotate the other one around all three axes X, Y, and Z. The rotation with the least procrustes distance is returned. Searching all the candidates in three dimensions ($2\pi * 2\pi * 2\pi$) is very time consuming and unlikely to meet the realtime criteria. Luckily, the rotation matrix recovered from the essential matrix can be an initial guess, so that only the neighborhood of it in three dimensions need to be searched.

Chapter 6

Evaluation

All key steps involved in performing realtime SfM have been covered in previous chapters. This chapter would be devoted to practical evaluation of the methods. Firstly, the camera calibration result is presented in section 6.1. The performance of feature detectors is analyzed in section 6.2. Then, section 6.3 shows the depth calculating process and how each local point cloud is generated. Afterward, the integration of local point clouds is presented in section 6.4. All code for implementing the algorithm was written in C++ and most of the image processing tasks are based on OpenCV [6].

As the goal of this project is to implement the realtime SfM algorithm on a mobile device and iPhone XR was selected in this project, the algorithm implementation was wrapped by Swift, a programming language for iOS applications. To call C++ functions in Swift, C++ code was encapsulated in Objective-C functions and Swift calls an Objective-C function to use the inside C++ code. The user interface of the iOS software and algorithm's performance, in reality, are presented in section 6.5.

6.1 Camera calibration

As indicated in section 2.3, to calibrate a camera, some number of chessboard images captured by that camera at various poses and distances are needed. In this experiment, the chessboard pattern was printed out and stuck onto a wall. A three-dimensional coordinate system is helpful to describe the image collection process. Set the origin of the coordinate system at the center of the chessboard pattern. X-axis points along the longer edge, Y-axis points along the shorter edge, and Z-axis points into the wall, which forms a right-handed coordinate (refer to figure 6.1). Initially, the camera was set at $(0,0,-k)$, where k is a positive scalar and the optical axis of the camera

points to the origin of the coordinate system. Images were captured by moving the camera forward and backward along the optical axis. Then, the camera was set at $(k,k,-k)$, $(k,0,-k)$, $(k,-k,-k)$, $(0,-k,-k)$, $(-k,-k,-k)$, $(-k,0,-k)$, $(-k,k,-k)$ and $(0,k,-k)$. Images were captured similarly as the camera was set at $(0,0,-k)$. 76 chessboard images with a resolution of 640×480 were used for calibration in this experiment.

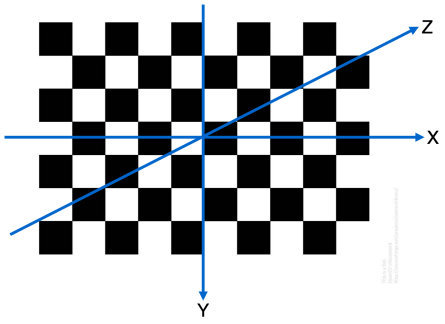


Figure 6.1: Coordinate system for chessboard images collection

To evaluate the accuracy of camera calibration, two measures were used in this experiment – Reprojection Errors and Estimation Errors which are defined in MathWorks [59] as "A reprojection error is the distance between a pattern keypoint detected in a calibration image, and a corresponding world point projected into the same image" and "Estimation errors represent the uncertainty of each estimated parameter". High quality of calibration should yield overall low reprojection errors. The uncertainty of parameter estimation is measured by its standard deviation.

The camera matrix found by OpenCV is

$$K = \begin{bmatrix} f_x & 0 & u_o \\ 0 & f_y & v_o \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 491.092 & 0 & 237.322 \\ 0 & 493.131 & 313.429 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

with standard deviations for

$$\begin{bmatrix} std_{f_x} & std_{f_y} & std_{u_o} & std_{v_o} \end{bmatrix} = \begin{bmatrix} 3.150 & 3.142 & 1.118 & 1.069 \end{bmatrix}. \tag{6.2}$$

One may notice that the computed centre $(313.429, 237.322)$ in the camera matrix K doesn't match the physical centre $(320, 240)$. For a lens system, there are two possible reasons causing this mismatch: decentration and misalignment [62]. For a simple lens,

there are actually two axes of symmetry, optical axis and mechanical axis. The optical axis is defined as a straight line joining the curvature centers of the two surfaces of a lens, while the mechanical axis of a lens is the centerline used by the machine to grind the lens' edge. Ideally, the optical axis coincides with the mechanical axis. However, in practice they won't. The another cause is the misalignment between lens and the sensor or the misalignment among lenses for a compound lens system. The distortion coefficients found by OpenCV are

$$\begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} = \begin{bmatrix} 0.281 & -1.706 & 0.006 & -0.002 & 2.939 \end{bmatrix} \tag{6.3}$$

with standard deviations for

$$\begin{bmatrix} std_{k_1} & std_{k_2} & std_{p_1} & std_{p_2} & std_{k_3} \end{bmatrix} = \begin{bmatrix} 0.013 & 0.150 & 0.001 & 0.001 & 0.479 \end{bmatrix}. \tag{6.4}$$

The extrinsic estimation for each image depicted by Matlab Computer Vision Toolbox [36] is presented in figure 6.2 with standard deviation for each extrinsic estimation depicted in figure 6.3 and figure 6.4.

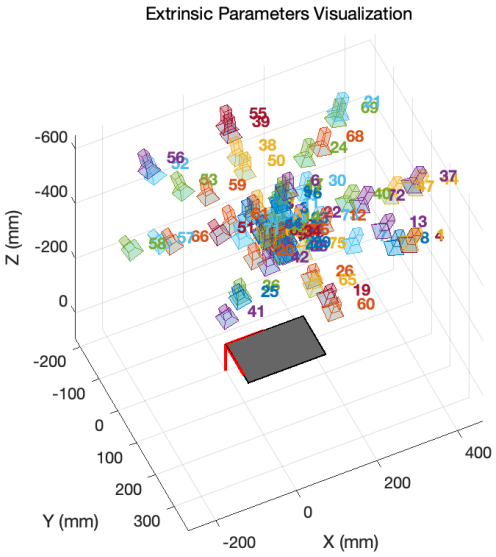


Figure 6.2: Estimated extrinsics

The reprojection error for each chessboard image is presented in figure 6.5.

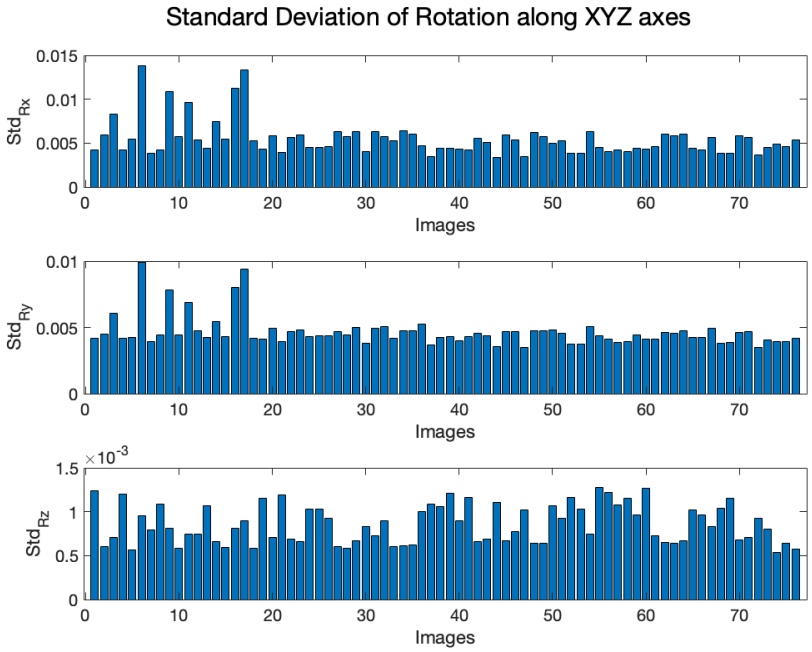


Figure 6.3: Standard deviation of rotation parameters

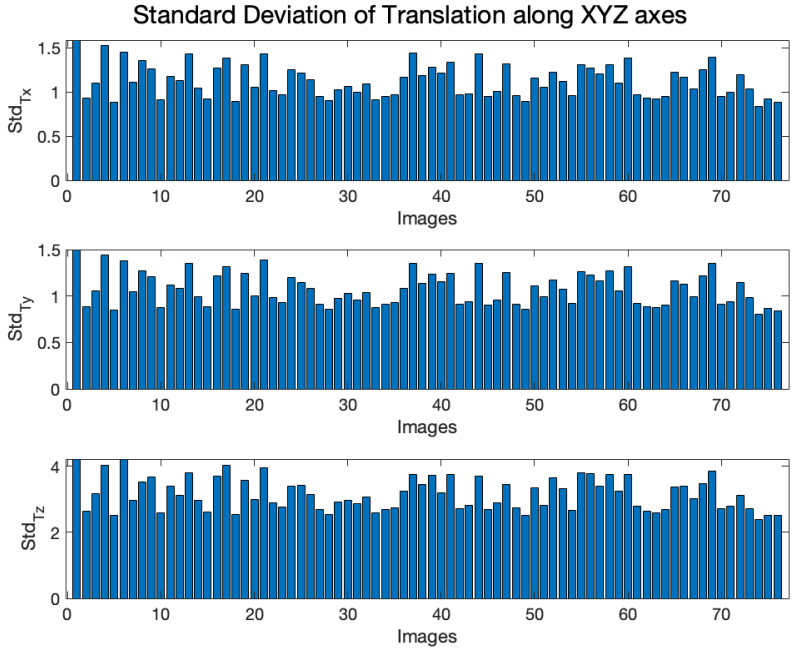


Figure 6.4: Standard deviation of translation parameters

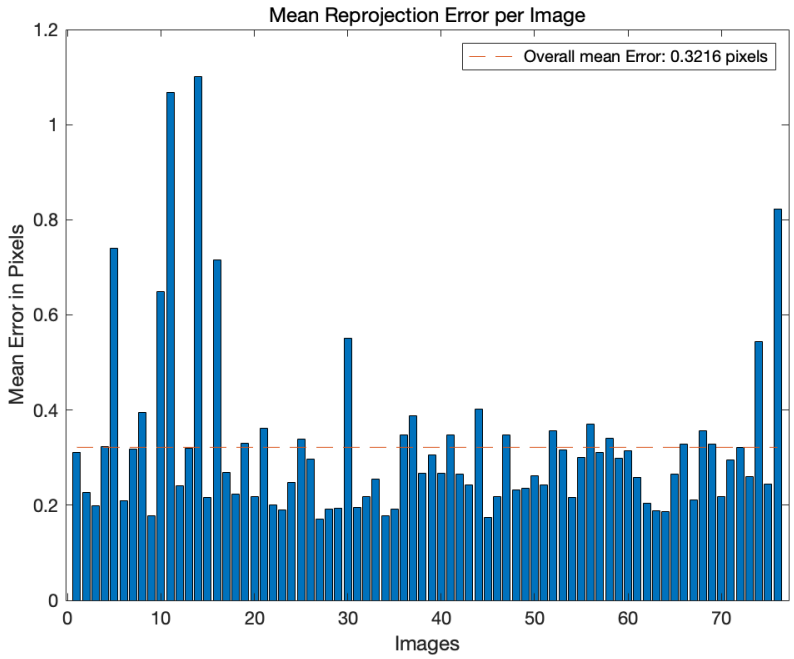


Figure 6.5: Mean reprojection error for each image

6.2 Features detection & matching

Feature detectors SURF and ORB were used in this project. To evaluate their performance and considering the reconstruction purpose of this project, the temple dataset from the *vision.middlebury.edu* [53] was used, an exemplar image of which is showing in figure 6.6.

Figure 6.6: An exemplar image of the temple dataset

The performance of a feature detector can be measured by its repeatability and matching scores between images [42]. The repeatability is computed as the ratio between the number of matches for a given error threshold and the number of features detected in the first image:

$$repeatability = \frac{\#matches\_between\_images}{\#features\_in\_the\_first\_image} \tag{6.5}$$

The matching score is computed as the ratio between the number of correct matches and the number of features detected in the first image:

$$matching\_score = \frac{\#correct\_matches\_between\_images}{\#features\_in\_the\_first\_image} \tag{6.6}$$

However, due to the lack of ground truth of feature matches. In this experiment, a *match* between images is defined if the feature points are nearest neighbors to each other and the distance between them is less than a threshold. A *correct_match* is defined if the ratio between the distance of the closet neighbor and the distance of the second closet neighbor is below a threshold, which is called Lowe's distance ratio test in section 3.3.6. The results of detection by two detectors would be presented in this section. Matches between images would be drawn with lines, which makes it easier to

distinguish false positive matches from true positive matches.

Figure 6.7 shows the feature points detected in images *templeR0013* on the left side and *templeR0014* on the right side by the SURF detector with a hessian threshold of 400. There are 773 feature points in image *templeR0013* and 786 feature points in image *templeR0014*. 521 matches between images *templeR0013* and *templeR0014* are found through nearest neighbor searching with a distance threshold of 0.2 and presented in figure 6.8. Having a close look at the matching result, one can easily find that there are many false positive matches. To filter out those false positive matches, the lowe's distance ratio test with a threshold of 0.6 is introduced and 205 "true" positive matches are presented in figure 6.9. The number of false positive matches decreases as the lowe's threshold decreases. Thus, the repeatability and the matching score by the SURF detector in images *templeR0013* and *templeR0014* is $521/773 \approx 0.674$ and $205/773 \approx 0.265$, respectively.

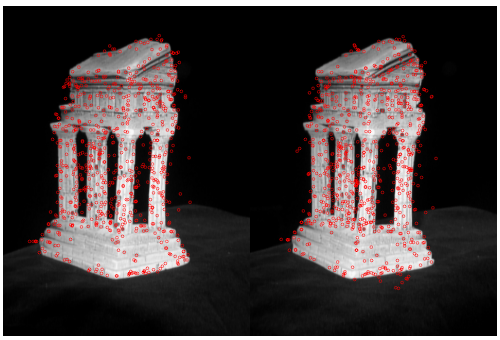


Figure 6.7: The detection result in images *templeR0013* and *templeR0014* by SURF

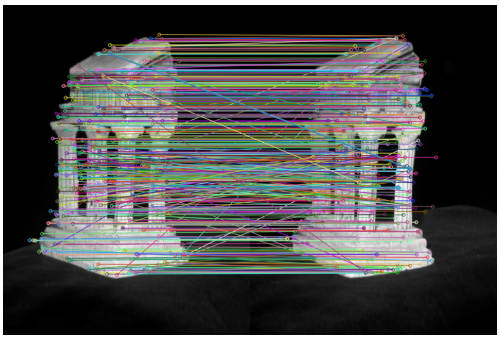


Figure 6.8: The matching result of SURF feature points in images *templeR0013* and *templeR0014* by nearest neighbor searching with a distance threshold of 0.2

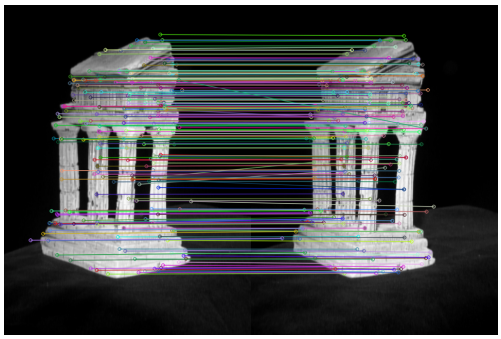


Figure 6.9: The matching result of SURF feature points in images *templeR0013* and *templeR0014* by nearest neighbor searching with a distance threshold of 0.2 and lowe's distance ratio test with a threshold of 0.6

Figure 6.10 shows the top 1000 feature points detected in images *templeR0013* on the left side and *templeR0014* on the right side by the ORB detector. 706 matches between images *templeR0013* and *templeR0014* are found through nearest neighbor searching with a hamming distance threshold of 50 and presented in figure 6.11. To filter out those false positive matches, the lowe's distance ratio test with a threshold of 0.8 is introduced and 323 "true" positive matches are presented in figure 6.12. Thus, the repeatability and the matching score by the ORB detector in images *templeR0013* and *templeR0014* is $706/1000 \approx 0.706$ and $323/1000 \approx 0.323$, respectively.

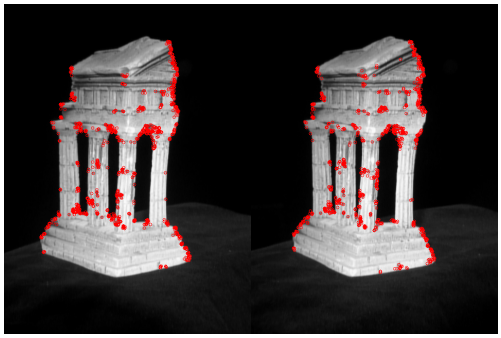


Figure 6.10: The detection result in images *templeR0013* and *templeR0014* by ORB

The reason for changing the lowe's threshold from 0.6 for SURF to 0.8 for ORB is to ensure that the performance of the ORB descriptor is reported objectively by the lowe's ratio distance test. A lowe's threshold of 0.6 for ORB yields 0 false positive matches as showing in figure 6.13 but many true positive matches are discarded compared with figure 6.12. Having a close look at the detection results by SURF and ORB

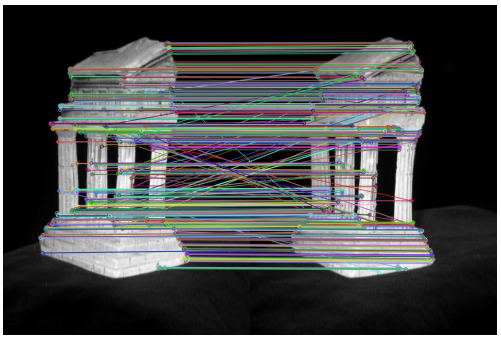


Figure 6.11: The matching result of ORB feature points in images *templeR0013* and *templeR0014* by nearest neighbor searching with a hamming distance threshold of 50

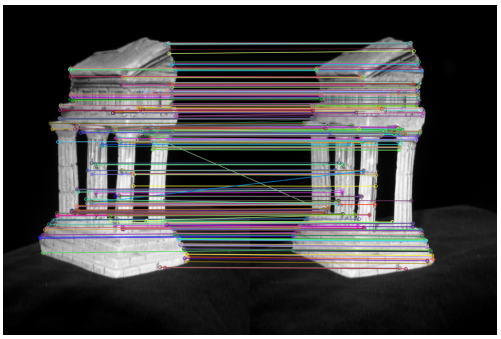


Figure 6.12: The matching result of ORB feature points in images *templeR0013* and *templeR0014* by nearest neighbor searching with a hamming threshold of 50 and lowe's distance ratio test with a threshold of 0.8

in figure 6.7 and figure 6.10, one can notice that the feature points detected by ORB are intended to cluster in groups while the feature points detected by SURF are more uniformly distributed. The distribution of feature points has a noticeable impact on the estimation of the fundamental matrix and then eventually effects the performance of the depth calculation. Based on the experiments, a uniform distribution is preferred.

6.3 Point cloud construction

As indicated in section 4.2, section 5.1.1 and section 5.1.2, the reprojection error method for depth searching is conducted by triangulations generated by the epipolar geometry. To perform depth searching, the fundamental matrix between images has to be computed using the feature matches found in section 6.2. With the given camera

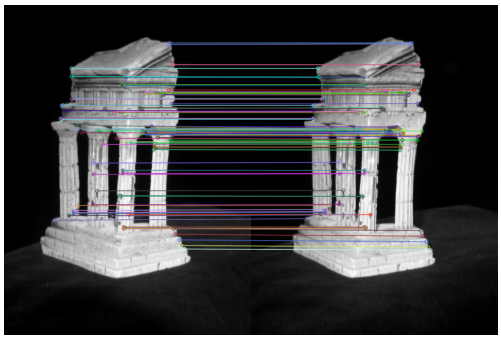


Figure 6.13: The matching result of ORB feature points in images *templeR0013* and *templeR0014* by nearest neighbor searching with a hamming threshold of 50 and lowe's distance ratio test with a threshold of 0.6

matrix in the temple dataset and the computed fundamental matrix, pose change between images can be estimated. By now, the triangulation for each feature match can be generated using the camera matrix and the rotation matrix and the translation vector between two poses by equation 5.4.

In the original reprojection error method for depth searching, the Zero-mean Normalized Cross-Correlation (ZNCC) score between two patches is computed and the patches with the least score value are regarded as a match, which is very time consuming, especially many feature matches have already been extracted for calculating the fundamental matrix. Thus, instead of the ZNCC scores being computed for each match, the euclidian distance between the projection of feature point from the first image onto the second image and the matching point from the second image is calculated. The depth candidate with the minimum distance is returned as the depth information for that feature match. Figure 6.14 shows the depth searching process for a feature match in images *templeR0013* on the left side and *templeR0014* on the right side. The green dots indicate a feature match by SURF between images and the red dot string in image *templeR0014* demonstrates the depth searching process. The straight line formed by those red dots conforms to the property of epipolar geometry: the projection of a point in the left view maps to a line in the right view and vice versa. The reprojection starts with a small depth value and the projection of green dot in image *templeR0013* falls on the left side of the red dot string. As the depth value increases, the projection moves toward the right side along the string. Once the projection achieved the minimum distance with the matching point in *templeR0014*, the depth searching process for this match ends. Figure 6.15 shows the sparse reconstruction result for images *templeR0013* and *templeR0014* presented in Meshlab [10].

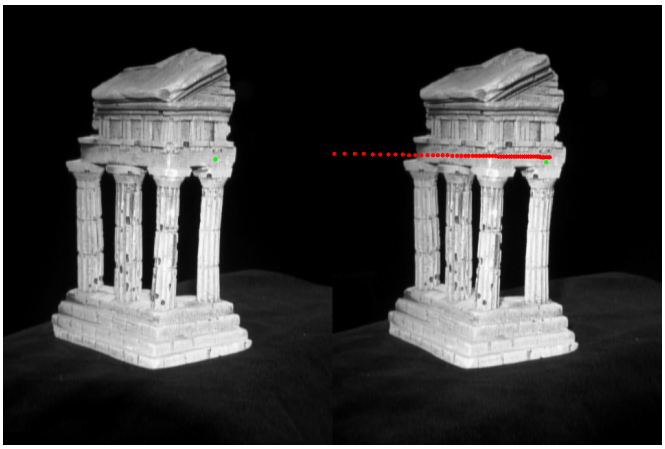


Figure 6.14: Depth searching process by the reprojection error method

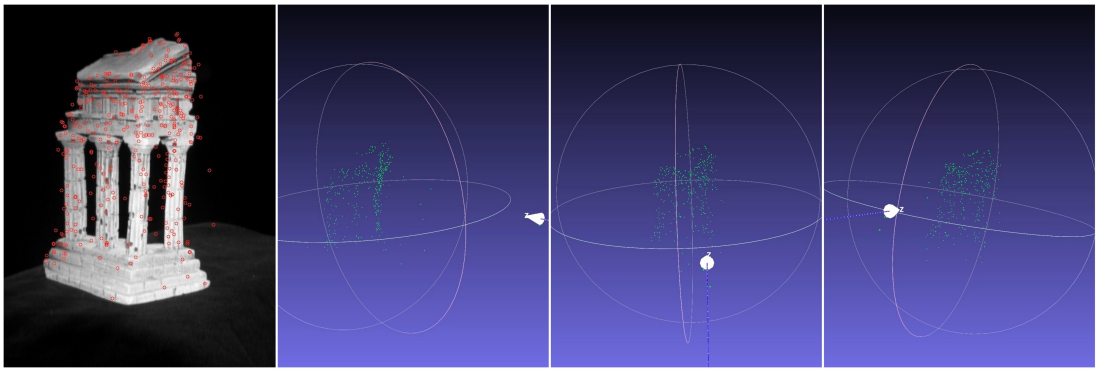


Figure 6.15: Local point cloud generated by the reprojection error method

The sparse reconstruction for images *templeR0013* and *templeR0014* by the least-squares method is presented in figure 6.16 which indicates a comparable result to the reconstruction by the depth searching method. In this experiment, the least-squares method is around twenty times faster than the depth searching method for the temple dataset. However, in the real world, there are more outliers when using the least-squares method than using the depth searching method, as using the depth searching method the depth searching range can be restricted into a specific range. Thus, the world points outside of the searching range, such as world points in the background, won't be included in the shape model using the depth searching method. Based on this observation, the depth searching method was used to implement the iOS reconstruction App which is presented in section 6.5.

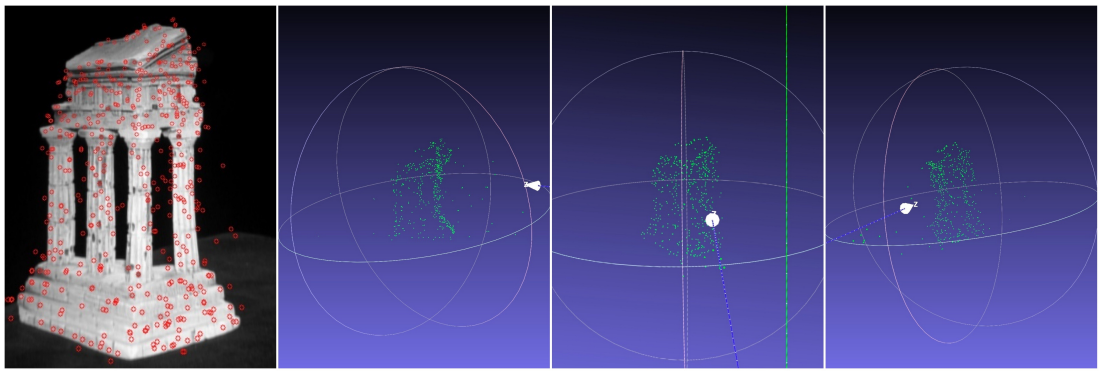


Figure 6.16: Local point cloud generated by the least squares method with a residual threshold of 0.5

6.4 Point cloud registration

The fused shape model from local point clouds in images from *templeR0013* to *templeR0024* by Procrustes Transformation is showing in figure 6.17. One can easily see that even though the resulting shape model reveals the shape of the temple, it's full of outliers which limits the precision of the shape model. The reason for this low quality of reconstruction is that realtime SfM is an online reconstruction instead of the classic offline SfM algorithms that process all images for reconstruction and perform a global optimization like bundle adjustment [57]. Instead, realtime SfM works incrementally. Only a limited number of images are used for the creation and registration of a local point cloud. In this experiment, two images are used in each creation and registration of a local point cloud. Besides, the quality of the resulting shape model also depends on the number and the precision of depth candidates for searching. However, more depth candidates result in a greater computation demand and longer running time, which makes the reconstruction less likely to meet the realtime criteria. It's a trade-off between precision and speed.

However, the performance of realtime SfM algorithm can be better than what the figure 6.17 implies. Apart from the drawbacks of realtime SfM algorithm mentioned in the previous paragraph that restrict the reconstruction quality for the temple dataset, the large baseline (translation) between images also confines the reconstruction quality. A large baseline leads to low repeatability of features because the appearance of a feature varies too much or it may even disappear in the second image so that there aren't enough good feature matches between images for calculating the fundamental matrix, eventually resulting in bad pose estimation and bad depth searching. This conjecture

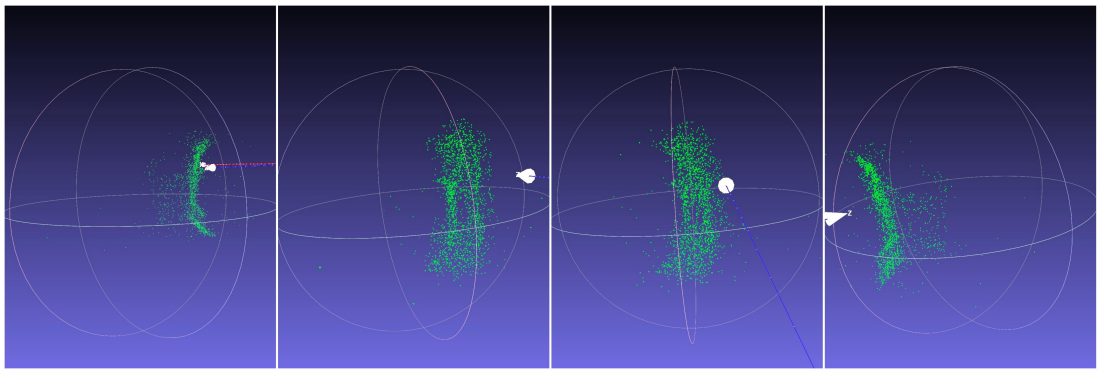


Figure 6.17: Fused shape model from local point clouds by the procrustes transformation

has been proved by performing a reconstruction in the real-world using the developed iOS application. A small baseline can be achieved by moving the iPhone slowly.

6.5 Realtime estimation by the APP

As one of the goals of this project is to develop a reconstruction application running on a mobile phone, the realtime SfM algorithm has been implemented on iOS. The experiment in this section was conducted on iPhone XR released by Apple in 2018.

Figure 6.18 shows the camera calibration interface of the reconstruction App. The algorithm part of this application is implemented based on OpenCV 3.4.10 [6]. Once the user launched the application, there are two buttons with instructions that appear on the screen. As there are three focus modes of the back camera Apple opens to iOS developers: locked, autoFocus and continuousAutoFocus[1], SfM algorithm used in this project needs a fixed focal length and locked mode always gives a blurred image, autoFocus mode is used in this application, in which the focal length will be auto-adjusted at the beginning and then become fixed. So a camera calibration is required for each reconstruction and **Reconstruction** can only be performed after **Camera Calibration**. Tapping the calibration button, the user is expected to input some information about the chessboard pattern and the size of the camera sensor. Into the image collection process, the application would collect chessboard images with a resolution of 640×480 automatically and there is a text field indicating the number of chessboard images being collected and if it's ready for calibration. Clicking the **Calibrate** button and waiting

[1] https://developer.apple.com/documentation/avfoundation/avcapturedevice/focusmode

for some time, the text field would ask the user to go back when the calibration is done and the calibration result is displayed on the homepage.

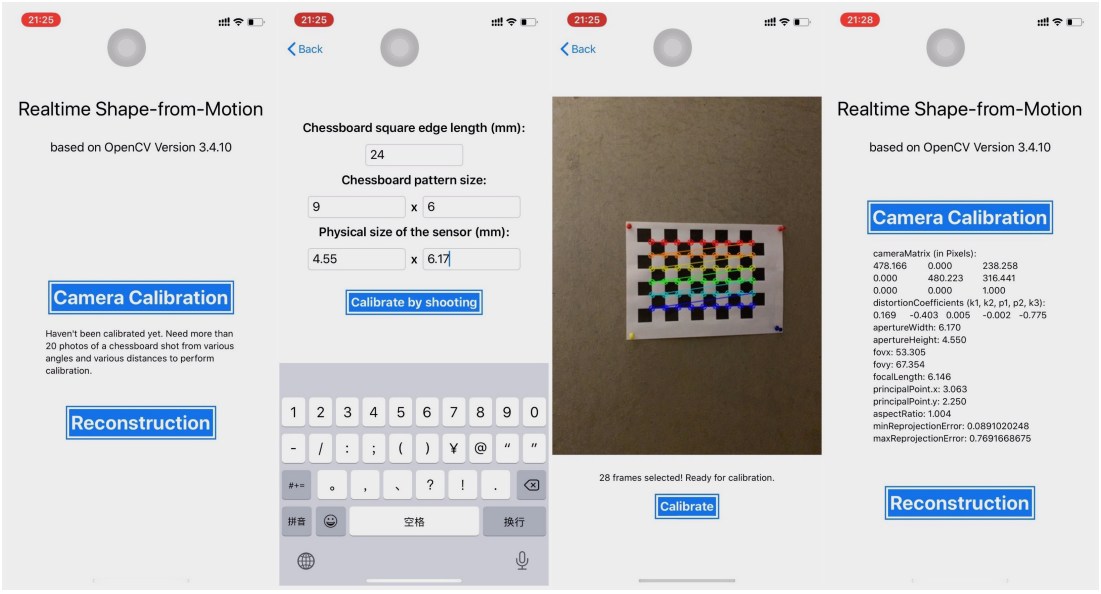


Figure 6.18: The camera calibration interface of the reconstruction App

Figure 6.19 shows the reconstruction interface of the reconstruction application. Users can enter the reconstruction process only after the camera calibration process. There are four small image views on the top indicating the feature points for calculating the fundamental matrix, depth searching and calculating the Procrustes Transformation. In the middle, it's the view for displaying the shape model constructed in realtime. Below the model view, there is a text field showing some log information for each reconstruction like the number of feature points used for each subtask and the amount of elapsed time for this reconstruction. The application would reconstruct the scene automatically while the user moves the phone. Sometimes, there are scenarios that the shape model isn't updated because of a large baseline. The user has to move towards the previous pose a little bit. As SURF can produce a uniform distributed feature points, leading to a good fundamental matrix and uniform reconstructions, SURF is used in this application. And the inliers of features for the fundamental matrix calculation are passed into further reconstruction. Generally, the application needs around 0.8 second for each reconstruction for around 500 feature points, in which approximately 50% of the time is used for the feature detection and the matching process by SURF for calculating the fundamental matrix and the time used for depth searching varies a lot from 20% to 50% depending on the average distance between the scene

and the camera. If the distance is large, it takes more time for depth searching. In the algorithm implementation, depth searching for a feature starts with a small depth value and stops if the reprojection error starts to increase. Reducing the number of feature points and the depth searching range would speed up the reconstruction process.

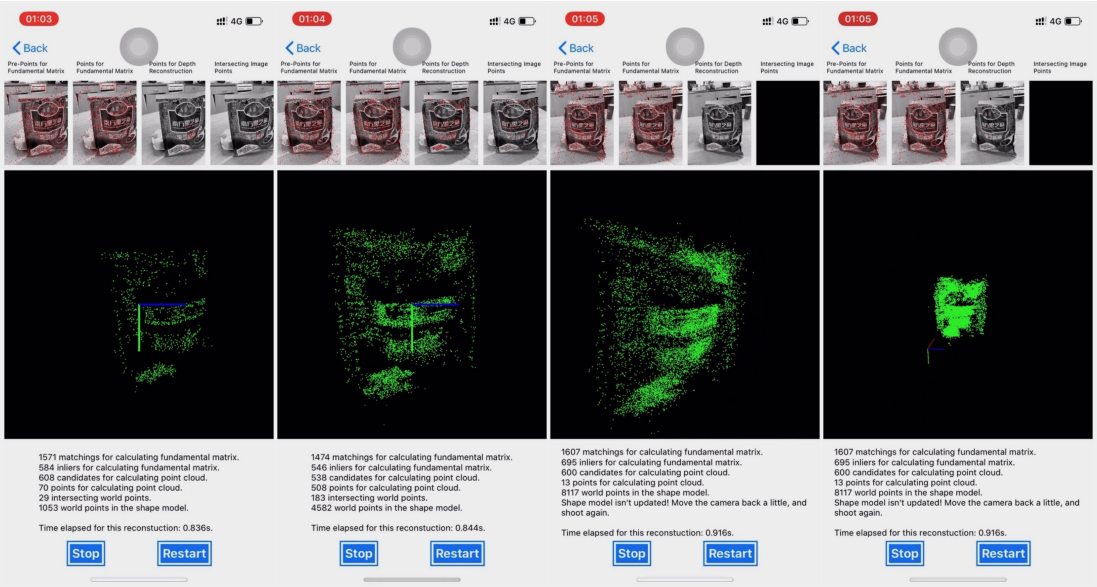


Figure 6.19: The reconstruction interface of the reconstruction App

Video demonstrations for camera calibration and reconstruction by the developed iOS App can be found in `https://youtu.be/oAReG5ddL-4` and `https://youtu.be/yzBv0YeNo88`.

Chapter 7

Conclusion & Outlook: Visual Mono-SLAM

7.1 Conclusion

This thesis gives a detailed introduction of realtime SfM algorithm along with an implementation to test and evaluate its effectiveness. In chapter 2, the camera model, the modeling for optical distortions, and the calibration algorithm computing values of the parameters in a camera system has been introduced. Two features detectors SURF and ORB and some common matching schemes were presented in chapter 3. As SfM computes the depth information of features according to the camera's motion, the pose estimation of the camera at different viewpoints is very crucial for environment reconstruction and has been fully described in chapter 4. Afterward, in chapter 5, the techniques for local point cloud generation and the combination of local point clouds were introduced. The evaluation of the realtime SfM algorithm was conducted in chapter 6 in which an iOS reconstruction App is developed and its functionality has been tested in the real world.

According to the experiment results in chapter 6, the iOS reconstruction App achieves an effective reconstruction at a speed of one reconstruction around per 0.8 second, while the main drawbacks of realtime SfM algorithm are its high computational complexity and the relatively low precision it achieves. Compared with ORB, SURF produces features with a uniform distribution which yields a better estimation of the fundamental matrix indicated by a good reconstruction. Besides, SURF also produces uniform reconstructions. Therefore, SURF is selected as the feature detector used in the iOS App. For each reconstruction by the iOS App, feature detection by SURF and

depth searching are two main factors for time-consuming. Considering to exclude outliers and to achieve higher precision of the reconstruction, the shape model constructed by the App should be followed with a bundle adjustment which gives more computational pressure for the realtime reconstruction purpose. Thus, to achieve a good environment reconstruction in realtime on a low computation power hardware platform, a fast and robust feature detector is needed. SURF is robust enough but is still time-consuming. ORB is as fast as its authors claim but not robust enough for real-world applications. Apart from the repeatability and the matching score as measurements for feature detectors' performance, the uniformity of detected features should also be considered. To exclude outliers, the Extended Kalman Filter provides a solution from the probability perspective, in which the uncertainty for a reconstructed world point is measured. Visual mono-SLAM as an application of SfM with the Extended Kalman Filter is introduced in the following section.

7.2 Outlook: Visual Mono-SLAM

As an application of SfM, visual mono-SLAM obtains new depth information of a static environment by triangulation and updates observed features' depth (updating the map of the environment) and the orientation of the camera (localization of the camera in that map) by a probabilistic method such as the Extended Kalman Filter [11]. Since the realtime SfM can only construct a sparse shape model of the environment, the map created by visual mono-SLAM is a three-dimensional point cloud of the environment. By the Extended Kalman Filter, repeating observations of the same world points and comparing the coordinate of their projections and their expected coordinates refine both the camera's orientation estimation and the shape model (the map) of the environment. The main idea of the Extended Kalman Filter and how it's integrated into Visual Mono-SLAM are explored in the following subsections.

7.2.1 Extended Kalman Filter

The Extended Kalman Filter is an extension of the original linear Kalman Filter to model non-linear systems. Please note that, in this subsection, only a brief introduction of EKF is given. Detailed explanation of EKF can be found in [56].

The Kalman Filter works as a recursive Gaussian filter to estimation the state of continuous linear systems under uncertainty, in which the state vector is modeled by

a multivariate Gaussian distribution with mean μ_t and covariance Σ_t at time t. The mean encodes the position, orientation and velocities of the camera and the estimated positions of all observed features in the world coordinate system. The covariance is a measure of estimated uncertainty of the state. Each time step t is divided into two phases: *prediction* step and *correction* step. The basic idea of the prediction is to estimate how the state x_t evolves from the previous state $x_{t-1} = (\mu_{t-1}, \Sigma_{t-1})$ with the influence of actions a_t, which is done by the prediction function $\bar{\mu}_t = g(a_t, \mu_{t-1})$. Then in the correction step, the prediction $\bar{x}_t = (\bar{\mu}_t, \bar{\Sigma}_t)$ is corrected with the measurement z_t of the system and a measurement function $h(\bar{\mu}_t)$ at time t. The assumption behind this is that the uncertainty for μ_t can be reduced by combining the results from the prediction and the measurement and, thus, μ_t would be a better estimation compared with the prediction or the measurement alone.

Table 7.1 presents the algorithm of the Extended Kalman Filter [2], where R_t and Q_t are the transition noise and the sensor noise at time step t that conform a gaussian distribution and G_t and H_t are Jacobian of the transition function g and measurement function h.

Algorithm: Extended Kalman Filter	
input: prev. mean μ_{t-1}, prev. covariance Σ_{t-1}, actions a_t, measurements z_t	
output: mean μ_t, covariance Σ_t	
1 $\bar{\mu}_t = g(a_t, \mu_{t-1})$	
2 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$	
3 $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$	
4 $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$	
5 $\Sigma_t = (\mathbf{1} - K_t H_t)\bar{\Sigma}_t$	// $\mathbf{1}$ is identity matrix

Table 7.1: The algorithm of the Extended Kalman Filter

7.2.2 Sate representation

For Visual Mono-SLAM, the state vector contains the current orientation, position, and velocities of the camera and the estimated locations of features making up the map. The camera state part will not vary in its dimension as the SLAM system runs, while the features state will vary in dimension since the map is initially empty and will grow over time.

The camera state part is defined in [2] as

$$x_c = \begin{bmatrix} p^w \\ o^w \\ v^w \\ w^c \end{bmatrix} \tag{7.1}$$

with the covariance matrix

$$P = \begin{bmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & V \end{bmatrix}, V = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} \tag{7.2}$$

where p^w is a 3×1 vector and denotes the position of the optical center in the world coordinate system, o^w is a 4×1 unit quaternion vector and denotes the orientation of the camera in the world coordinate system, v^w and w^c are 3×1 vectors and denote the linear velocity of the camera in the world coordinate system and the angular velocity relative to the camera coordinate system.

A feature f_i in inverse depth encoding in the feature state part is defined in [2] as

$$f_i = \begin{bmatrix} x_o & y_o & z_o & \theta_i & \phi_i & \rho_i \end{bmatrix}^T \tag{7.3}$$

where $\begin{bmatrix} x_o & y_o & z_o \end{bmatrix}^T$ is the location of the camera's optical center in the world coordinate system, θ_i and ϕ_i are *azimuth* and *elevation* of feature f_i in the camera coordinate system and ρ_i is the inverse depth of f_i. The coordinate of f_i in the world coordinate system can be obtained by

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i) \tag{7.4}$$

where $m(\theta_i, \phi_i) = \begin{bmatrix} \sin\theta_i \cos\phi_i & -\sin\phi_i & \cos\theta_i \cos\phi_i \end{bmatrix}^T$ is a unit vector pointing from

the optical center to feature f_i. Therefore, the full state vector is defined as

$$x_t = \begin{bmatrix} x_c^T & f_1^T & \dots & f_n^T \end{bmatrix}^T. \tag{7.5}$$

The initialization of the state vector μ_0 and the with covariance matrix Σ_0 can be

$$\mu_0 = \mathbf{0}, \Sigma_0 = \begin{bmatrix} P & \infty \\ \infty & \infty \end{bmatrix} \tag{7.6}$$

7.2.3 Prediction step

Table 7.2 shows the prediction part in the EKF algorithm.

Algorithm: The prediction part in the Extended Kalman Filter
input: prev. mean μ_{t-1}, prev. covariance Σ_{t-1}, actions a_t, measurements z_t
output: mean μ_t, covariance Σ_t
1 $\bar{\mu}_t = g(a_t, \mu_{t-1})$
2 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

Table 7.2: The algorithm of the prediction part in the Extended Kalman Filter

The prediction function g for the camera state part is defined in [11] as

$$g_c(\mu_{t-1}) = \begin{bmatrix} p_{t-1}^w + (v_{t-1}^w + a_t^w \triangle t)\triangle t \\ o_{t-1}^w \times quat((w_{t-1}^c + \alpha_t^c \triangle t)\triangle t) \\ v_{t-1}^w + a_t^w \triangle t \\ w_{t-1}^c + \alpha_t^c \triangle t \end{bmatrix} \tag{7.7}$$

where a_t^w and α_t^c are accelerations for v_{t-1}^w and w_{t-1}^c at time step t. As the world points are supposed to be static, the estimation of their locations should not be changed. Thus, the complete transition function is defined in [2] as

$$g(\mu_{t-1}) = \begin{bmatrix} g_c(\mu_{t-1}) \\ \mathbf{0} \end{bmatrix} \tag{7.8}$$

The detailed derivation of the Jacobian G_t of function g can be found in [2].

7.2.4 Correction step

Table 7.3 shows the correction part in the EKF algorithm.

Algorithm: The correction part in the Extended Kalman Filter	
input: prev. mean μ_{t-1}, prev. covariance Σ_{t-1}, actions a_t, measurements z_t	
output: mean μ_t, covariance Σ_t	
3 $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$	
4 $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$	
5 $\Sigma_t = (\mathbf{1} - K_t H_t)\bar{\Sigma}_t$	// $\mathbf{1}$ is identity matrix

Table 7.3: The algorithm of the correction part in the Extended Kalman Filter

The measurement function h for the feature state part is defined in [2] as

$$h(\bar{\mu}_t) = R^{w \to c}(f_i - O^w) \tag{7.9}$$

where O^w is the representation of the optical center in inverse depth encoding in the world coordinate system and $R^{w \to c}$ is the rotation matrix from the world coordinate system to the camera coordinate system. z_i is the new observation of location of a world point in the camera coordinate system and is obtained by triangulation (refer to subsection 5.1.1).

The detailed derivation of the Jacobian H_t of function h can be found in [2].

Bibliography

[1] Y. Abdel-Aziz, H. Karara, and M. Hauck. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. *Photogrammetric Engineering & Remote Sensing*, 81(2):103–107, 2015.

[2] S. Albrecht. An analysis of visual mono-slam. *Diss. Master's Thesis. Universität Osnabrück*, 2009, 2009.

[3] K. E. Atkinson. *An introduction to numerical analysis*. John wiley & sons, 2008.

[4] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[5] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1000–1006. IEEE, 1997.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] D. C. Brown. Decentering distortion of lenses. *Photogrammetric Engineering and Remote Sensing*, 1966.

[8] M. Brown and D. G. Lowe. Invariant features from interest point groups. In *BMVC*, volume 4, 2002.

[9] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[10] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136. Salerno, 2008.

[11] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[12] P. E. Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, Computer Science Division, Berkeley CA, 1996.

[13] L. Di Stefano, S. Mattoccia, and F. Tombari. Zncc-based template matching using bounded partial correlation. *Pattern recognition letters*, 26(14):2129–2134, 2005.

[14] en.wikipedia.org. Distortion (optics). Accessed: 2020-07-23.

[15] en.wikipedia.org. Epipolar geometry. Accessed: 2020-04-23.

[16] en.wikipedia.org. Pinhole camera model. Accessed: 2020-04-23.

[17] en.wikipedia.org. Procrustes analysis. Accessed: 2020-08-27.

[18] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[19] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[20] L. M. Florack, B. T. H. Romeny, J. J. Koenderink, and M. A. Viergever. General intensity transformations and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2):171–187, 1994.

[21] D. A. Forsyth and J. Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[22] fr.wikipedia.org. Magnification. Accessed: 2020-07-23.

[23] fr.wikipedia.org. Structure from motion. Accessed: 2020-04-23.

[24] C. G. Harris, M. Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[25] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[26] K. Hata and S. Savarese. Cs231a course notes 1: Camera models. Accessed: 2020-07-29.

[27] http://www.vision.caltech.edu. Description of the calibration parameters. Accessed: 2020-04-23.

[28] iMorpheus.ai. Feature map. Accessed: 2020-08-15.

[29] G. Klein and D. Murray. Improving the agility of keyframe-based slam. In *European conference on computer vision*, pages 802–815. Springer, 2008.

[30] J. J. Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.

[31] T. Lindeberg. Scale-space for discrete signals. *IEEE transactions on pattern analysis and machine intelligence*, 12(3):234–254, 1990.

[32] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

[33] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[34] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[35] A. Marrugo. Lecture notes on the graduate course computer vision, 1996. Accessed: 2020-08-04.

[36] MathWorks. Computer vision toolbox. Accessed: 2020-08-23.

[37] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 525–531. IEEE, 2001.

[38] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1):63–86, 2004.

[39] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[40] F. Mindru, T. Tuytelaars, L. Van Gool, and T. Moons. Moment invariants for recognition under changing viewpoint and illumination. *Computer Vision and Image Understanding*, 94(1-3):3–27, 2004.

[41] A. Mordvintsev and A. K. Feature matching. Accessed: 2020-08-15.

[42] T. Mouats, N. Aouf, D. Nam, and S. Vidas. Performance evaluation of feature detectors and descriptors beyond the visible. *Journal of Intelligent & Robotic Systems*, 92(1):33–63, 2018.

[43] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[44] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 850–855. IEEE, 2006.

[45] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[46] O. Ozyesil, V. Voroninski, R. Basri, and A. Singer. A survey of structure from motion. *arXiv preprint arXiv:1701.08493*, 2017.

[47] photographylife.com. What is lens distortion? Accessed: 2020-07-29.

[48] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche. Monofusion: Real-time 3d reconstruction of small scenes with a single web camera. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88. IEEE, 2013.

[49] P. L. Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999.

[50] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[51] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[52] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[53] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 1, pages 519–528. IEEE, 2006.

[54] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[55] G. Terzakis. Relative camera pose recovery and scene reconstruction with the essential matrix in a nutshell. Technical report, Technical report MIDAS. SNMSE. 2013. TR. 007, Marine and Industrial Dynamic . . . , 2013.

[56] S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[57] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[58] R. Y. Tsai and T. S. Huang. Uniqueness and estimation of three-dimensional motion parameters of rigid objects with curved surfaces. *IEEE Transactions on pattern analysis and machine intelligence*, (1):13–27, 1984.

[59] uk.mathworks.com. Evaluating the accuracy of single camera calibration. Accessed: 2020-04-23.

[60] Y.-m. Wei, L. Kang, B. Yang, et al. Applications of structure from motion: a survey. *Journal of Zhejiang University SCIENCE C*, 14(7):486–494, 2013.

[61] T. Werner and T. Pajdla. Cheirality in epipolar geometry. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 548–553. IEEE, 2001.

[62] R. G. Willson and S. A. Shafer. What is the center of the image? *JOSA A*, 11(11):2946–2955, 1994.