An Algorithm to Improve Image Registration for a Piecewise Planar World

Master Thesis

KIT – Karlsruhe Institute of Technology
Fraunhofer IOSB – Fraunhofer Institute of Optronics,
System Technologies and Image Exploitation

Bo Cao

August 12, 2020

Responsible supervisor: M.Sc. Bastian Erdnüß
Examiner: Prof. Dr. Ralf Mikut
PD Dr.-Ing. Markus Reischl

Institute for Automation and Applied Informatics
IAI

Statutory Declaration

I hereby confirm that I have written the accompanying master thesis by myself, without contributions from any sources other than those cited in the text and acknowledgments, I have identified the passages taken over either literally or in content as such and that I have observed the current version of the statues of Karlsruhe Institute of Technology to ensure good scientific practice.

Karlsruhe, August 12, 2020

(Bo Cao)

Abstract

Image registration is an important and fundamental topic in computer version, which aim to find the corresponding pixel or displacement field between two consecutive images within an image sequence. The purpose of this thesis is to derive a new algorithm to improve the image registration of piecewise planar world by estimating the multiple homography of planar scenes in consecutive images or stereo images. In general, the homography matrix can only be used to register images with one large flat plane. But the proposed algorithm expands the single homography matrix to multiple homography matrices, which is linked by constant variables. It realizes the registration of planar sub-patches in the image. In addition, the estimated multiple homography contains the information of Fundamental matrix, which can be used to follow the camera path. Specifically, a mathematical model of the algorithm is deduced through the existing research results and related theoretical basis. Multiple homographies for different plane patches are used to build the mapping model, after initialization, the parameters of model are iteratively refined in the way of Gauss-Newton algorithm until catching the stopping criteria. At the same time, a new dataset is also built in order to get a more comprehensive evaluation of the algorithm in the thesis. Finally, the algorithm is implemented and applied to both public datasets and self-built dataset to do the evaluation. The self-built dataset is uploaded at : `https://github.com/Zauberr/Multi-H-Dataset`.

Acknowledgement

First of all, I would like to thank particularly my supervisor M.Sc. Bastian Erdnüß, who gave me the opportunity to work on a subject in my favorite field of study, computer version, and tirelessly took the time to provide me insightful suggestions. I am also grateful to Prof. Dr. Ralf Mikut for the advice throughout this master thesis. Last but not least, I have to thank my family for their continuous support and great help during my studies.

Notation and Symbol

The main symbols and notations adopted are defined below. All symbols used are defined upon their first appearance. Some symbols are used for more than one quantity and they are defined immediately before or after their appearance. Other symbols unlisted here are defined where they appear.

General Symbols

$\alpha, \ldots, \omega$	Scalar
$a, \ldots, z$	Scalar, variable
$\boldsymbol{a}, \ldots, \boldsymbol{z}$	Point
$\vec{a}, \ldots, \vec{z}$	Vector
$A, \ldots, Z$	Matrix, Function Symbols

General Notations

$\mathrm{vec}(A)$	Vectorization of matrix A formed by stacking the columns of A into a single column vector
A^T	Transpose of matrix A
A^{-1}	Inverse of matrix A
$5.69e+02$	Scientific notation of 5.69×10^{02}
$\propto$	Proportional to

Special Symbols

I_n	Identity matrix of n dimensions
0_n	Zero matrix of n dimensions
$\vec{0}_n$	Zero vector of n dimensions
$\mathbf{J}$	Jacobian matrix
n	Number of plane patches
x_{ij}	homogeneous coordinate of pixel in image
x'_{ij}	Cartesian coordinate of pixel in image
k	iterative times
$\vec{e}$	Epipolar line
H_∞	Homography matrix for the plane at infinity
r_1,r_2	Parameters of radiometric correction
$\boldsymbol{c}$	Location of the camera center in the world coordinate system
R	Rotation matrix of world coordinate system relative to the camera
K	Calibration matrix of camera
$\vec{n}$	Unit outward normal vector of plane

Contents

1 Introduction

1.1 Motivation

The master thesis aims to invent a new algorithm to improve image registration for a piecewise planar world. The algorithm estimate compatible homographies between multiple plane patches in an image pair. The image registration technology is used to do the estimation. The result of algorithm can be stated as the epipolar geometry or a set of multiple homography matrices, which can be also used to derive the Fundamental matrix. For future work, it will be used for following the camera path, camera calibration and 3D reconstruction.

This algorithm is related to image registration. It is well suited for image matching which is a key component in almost all image analysis processes. It is essential for a wide range of application, such as navigation, guidance, automatic surveillance, robot vision, and graphics science.

This thesis was prepared in association with department VID of Fraunhofer IOSB that deals with the automatic evaluation of signals from imaging sensors. A special focus is on the evaluation of video data on a moving platform. Such sensor system is used, for example, in the field of reconnaissance and surveillance as an integrated component in flying and space-based platform. Therefore, the algorithm is designed to deal especially with aerial video images. Corresponding to this, the algorithm is applied to consecutive images, in which there are large planar objects.

Basically, the objects in aerial video are flat earth surface. This is in line with the applicable conditions of homography. Global homography transformation can be used to warp the big flat dominated plane in the consecutive images. But in most case, there are not only one dominated plane in image or planes in image are not flat enough, like terraces and mountains. In order to track them, more than one homography between an image pair are used. They can't be chosen freely, but have to obey some rules. The proposed algorithm is to find these compatible homographies between image pairs and use it to do the image registration of sub-patches and refine the epipolar geometry or fundamental matrix. Further, the fundamental matrix is important for searching and detecting the correspondences in images.

1.2 Related Work

Wojciech Chojnacki, Zygmunt L. Szpak, Michael J.Brooks and Anton van den Hengel have sorted out the structure of multiple homography and present an approach for estimation a set of interdependent homography matrices linked together by latent variables [Cho+15] [Cho+10]. The algorithm proposed in this thesis uses the same structure of multiple homography. The difference is that the proposed algorithm uses image registration technology to estimate the multiple homography. A W Gruen has applied the adaptive least squares correlation to image matching, it allows for simultaneous radiometric corrections and local geometrical image shaping, whereby the system parameters are automatically assessed, corrected, and thus optimized during the least squares iterations [Gru85a]. The proposed algorithm also based on the least square correlation, but the result of estimation are multiple homography matrices between different planar patches in the scene i.e. the proposed algorithm combines the least square iteration and homography transformation to do the image registration. The radiometric corrections introduced in [Gru85a] is also used in the proposed algorithm to improve the accuracy of the result.

1.3 Goal of This Thesis

The goal of this thesis is to derive a new algorithm to improve image registration for a piecewise planar world with estimation of multiple homography. The main goal of the proposed algorithm is to estimate compatible multiple homography matrices of different sub-patches in the image pair. The result of the proposed algorithm can be used to refine the epipolar geometry or fundamental matrix. And it can also be used to register the sub-patches in image pair with acceptable error.

In order to achieve this goal, the thesis contains several consecutive goals. First of all, the derivation of the mathematical basic of the algorithm is completed, and secondly the implementation of the algorithm in a program is provided. Finally, a reasonable estimation of multiple homography matrices between image pair is got through the realization of the proposed algorithm. Moreover, an own test scenario is provided to collect test images for the algorithm in a virtual environment. The estimation performance of the algorithm is evaluated based on some public datasets and the self-built dataset.

1.4 Organization

This thesis is structured as follows: Chapter 2 presents some background about image registration, homography and lest square correlation for the thesis. The main idea of the proposed algorithm is described in Section 3.1. The derivation of the algorithm and implementation for image registration and parameter estimation process are shown in Section 3.2. Here some considerations for the program of algorithm is introduced in Section 3.3. Then the result of algorithm is evaluated in Chapter 4. Finally, we conclude this thesis in Chapter 5.

2 Background

This chapter presents some background information. Some theorem and formulas that I will be used in the algorithm later is also mentioned here. The situation of image registration is first introduced as the application scenario of the algorithm. And the algorithm bases on the multiple homography, it is also a section of this chapter. Finally, the main mathematical theorem related to my algorithm is shown in the end.

2.1 Image Registration

Image registration is the process of transforming different sets of data for the same scene, which may be multiple images taken at different times, from different viewpoints, and/or by different sensors, into one coordinate system. Almost all large systems that evaluate images require image registration, or closely related operations on the images, as an intermediate step, in which the final information is gained from the combination of various data sources like in image fusion, change detection, and multichannel image restoration.

Specific applications include, but are not limited to remote sensing (example Figure 2.1) (Land monitoring, multi-spectral classification, change detection, weather forecast, map updating, create super-resolution image, compiling and analyzing images and data from satellites), computer vision (image mosaicing, motion detection, automatic target localization) and medical imaging (example Figure 2.2) (monitoring tumor growth, treatment verification, comparison of the patient's data with anatomical atlases). In the past few decades, image acquisition devices have developed rapidly, so that the number and diversity of images obtained have increased correspondingly, which has led to the study of image registration. A comprehensive survey of image registration methods has been published in 2003 by Barbara [ZF03]. Typically, the process of the most image registration technology is as follows:

- *Feature detection*: Salient and distinctive objects (closed-boundary regions, edges, contours, line intersections, corners, etc.) are manually or, preferably, automatically detected. For further processing, these features can be represented by their point representatives (centers of gravity, line endings, distinctive points).

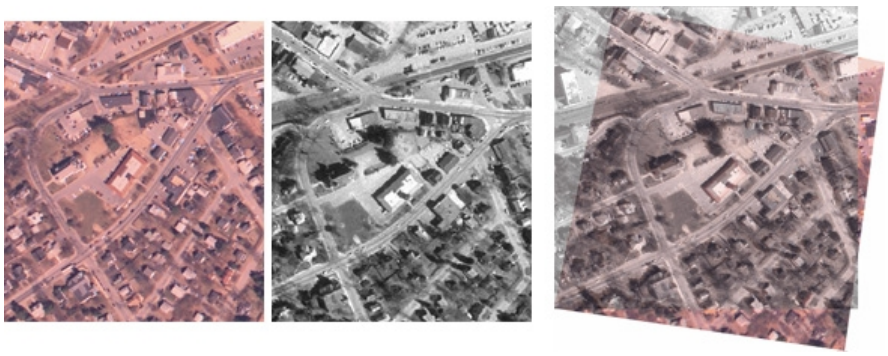


Figure 2.1: Registering aerial photos (© MathWorks)

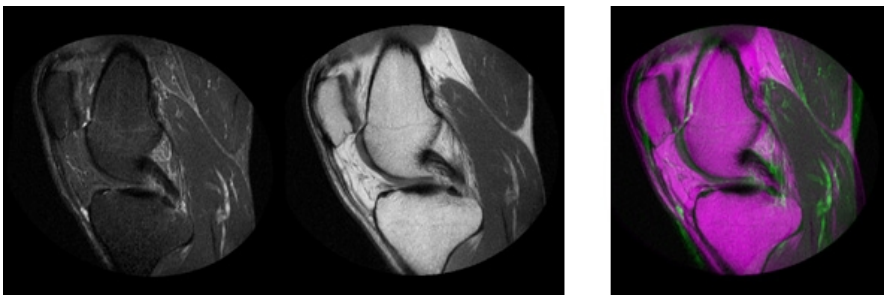


Figure 2.2: Automatic registration on multimodal medical images (© MathWorks)

- *Feature matching*: The connection between corresponding features in image pair is established through this step with various feature descriptors or similarity measures along with spatial relationships.

- *Transform model estimation*: The coordinate transformation so-called mapping function is estimated by matching features. Image registration is performed by the mapping function.

- *Image resampling and transformation*: In this step, one image is transformed by mapping function to another. Image values in non-integer coordinates are computed by the appropriate interpolation technique.

Based on years of development, the current main theories of image registration can be divided into two categories: some typical dense methods seen in Table 2.1 and some typical sparse methods seen in Table 2.2.

Dense Method	
SGM	Semi-Global Matching method performs pixel-wise matching based on Mutual Information and the approximation of a global smoothness constraint. [Hir05]
Correlation based Method	Cross-correlation is the basic statistical approach to registration. It is often used for template matching or pattern recognition in which the location and orientation of a template or pattern are found in a picture. [Gru85a]

Table 2.1: Dense Method

Sparse Method	
SIFT	The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images [Low99]
ORB	Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector, that is based on the FAST key-point detector and a modified version of the visual descriptor BRIEF (Binary Robust Independent Elementary Features). [Rub+11]

Table 2.2: Sparse Method

But sometimes, it's hard to tell them apart. For example, image stitching with SIFT features used the sparse feature points to estimate a dense homography. P. Hellier use simultaneously both dense and landmark-based approaches for nonrigid registration. [HB03]. The Algorithm in the thesis first estimate the multiple homography matrix based on different plane patches in the image, then this series of transformation matrices could be seen as a multiple homography matrices that is used to register the whole image..

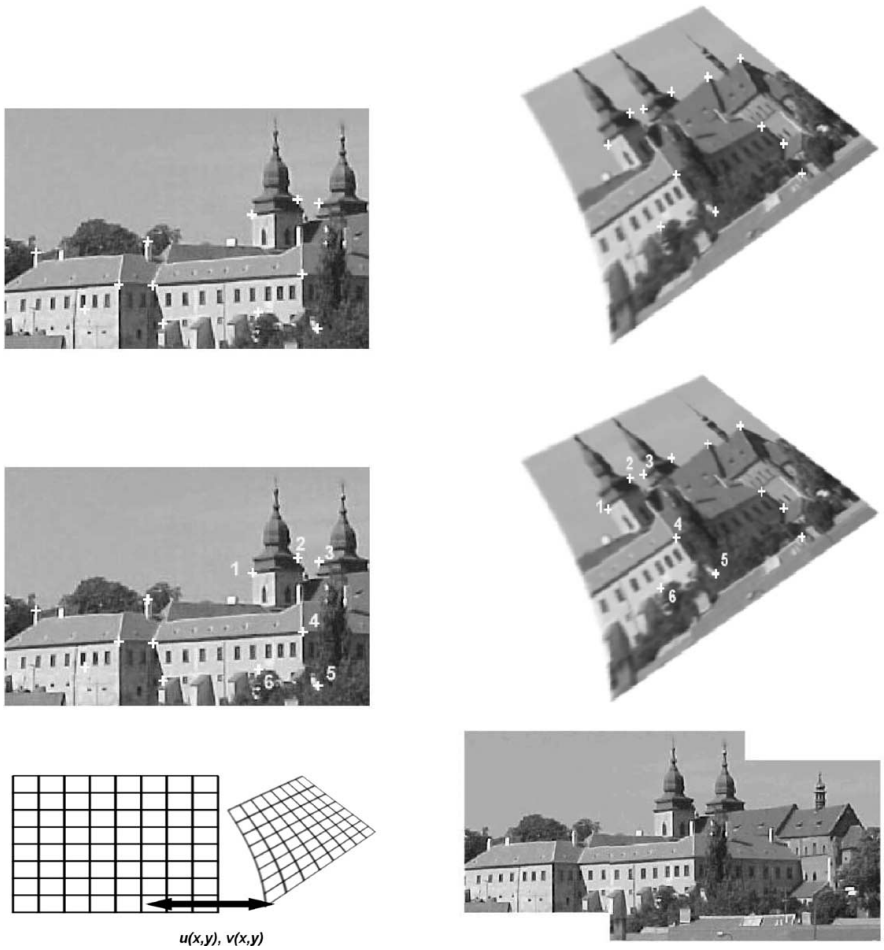


Figure 2.3: Four steps of image registration

Top row – feature detection (corners were used as the features in this case). Middle row – feature matching by invariant descriptors (the corresponding pairs are marked by numbers). Bottom left – transform model estimation exploiting the established correspondence. Bottom right – image resampling and transformation using appropriate interpolation technique. [ZF03]

2.2 Homography

The homography is a projective linear mapping, a special case of the mapping between the images of corresponding points, when those are the image points of 3D planar scene. A 2D point (x, y) in an image can be represented as a 3D vector $\boldsymbol{x} = (x_1, x_2, x_3)$ where $x = \frac{x_1}{x_3}$ and $y = \frac{x_2}{x_3}$. This is called the homogeneous representation of a point and it lies on the projective plane $\mathbb{P}^2$ [Sze]. As mentioned above, a homography is an invertible mapping of points and lines on the projective plane $\mathbb{P}^2$.Other terms for this transformation include *collineation*, *projectivity*, and *planar projective transformation*. Hartley and Zisserman [HZ04] provide the specific definition that a homography is an invertible mapping from $\mathbb{P}^2$ to itself such that three points lie on the same line if and only if their mapped points are also collinear. They also give an algebraic definition by proving the following Theorem 2.1. This tells us that in order to calculate the homography that maps each $\boldsymbol{x}_i$ to its corresponding $\boldsymbol{x}_i^{'}$ it is sufficient to apply a 3×3 matrix H to $\boldsymbol{x}_i$. For more mathematical derivation process, see Section 3.1

Theorem 2.1 (Homography) *A mapping from $\mathbb{P}^2 \rightarrow \mathbb{P}^2$ is a projectivity if and only if there exists a non-singular 3×3 matrix H such that for any point in $\mathbb{P}^2$ represented by vector $\boldsymbol{x}$ it is true that its mapped point equals $H\boldsymbol{x}$.*

Homography have many applications in computer vision and photogrammetry: image registration, image rectification, mosaicing (Figure 2.4) and estimation of camera motion. After extracting the camera motion from the estimated homography matrix, this information can be used for navigation, camera calibration, or to insert a 3D object model into an image or video. The result of the proposed algorithm with the multiple homography could also be used to fundamental matrix estimation and following camera path.

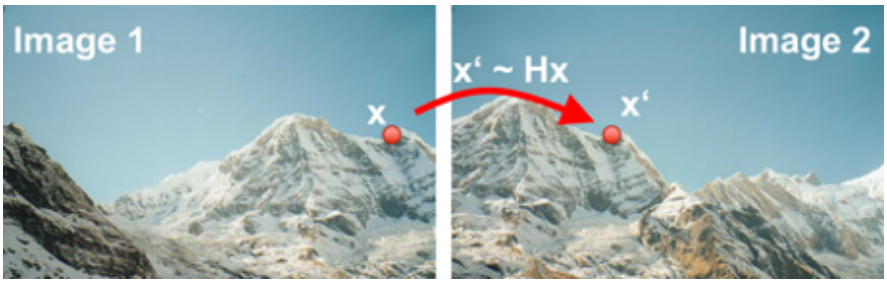


Figure 2.4: Homography application: Panorama [Str20]

But in some situations, like remote sensing, there are more than one flat planes in images (here mostly refers to the surface of earth) or the planes in images are not flat enough to be one plane. According to the definition of homography it can only be used for one flat plane

and more than one homography matrix are needed for this case. And this whole array of homography matrices are all intrinsically interconnected by latent variables, called multiple homography (More mathematical derivation can be found in Subsection 3.1.2) . The algorithm is to find a set of compatible homographies between images pair.

2.3 Epipolar Geometry

The Epipolar geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis (the baseline is the line joining the camera centres). This geometry is usually motivated by considering the search for corresponding points in stereo matching.

Figure 2.5 shows how a pixel in one image $\boldsymbol{x}_0$ projects to an *epipolar line segment* in the other image. The segment is bounded at one end by the projection of the original viewing ray at infinity $\boldsymbol{p}_\infty$ and at the other end by the projection of the original camera center $\boldsymbol{c}_0$ into the second camera, which is known as *epipole* $\boldsymbol{e}_1$. If we project the epipolar line in the second image back into the first, we get another line (segment), this time bounded by the other corresponding *epipole* $\boldsymbol{e}_0$. Extending both line segments to infinity, we get a pair of corresponding *epipolar lines* Figure 2.5, witch are the intersection of the two image planes with the *epipolar plane* that passes through both camera centers $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ as well as the point of interest $\boldsymbol{p}$.

Supposing now that we know only $\boldsymbol{x}_0$, we ask how the corresponding point $\boldsymbol{x}_1$ is constrained. The *Epipolar plane* is determined by the baseline and the object $\boldsymbol{p}$. From above we know that the point $\boldsymbol{x}_1$ must lie on the line of intersection $\vec{l}_1$ of the *epipolar plane* with the second image plane, exactly the *epipolar lines* mentioned above.

In terms of a stereo correspondence algorithm the benefit is that the search for the point corresponding to $\boldsymbol{x}_0$ need not cover the entire image plane but can be restricted to the line $\vec{l}_1$. And my algorithm could also be partly proved with *epipolar geometry* (seen in Subsection 3.2.4).

2.4 Least Squares Correlation

The method of least squares is a standard approach in regression analysis to approximate the solution of over-determined systems (sets of equations which there are more equations than unknowns) by minimizing the sum of the squares of the residuals made in the results of every single equation. What's more, the least squares correlation is a very potent and flexible technique for all kinds of data matching problems. Here its detailed application process to image matching is outlined.

Start from the most basic, linear least squares.

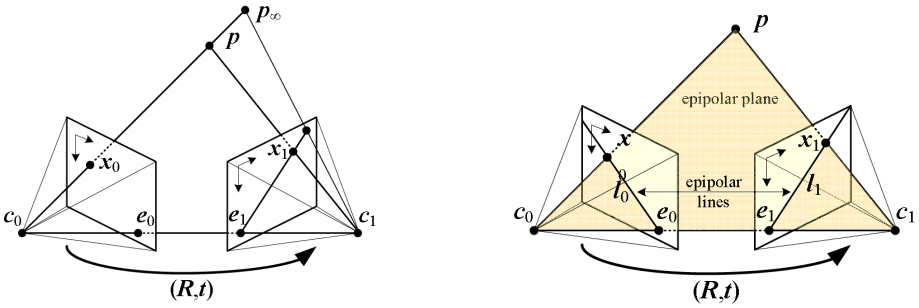


Figure 2.5: Epipolar geometry (a) Epipolar line segment corresponding to one ray; (b) Corresponding set of epipolar lines and their epipolar plane.

Definition 2.2 (Linear Least Squares) *A regression model is a linear one when the model comprises a linear combination of the parameters* $\vec{\beta} = (\beta_1, \beta_2, \cdots, \beta_n)^T$*, i.e.,*

$$f(\vec{x}, \vec{\beta}) = \sum_{j=1}^{m} \beta_j \varphi_j(\vec{x}) \tag{2.1}$$

where the function $\varphi_j(\vec{x})$ *is a function of input* $\vec{x} = (x_1, x_2, \cdots, x_i)^T$*.*

Letting $X_{ij} = \varphi_j(x_i)$ *and putting the independent and dependent variables in matrices* X *and* Y *we can compute the sum of squares in the following way, note that* D *is the set of all data:* X *and* Y*.*

$$E(D, \vec{\beta}) = \left\| X\vec{\beta} - Y \right\|^2 \tag{2.2}$$

Finding the minimum of it can be achieved through setting the gradient of the loss to zero and solving for $\vec{\beta}$

$$\frac{\partial E(D, \vec{\beta})}{\partial \vec{\beta}} = \frac{\partial ((X\vec{\beta} - Y)^T (X\vec{\beta} - Y))}{\partial \vec{\beta}} = -2X^T Y + 2X^T X\vec{\beta} \tag{2.3}$$

Finally setting the gradient of the loss to zero and solving for $\vec{\beta}$ *we get:*

$$\begin{aligned} -2X^T Y + 2X^T X\vec{\beta} = 0 \Rightarrow \\ \hat{\vec{\beta}} = (X^T X^{-1})^{-1} X^T Y \end{aligned} \tag{2.4}$$

When function $f(\vec{x}, \vec{\beta})$ represent the image mapping model, $\vec{x}$ means correspondingly the all points in the image and $\vec{\beta}$ is the parameters in the model, the process will change to an image registration process for a series of consecutive images from the video of the same scene (e.g. aerial video), since the model of image mapping according to the eq. (2.1) is nonlinear, it is a nonlinear least square problem. This is a over-determined problem which has more observations than the parameters. Most algorithms involve choosing initial values for the parameters. Then, the parameters are refined iteratively i.e. the values are obtained by successive approximation. In the thesis, Gauss-Newton algorithm is used to solve this nonlinear least squares problem.

Definition 2.3 (Gauss-Newton algorithm) *Given m functions $\vec{r} = (r_1, \cdots, r_m)^T$ (often called residuals) of n variable $\vec{\beta} = (\beta_1, \cdots, \beta_n)^T$, with $m \geq n$, the Gauss-Newton algorithm iteratively finds the value of the variables that minimizes the sum of squares [Deu11]*

$$S(\vec{\beta}) = \sum_{i=1}^{m} r_i^2(\vec{\beta}) \tag{2.5}$$

Starting with an initial guess $\vec{\beta}^{(0)}$ for the minimum, the method proceeds by the iterations

$$\vec{\beta}^{(s+1)} = \vec{\beta}^{(s)} - \left(\mathbf{J_r}^T \mathbf{J_r}\right)^{-1} \mathbf{J_r}^T \vec{r}\left(\vec{\beta}^{(s)}\right) \tag{2.6}$$

where the entries of the Jacobian matrix it $\vec{\beta}^s$

$$(\mathbf{J_r})_{ij} = \frac{\partial r_i\left(\vec{\beta}^{(s)}\right)}{\partial \beta_j} \tag{2.7}$$

In data fitting, where the goal is to find the parameters $\vec{\beta}$ such that a given model function $y = f(\vec{x}, \vec{\beta})$ best fits some data points$(\boldsymbol{x}_i, \boldsymbol{y}_i)$, the function r_i are the residuals:

$$r_i(\vec{\beta}) = y_i - f\left(\boldsymbol{x}_i, \vec{\beta}\right) \tag{2.8}$$

Then, the Gauss-Newton method can be expressed in terms of the Jacobian $\mathbf{J}_f$ of the function f with respect to $\vec{\beta}$ as

$$\vec{\beta}^{(s+1)} = \vec{\beta}^{(s)} + \left(\mathbf{J_f}^T \mathbf{J_f}\right)^{-1} \mathbf{J_f}^T \vec{r}\left(\vec{\beta}^{(s)}\right) \tag{2.9}$$

Note that now the Jacobian $\mathbf{J}_f$ is for function f not r, the sign before $\left(\mathbf{J_f}^T \mathbf{J_f}\right)^{-1} \mathbf{J_f}^T \vec{r}\left(\vec{\beta}^{(s)}\right)$ changes.

3 Method

3.1 Main idea

In this chapter, the algorithm, how to use homography transformation to do image registration is introduced. It will start with the camera projection to get 3D projection transformation. Then it is simplified to 2D projection with the constrains of points on a plane i.e. Homography transformation. Finally, the algorithm of multiple homography will be introduced.

3.1.1 Projection of Points on a Plane

Camera projection is a projection progress for a set of points from world coordinate system to image coordinate system, i.e. to calculate a projection matrix between these two coordinate system, which is called camera matrix P. And the exact definitions of different coordinates is in Definition 3.1, Definition 3.2, Definition 3.3 and Definition 3.4.

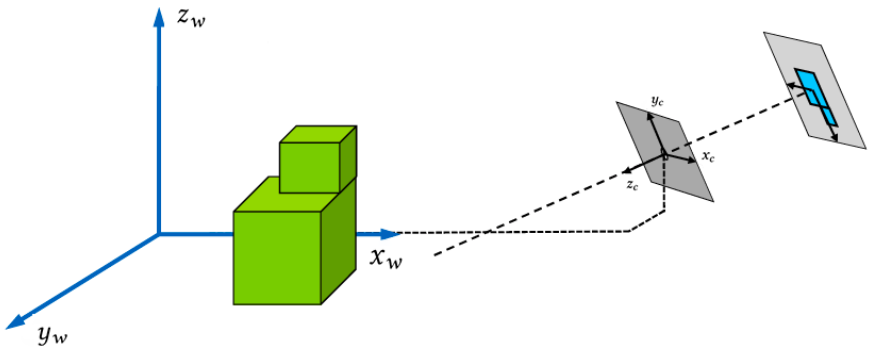


Figure 3.1: World to image mapping [Col]

Just like 3.1 shows, the first step is to transform from world coordinate to camera coordinate. The extrinsic parameter is the 3 by 3 rotation matrix R of camera coordinate system relative to the world coordinate system and the location $\boldsymbol{c}$ of the camera center in the world coordinate

system. The mapping is describe as: (in the equation, $\vec{0}_3$ means a vector of three zeros.)

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{bmatrix} R^T & -R^T\boldsymbol{c} \\ \vec{0}_3^T & 1 \end{bmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \tag{3.1}$$

Then with the 3 by 3 calibration matrix K (consists of intrinsic parameters), the coordinate is transformed to pixel coordinate as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} K & \vec{0}_3 \end{bmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} \tag{3.2}$$

Combine eq. (3.1) and eq. (3.2),

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto z_c \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} K & \vec{0} \end{bmatrix} \cdot \begin{bmatrix} R^T & -R^T\boldsymbol{c} \\ \vec{0}^T & 1 \end{bmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \tag{3.3}$$

And the middle part is the 3 by 4 camera matrix P, which describes the mapping of a pinhole camera from 3D points in the world to 2D points in an image.

$$P = KR^T \begin{bmatrix} I_3 & -\boldsymbol{c} \end{bmatrix} \tag{3.4}$$

Here distortion is ignored to ensure that it is a linear transformation.

When the object more specifically the points of object $\boldsymbol{x}$is on one plane and $\vec{n}$ is defined as the unit outward normal vector of plane, then the scalar product $\boldsymbol{x} \cdot \vec{n}$ is constant. Thus the camera matrix will be simplified to a 3 by 3 matrix, called homography matrix H. Suppose the plane is on the x-y plane in the world, detailed process of this shows in the Figure 3.2. The homography matrix between world and image coordinate is finally simplified to:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{3.5}$$

Go further, when there is a planar object in two different cameras or one camera with different

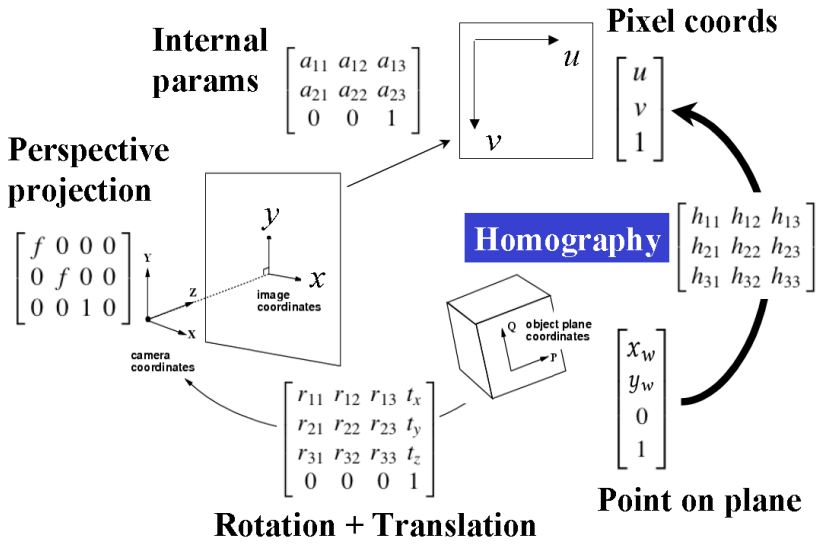


Figure 3.2: Planar projection [Col]

poses, we can get two homography matrix H_1 and H_2 for this two transformation. Then the homography matrix $H_{2\leftarrow 1}$ between this two images of the same object can be calculate as:

$$H_{2\leftarrow 1} = H_2 \cdot H_1^{-1} \tag{3.6}$$

We assume that two fixed uncalibrated cameras give rise to two camera matrices $P_1 = K_1 R_1^T [I_3, -\boldsymbol{c}_1]$ and $P_2 = K_2 R_2^T [I_3, -\boldsymbol{c}_2]$. Moreover, these two cameras photograph the same plane n with a unit outward normal $\vec{n}_n$ situated at a distance d_n from center $\boldsymbol{c}_1$ of camera 1. Then the plane n gives rise to a planar homography from the camera 1 to camera 2: [Cho+15] [BDK]

$$H = K_2 R_2^T (I - \frac{(\boldsymbol{c}_2 - \boldsymbol{c}_1)\vec{n}_n^T}{d_n}) R_1 K_1^{-1} \tag{3.7}$$

where

- $\boldsymbol{c}_1$,$\boldsymbol{c}_2$: location of the camera center in the world coordinate system
- R_1,R_2: rotation matrix of camera coordinate system relative to the world
- K_1,K_2: calibration matrix of camera

- $\vec{n}_n$: unit outward normal vector of plane
- d_n: distance from the plane to camera center $\boldsymbol{c}$ and $d_n = \vec{n}_n\ (\boldsymbol{x} - \boldsymbol{c}_1)$ for any point $\boldsymbol{x}$ on the plane

Definition 3.1 (World coordinate system) *World coordinate system $(x_w, y_w, z_w, 1)^T$ is the right handed cartesian coordinate system where people take the picture . The unit is meter.*

Definition 3.2 (Camera coordinate system) *Camera coordinate system $(x_c, y_c, z_c, 1)^T$ is established on the camera in order to describe the position of object from perspective of camera, as a middle link between the world coordinate system and the image/pixel coordinate system. In the thesis, the origin is set at the center of camera in meters, z axis is pointing from the center of camera to object, y axis is upwards and z axis points to right.*

Definition 3.3 (Image coordinate system) *Image coordinate system$(x, y, 1)^T$ is established on the image in order to describe how locations are measured in the image and the projection transmission relationship of the object from camera coordinate to pixel coordinate during camera projection. In the thesis, the origin is set at the center of image in meters.*

Definition 3.4 (Pixel coordinate system) *Pixel coordinate $(u, v, 1)^T$ defines the pixel location as an array in multi-dimensional space. Each image axis has a length, in pixels, so that the image coordinate run between 1 and the length of axis. The total number of pixels in the image equals the product of the axis length for all image axes. In the thesis, the origin is set at the center of the first upper left pixel in the image.*

And in all definitions above have used the homogeneous coordinate in projective geometry. They have the advantage that the coordinates of points, including points at infinity, can be represented using finite coordinates.

3.1.2 Multiple Homography

Just like in Section 2.2 introduced, when there are more than one flat planes in the image, only one homography matrix can not describe all the planes. For this situation, the multiple homography is needed. Multiplying out eq. (3.7) and rearranging it leads to

$$H = K_2 R_2^T R_1 K_1^{-1} + K_2 R_2^T (\boldsymbol{c}_1 - \boldsymbol{c}_2) \cdot \frac{\vec{n}_n^T}{d_n} R_1 K_1^{-1} \tag{3.8}$$

and define that,

- $H_\infty = K_2 R_2^T R_1 K_1^{-1}$
- $\vec{e} = K_2 R_2^T (\boldsymbol{c}_1 - \boldsymbol{c}_2)$
- $\vec{q}_n^T = \frac{\vec{n}_n^T}{d_n} R_1 K_1^{-1}$

Among them, H_∞ represent the homography matrix for the plane at infinity, $\vec{e}$ is the epipole in the second view, and $\vec{q}_n$ means a 3×1 vector for the plane n, which all the points belongs to. Finally, the homography matrix for plane n is

$$H_n = H_\infty + \vec{e} \cdot \vec{q}_n^T \tag{3.9}$$

If the homography H_∞ at infinity can't be found, a homography matrix H_0 for an exist plane can be used as reference plane and all other planes can be expressed in terms of H_0 with:

$$H_0 = H_\infty + \vec{e} \cdot \vec{q}_0^T \tag{3.10}$$

And for another plane n in the same image, $\vec{p}_n^T$ is defined as $\vec{p}_n^T = \vec{q}_n^T - \vec{q}_0^T$. With eq. (3.10), the new form of homography matrix for plane n is got

$$H_n = H_0 + \vec{e} \cdot \vec{p}_n^T \tag{3.11}$$

In this equation, H_0 is a global and constant value for the whole image, $\vec{e}$ is also a global variable (the epipole in the second view) for the whole image. And $\vec{p}_n^T$ is different for different planes. Therefore, eq. (3.11) is an express of multiple homography with two global variables H_0, $\vec{e}$ and one local variable $\vec{p}_n^T$. People can use it to transform the plane in one image to another or build the corresponding matching for the same plane in different images. The algorithm also use this multiple homography to transform the plane and make image registration per plane.

3.2 Derivation of the algorithm

The derivation of the algorithm of the thesis is shown in this section. First, I apply the theories and methods mentioned before on the image registration process to get my algorithm. After that, the application of my algorithm to a simple case with rectified stereo images is shown. In the end, my algorithm is applied to the unrectified images i.e. arbitrary images.

3.2.1 Image registration Process

Similar images are created in many situations, such as from stereo cameras, map updating and motion detection. Image registration process can be applied on this similar images to match or register the same structure or patches in both images. And the algorithm is a kind of image registration algorithm to register the plane patches in images with the multiple homography approach.

In order to introduce the algorithm, we assume that there is a pair of images with same plane patches, which are from the stereo cameras or consecutive images from the aerial video. Let the two images be I_1 and I_2 where we call I_1 the "template image" and I_2 the "target image". The template image is is formulated as the brightness function $I_1(\boldsymbol{x}'_{ij})$, in which $\boldsymbol{x}'_{ij}$ means the pixel coordinate of the pixel at row i and column j. And the target image is defined as $I_2(\boldsymbol{y}'_{ij})$, in which $\boldsymbol{y}'_{ij}$ means the pixel coordinate of the point in target image, which is corresponding to the point $\boldsymbol{x}'_{ij}$ in template image. The symbol I_1 and I_2 here represent respectively the function that can transform the coordinate variable to pixel value, i.e. I transform the variable in space $\mathbb{R}^2$ to space $\mathbb{R}$(In the thesis, only gray-scaled images is consider for simplicity. But the generalization to color images is not complicated, applying the algorithm on three channels of the image separately and add them up finally). Further more, we will use the homography transformation per patch to do the registration separately, so function T_n represents the process of homography transformation from space $\mathbb{R}^3$ to space $\mathbb{R}^3$ with homogeneous coordinate for plane patch n. Here we define T_n for plane patch n from template to target image.

But a problem comes now, the output of T_n is a 3-length vector in $\mathbb{R}^3$ space, the input of I_2 is paradoxically a 2-length vector in $\mathbb{R}^2$ space. So we still need a dehomogenization function to connect them to describe the transformation process. Therefore, we introduce the function N to do dehomogenization transformation:

$$\boldsymbol{x}'_{ij} = N(\boldsymbol{x}_{ij}) \tag{3.12}$$

$$\boldsymbol{y}'_{ij} = N(\boldsymbol{y}_{ij}) \tag{3.13}$$

where $\boldsymbol{x}_{ij}(\boldsymbol{y}_{ij})$ and $\boldsymbol{x}_{ij}{}'(\boldsymbol{y}_{ij}{}')$ are homogeneous coordinate and Cartesian coordinate of pixel in template image(target image).

From the eq. (3.9), the corresponding pixels in the plane region n of template and target image can be connected as:

$$\begin{aligned} \boldsymbol{y}_{ij} &= H_n \cdot \boldsymbol{x}_{ij} \\ &= (H_\infty + \boldsymbol{e} \cdot \vec{q}_n^T)\boldsymbol{x}_{ij} \end{aligned} \tag{3.14}$$

where $\boldsymbol{x}_{ij} = (\boldsymbol{x}'_{ij}, 1)^T$ and $\boldsymbol{x}_{ij}$ means the i row j column in every patch. H_∞ and $\vec{q}_n^T$ are used in the calculation process. If they can't be found in real application, they could be replaced by arbitrary existing H_0 and $\vec{p}_n^T$ as eq. (3.11) shown. Then function T_n can be expressed as

$$T_n(H_\infty, \boldsymbol{e}, \vec{q}_n, \boldsymbol{x}_{ij}) = (H_\infty + \boldsymbol{e} \cdot \vec{q}_n^T)\boldsymbol{x}_{ij} \tag{3.15}$$

So in this way, a projection of planar region from target to template image has been established.

$$I_2(N(T_n(H_\infty, \boldsymbol{e}, \vec{q}_n, \boldsymbol{x}_{ij}))) \tag{3.16}$$

After getting the mapping model, we have to get an estimation of the variable in it to ensure the accuracy of mapping. According to the Section 2.4, the method Least Squares Correlation is used to solve this problem. Because the function I is nonlinear, Gauss-Newton algorithm is utilized.

As stated in Definition 2.3, most algorithms involve choosing initial values for the parameters. Then, the parameters are refined iteratively. In our application scenario, the parameter is defined as

$$\vec{\beta} = \begin{pmatrix} \mathrm{vec}(H_\infty) \\ \boldsymbol{e} \\ \vec{q}_n \end{pmatrix} \tag{3.17}$$

And the dimension of $\vec{\beta}$ is s. Then the values are obtained by successive approximation:

$$\vec{\beta}^{(k+1)} = \vec{\beta}^{(k)} + \Delta\vec{\beta} \tag{3.18}$$

where a superscript k is an iteration number, and the vector of increments $\Delta\vec{\beta}$ is called the shift vector. For the convenience of expression, we name function $I_2(N(T_n(\vec{\beta}, \boldsymbol{x}_{ij})))$ as $F_n(\vec{\beta}, \boldsymbol{x}_{ij})$. At each iteration the transformation model $F_n(\vec{\beta}, \boldsymbol{x}_{ij})$ is linearized by approximation to a first-order Taylor series expansion around $\vec{\beta}^{(k)}$:

$$\begin{aligned} F_n(\vec{\beta}^{(k+1)}, \boldsymbol{x}_{ij}) &\approx F_n^(k)(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) + \sum_{u=1}^{s} \frac{\partial F_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u} \left({\beta_u}^{(k+1)} - {\beta_u}^{(k)} \right) \\ &\approx F_n^(k)(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) + \sum_{u=1}^{s} j_{nu} \Delta\beta_u \end{aligned} \tag{3.19}$$

where

$$j_{nu} = \frac{\partial F_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u} \tag{3.20}$$

as the coefficient of the Jacobian.

We define mapping error e_{nij} of pixel $\boldsymbol{x}_{ij}$ of template image in patch n as

$$\begin{aligned} e_{nij}^{(k+1)} &= I_1(N(\boldsymbol{x}_{ij})) - F_n^{(k+1)}(\vec{\beta}^{(k+1)}, \boldsymbol{x}_{ij}) \\ &= I_1(N(\boldsymbol{x}_{ij})) - F_n^{(k)}(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) - \sum_{u=1}^{s} j_{nu} \Delta\beta_u \\ &= I_1(N(\boldsymbol{x}_{ij})) - F_n^{(k)}(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) - \mathbf{J}_{F_n} \Delta\vec{\beta}_s \end{aligned} \tag{3.21}$$

where the Jacobian $\mathbf{J}_{F_n}$ denotes a vector with entries j_{nu} for u running from 1 to s. It changes for each iteration. $\Delta\vec{\beta}_s$ consists of the changes of all elements in $\vec{\beta}$

Our aim is to minimize the mapping error. When the gradient for $\vec{\beta}$ of eq. (3.21) is set to zero, e gets the minimum value. When we applied the above equation on all pixels in all plane patches. Then t is the total number of the pixels in all patches and the parameter for $\vec{q}_n$ will change to a big vector $\vec{Q} = (\vec{q}_1^T, \vec{q}_2^T, \cdots, \vec{q}_n^T)^T$, which contains all $\vec{q}_n$ for all patch n. Obviously, the dimension of $\vec{Q}$ is $3n$. $\vec{E}$ is defined as the combination of all $\vec{e}_{nij}$ in vector. The combination of F_n is $F(\vec{\beta}, \boldsymbol{x}_{ij}) = (F_1, F_2, \cdots, F_n)^T$. According to eq. (2.9), the result comes to:

$$\vec{\beta}^{(k+1)} = \vec{\beta}^{(k)} + \left(\mathbf{J}_F^T \mathbf{J}_F\right)^{-1} \mathbf{J}_F^T \vec{E}^{(k)} \tag{3.22}$$

Now, the only left thing is the initialization of the parameters. The algorithm is applied on stereo images or the consecutive images from the video. So there must be EXIF information in the metadata. We have an estimation of the camera positions $\boldsymbol{c}_1, \boldsymbol{c}_2$ from the GPS, and an estimation of the camera rotation R_1, R_2 from the INS, and an estimation of K_1, K_2 from the image size and field-of-view reported by the camera. With this we can get already an estimate of H_∞ and $\boldsymbol{e}$ by eq. (3.8). Moreover, we know approximately where the plane is in the image, this gives us also an estimate of $\vec{q}_n$. (This is the job of preprocessing. What I got is the position of planes, so the plane patches are chosen currently by hand.) And the flow chart of this process is shown in Figure 3.3.

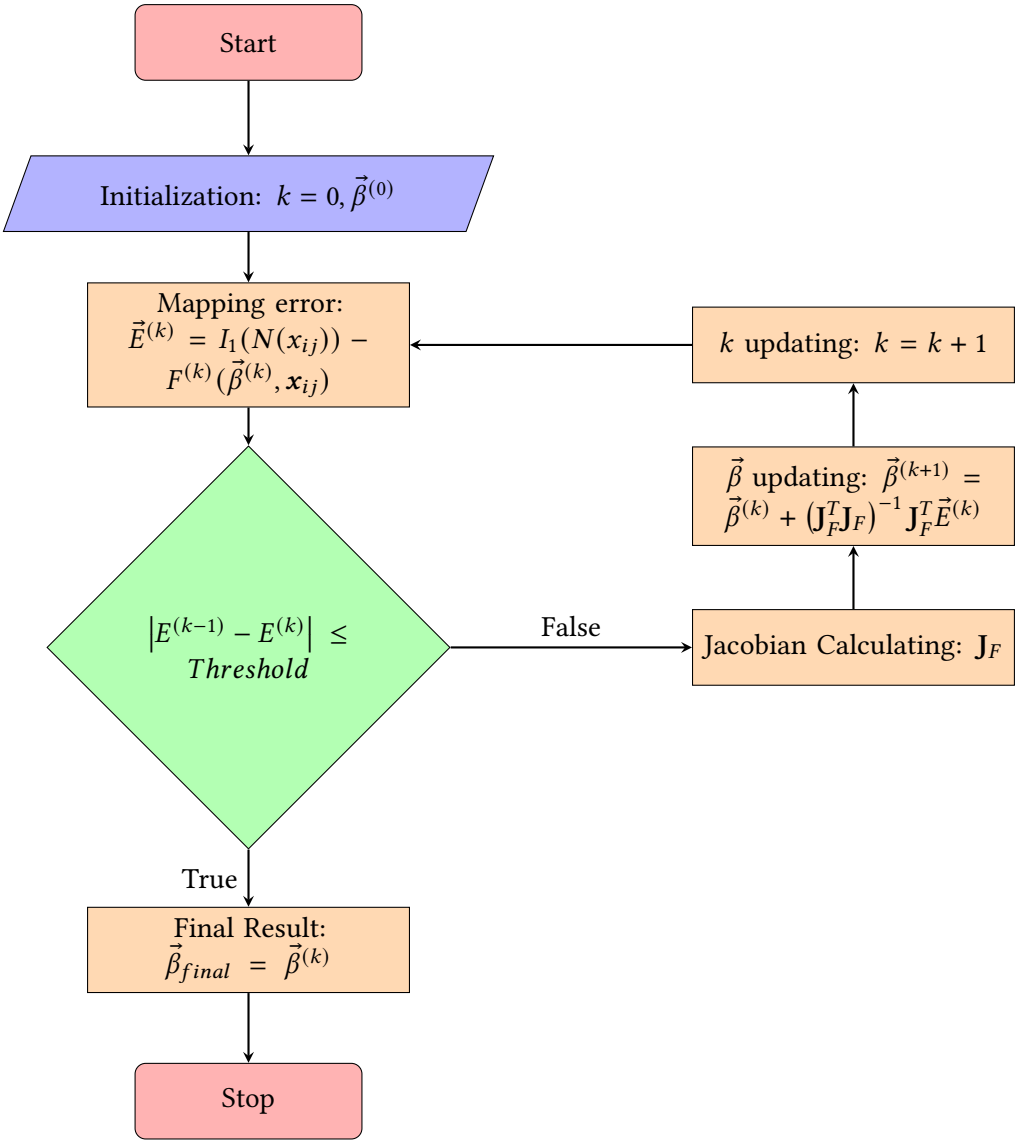


Figure 3.3: Flow chart of Gauss-Newton algorithms

3.2.2 Iterative process

In the iterative process, there is a left peseudoinverse of ${\mathbf{J}_\mathbf{F}}^T$. It's difficult to solve for all the parameters in $\vec{\beta}$, but when the parameter $\vec{\beta}$ is taken apart to optimize, the computational complexity of the total peseudoinverse will be significantly reduced (without proof). On the basis of Subsection 3.1.2, the parameters in $\vec{\beta}$ can be divided into tow parts, global variables H_∞ and $\boldsymbol{e}$ and local variables $\vec{q}_n$.

In this way, $\vec{Q}$ is optimized per patches separately with constant H_∞ and $\boldsymbol{e}$ after the initialization. Then H_∞ and $\boldsymbol{e}$ for all patches can be refined, when we assume $\vec{Q}$ for all patches obtained by the previous step to be fixed . The optimized iterative process is shown in Figure 3.4. And the step "Optimize $\vec{Q}^k$" and "Optimize $H_\infty^k, \boldsymbol{e}^k$" can be done like the way in Figure 3.3.

Next we need to deal with the calculation about function F_n in details. The Jacobian matrix J_F is needed in every iterative step. Put j_{nu} (eq. (3.20)), I_2 (eq. (3.16)), $\vec{\beta}$ (eq. (3.17)) and the definition of F_n together and use the chain rule to calculate the derivative:

$$\begin{aligned} j_{nu} &= \frac{\partial F_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u} \\ &= \frac{\partial I_2(N(T_n(\vec{\beta}, \boldsymbol{x}_{ij})))}{\partial \beta_u} \\ &= \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \frac{\partial T_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u} \end{aligned} \tag{3.23}$$

In order to get j_{nu}, we have to first to calculate $\frac{\mathrm{d}I_2(N)}{\mathrm{d}N}$ and $\frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n}$, then $\frac{\partial T_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u}$. Here d and ∂ both denote the Jacobian, but d means there is derivative of only one vector parameter and ∂ means derivative of one of vector parameters. From the definition of function N (eq. (3.12)), the output of it is a 2-length vector $\boldsymbol{y}_{ij} = (y_i, y_j)^T$, which represents the position of the corresponding pixel in image I_2. So the first term $\frac{\mathrm{d}I_2(\boldsymbol{y}_{ij})}{\mathrm{d}\boldsymbol{y}_{ij}}$ means the gradient of image intensity function. It can be approximated by some differentiation operator. We will come back to this in the Subsection 3.3.3.

We start to solve the second term $\frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n}$. The result of function $T_n(\vec{\beta})$ is a homogeneous

coordinate $\vec{t} = (t_1, t_2, t_3)^T$ of corresponding pixel in image I_2 and $N(\vec{t}) = \vec{t_{12}}/t_3$.

$$\begin{aligned}
\frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} &= \frac{\mathrm{d}N(\vec{t})}{\mathrm{d}\vec{t}} = \frac{\mathrm{d}(t_3^{-1} \cdot \vec{t}_{12})}{\mathrm{d}\vec{t}} \\
&= {t_3}^{-1} \cdot \frac{\mathrm{d}\vec{t}_{12}}{\mathrm{d}\vec{t}} + \vec{t}_{12} \cdot \frac{\mathrm{d}t_3^{-1}}{\mathrm{d}\vec{t}} \\
&= {t_3}^{-1} \cdot \frac{\mathrm{d}\vec{t}_{12}}{\mathrm{d}\vec{t}} - t_3^{-2} \cdot \vec{t}_{12} \cdot \frac{\mathrm{d}t_3}{\mathrm{d}\vec{t}} \\
&= t_3^{-1} \frac{\mathrm{d}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \vec{t}\right)}{\mathrm{d}\vec{t}} - t_3^{-2} \cdot \vec{t}_{12} \cdot \frac{\mathrm{d}\left(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \vec{t}\right)}{\mathrm{d}\vec{t}} \\
&= t_3^{-1} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} - t_3^{-2} \cdot \vec{t}_{12} \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\
&= t_3^{-1} \cdot \begin{bmatrix} I_2 & -\vec{t}_{12}/t_3 \end{bmatrix} \\
&= t_3^{-1} \cdot \begin{bmatrix} I_2 & -N(\vec{t}) \end{bmatrix}
\end{aligned} \tag{3.24}$$

where $\vec{t}_{12}$ means $(t_1, t_2)^T$

Finally, the third part $\frac{\partial T_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u}$. Because $\vec{\beta}$ contains three parts: $\mathrm{vec}(H_\infty)$, $\boldsymbol{e}$ and $\vec{q}_n$ (eq. (3.17)), $\frac{\partial T_n(\vec{\beta}_u)}{\partial \beta}$ is calculated also in three situations.

1. $\beta = \mathrm{vec}(H_\infty)$
2. $\beta = \boldsymbol{e}$
3. $\beta = \vec{q}_n$

When $\beta_u = \mathrm{vec}(H_\infty)$,

$$\begin{aligned}
\frac{\partial T_n(\vec{\beta})}{\partial \,\mathrm{vec}(H_\infty)} &= \frac{\partial((H_\infty + \boldsymbol{e}\vec{q}_n^T) \cdot \boldsymbol{x}_{ij})}{\partial \,\mathrm{vec}(H_\infty)} \\
&= \frac{\partial((H_\infty \cdot x_{ij} + e\vec{q}_n^T \boldsymbol{x}_{ij}))}{\partial \,\mathrm{vec}(H_\infty)} \\
&= \frac{\partial(H_\infty \boldsymbol{x}_{ij})}{\partial \,\mathrm{vec}(H_\infty)}
\end{aligned} \tag{3.25}$$

since $e\vec{q}_n^T \boldsymbol{x}_{ij}$ is independent from $\mathrm{vec}(H_\infty)$.

Here it comes to matrix differentiation. And I have used the kronecker product to solve this differentiation. The definition of Kronecker product is first introduced.

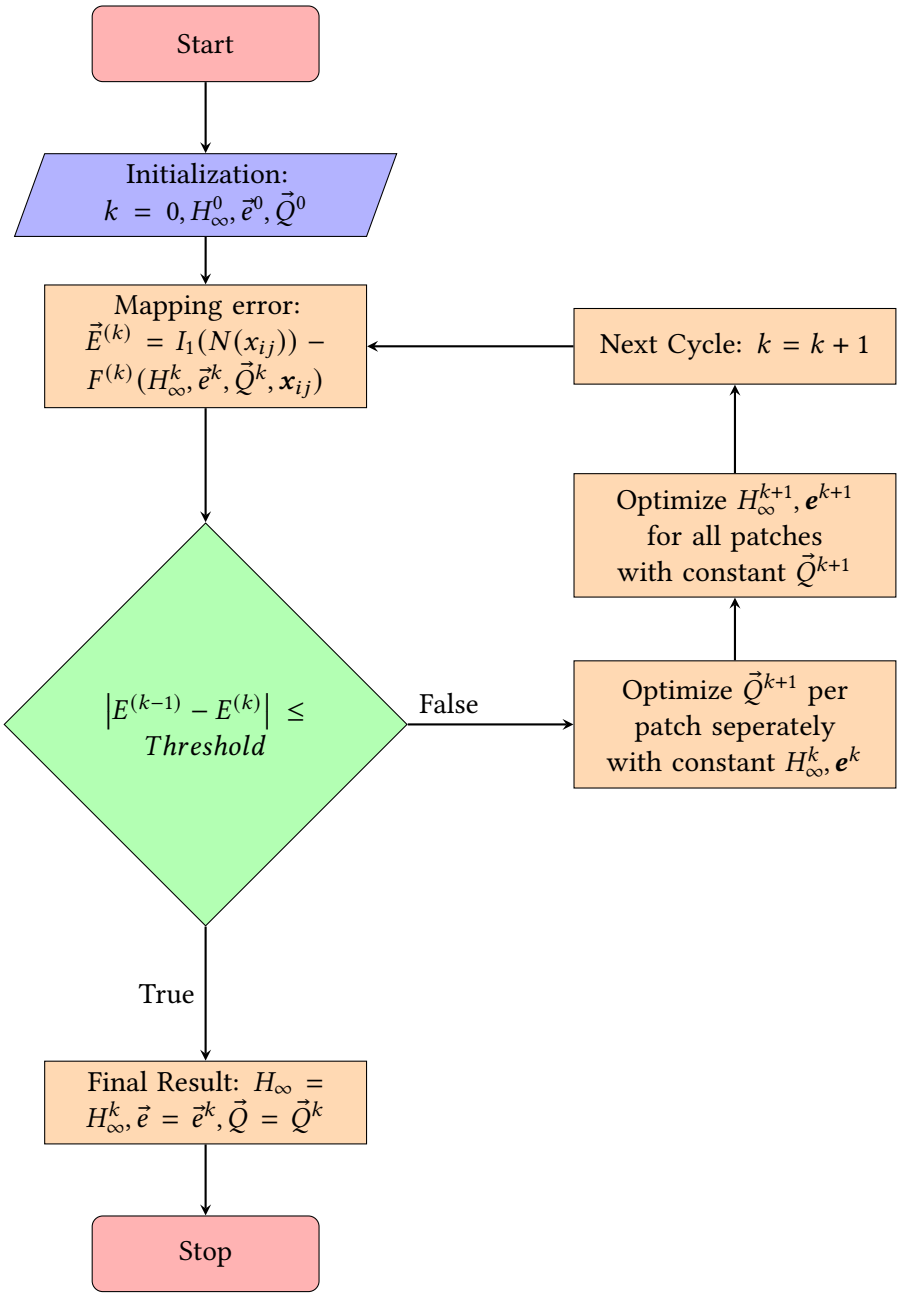


Figure 3.4: Flow chart of optimized iterative process

Definition 3.5 (Kronecker product) *Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$. Then the **kronecker product** (or tensor product) of A and B is defined as the matrix*

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{R}^{mp\times nq} \tag{3.26}$$

Theorem 3.6 (Vectorization of matrix product) *For any three matrices A, B, and C for which the matrix product ABC is defined,*

$$\operatorname{vec}(ABC) = (C^T \otimes A)\operatorname{vec}(B) \tag{3.27}$$

Write eq. (3.25) as

$$\begin{aligned} \frac{\partial T_n(\vec{\beta})}{\partial \operatorname{vec}(H_\infty)} &= \frac{\partial (H_\infty \boldsymbol{x}_{ij})}{\partial H_\infty} \\ &= \frac{\partial \operatorname{vec}(H_\infty \boldsymbol{x}_{ij})}{\partial \operatorname{vec}(H_\infty)} \\ &= \frac{\partial \operatorname{vec}(I_3 H_\infty \boldsymbol{x}_{ij})}{\partial \operatorname{vec}(H_\infty)} \end{aligned} \tag{3.28}$$

Since Theorem 3.6, eq. (3.28) can be rewritten in the form

$$\begin{aligned} \frac{\partial T_n(\vec{\beta})}{\partial \operatorname{vec}(H_\infty)} &= \frac{\partial ((\boldsymbol{x}_{ij}^T \otimes I_3)\operatorname{vec}(H_\infty))}{\partial \operatorname{vec}(H_\infty)} \\ &= \boldsymbol{x}_{ij}^T \otimes I_3 \end{aligned} \tag{3.29}$$

When $\beta = \boldsymbol{e}$,

$$\begin{aligned} \frac{\partial T_n(\vec{\beta})}{\partial \boldsymbol{e}} &= \frac{\partial ((H_\infty + \boldsymbol{e}\vec{q}_n^T)\cdot \boldsymbol{x}_{ij})}{\partial \boldsymbol{e}} \\ &= \frac{\partial (H_\infty \boldsymbol{x}_{ij} + \boldsymbol{e}\vec{q}_n^T \boldsymbol{x}_{ij})}{\partial \boldsymbol{e}} \\ &= \frac{\partial (\boldsymbol{e}\vec{q}_n^T \boldsymbol{x}_{ij})}{\partial \boldsymbol{e}} \\ &= I_3 \vec{q}_n^T \boldsymbol{x}_{ij} \end{aligned} \tag{3.30}$$

since H_∞ and $\boldsymbol{x}_{ij}$ is independent of $\boldsymbol{e}$

When $\beta = \vec{q}_n$,

$$\begin{aligned}\frac{\partial T_n(\vec{\beta})}{\partial \vec{q}_n} &= \frac{\partial((H_\infty + \boldsymbol{e}\vec{q}_n^T) \cdot \boldsymbol{x}_{ij})}{\partial \vec{q}_n} \\ &= \frac{\partial(H_\infty \boldsymbol{x}_{ij} + \boldsymbol{e}\vec{q}_n^T \boldsymbol{x}_{ij})}{\partial \vec{q}_n} \\ &= \frac{\partial(\boldsymbol{e}\vec{q}_n^T \boldsymbol{x}_{ij})}{\partial \vec{q}_n} \\ &= \frac{\partial(\boldsymbol{e}\boldsymbol{x}_{ij}^T \vec{q}_n)}{\partial \vec{q}_n} \\ &= \boldsymbol{e}\boldsymbol{x}_{ij}^T \end{aligned} \tag{3.31}$$

So far, we have completed the entire calculation process and can iteratively solve it.

3.2.3 Radiometric Correction

The algorithm is based on least squares correlation. But it cannot properly respond to many facts that are inseparable from the stereo images of three-dimensional and sometimes even two-dimensional objects. The conjugate images created under the perspective projection rule may be very different from each other.

The terrain gradient, height difference and position and attitude differences of the sensors may cause geometric distortions. Illumination and reflection conditions may distort the image radiometrically. Under certain conditions, this may even trigger geometric displacement. The noise and sampling rate of electronic components may also affect the geometrical and radiometric correspondence of the images.

For aerial images, the energy that sensors on aircraft or satellites record can differ from the actual energy emitted or reflected from a surface on the ground. This is due to the sun's azimuth and elevation and atmospheric conditions that can influence the energy observed by the sensor. There, in order to obtain the real or true ground radiance or reflectance values, radiometric errors must be accounted for. And the algorithm works fast and well if the patches to be matched contain enough signal without too much high-frequency content and if geometrical and radiometric distortions are kept at a minimum. Both conditions are often not encountered in standard aerial images. Therefore radiometric correction is required for the algorithm.

Radiometric correction is a technique to reconstruct physically calibrated values by correcting the spectral distortions caused by sensors , sun angle, topography and the atmosphere as shown in Figure 3.5.

Radiometric correction is classified into two types: absolute and relative correction.:

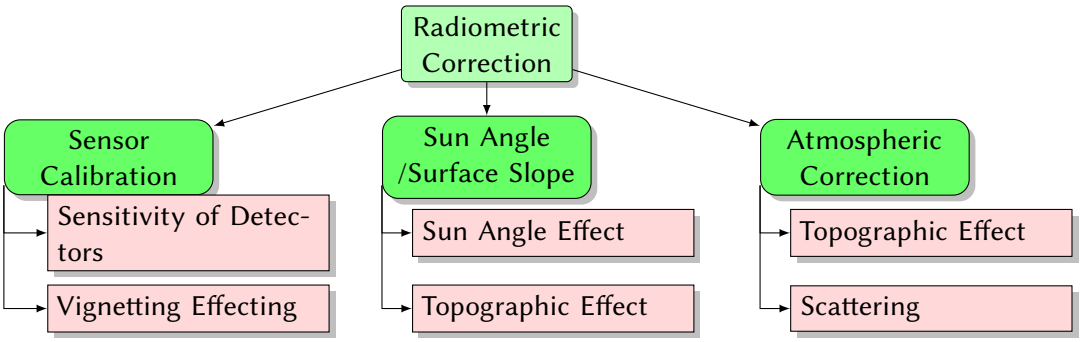


Figure 3.5: Radiometric correction

Absolute correction: Correct radiance or reflectance should be measured or converted by using the sensor calibration data, the sun angle and view angle, atmospheric models and ground truth data. The incident energy input to sensors should be analyzed correctively by radiometric correction. However it can not be applied in most applications, therefore the relative correction is applied because the atmospheric model is so complicated and the exact measurement of atmospheric condition is difficult.

Relative correction: Relative correction is to normalize multi-temporal data taken on different dates to a selected reference data at specific time. The following techniques Table 3.1 will be typical.

My thesis is not mainly about radiometric correction, I only want to reduce the influence of radiometric distortion on my algorithm. So I just choose the least square method to eliminate its influence and make my algorithm more accurate. The specific application process is shown below.

In my case, I_1 or I_2 should be normalized. I choose I_1 here for convenience. Because if I_2 is chosen, then a linear function $R = r_1 I_2(N(T_n)) + r_2$ will replace $I_2(N(T_n))$. It will cause some more calculation.

Relative Radiometric Correction	
Adjustment	Adjustment of average and standard deviation values.
Conversion to normalized index	The normalized difference vegetation index.
Histogram matching	The histograms per band and/or per sensor are calculated and the cumulative histogram with cut-offs at 1% and 99% will be relatively adjusted to the reference histogram.
Least square method	linear function of $y = ax + b$ is determined, where y is reference data and x is data to be normalized.

Table 3.1: Relative Radiometric Correction

After the linear function of $I_1(\boldsymbol{x}_{ij})$ for different plane patches n is defined as $R_n(I_1) = r_{1n}I_1 + r_{2n}$, the mapping error e_{nij} (eq. (3.21) is changed to

$$e_{nij}^{(k+1)} = R_n^{K+1}(I_1(\boldsymbol{x}_{ij})) - I_2(N(T_n^{(k+1)}(H_\infty, \boldsymbol{e}, \vec{q}_n^T, \boldsymbol{x}_{ij})) \tag{3.32}$$

with the definition of the parameter $\vec{r}_n = (a_n, b_n)^T$, above equation will be changed to

$$e_{nij}^{(k+1)} = R_n^k(I_1(\boldsymbol{x}_{ij})) + \frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial r_{1n}}\Delta r_{1n} + \frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial r_{2n}}\Delta r_{2n} - I_2(N(T_n^{(k+1)}(H_\infty, \boldsymbol{e}, \vec{q}_i^T, \boldsymbol{x}_{ij})) \tag{3.33}$$

Put eq. (3.21) and eq. (3.33) together:

$$e_{nij}^{(k+1)} = R_n^k(I_1(\boldsymbol{x}_{ij})) + \frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial r_{1n}}\Delta r_{1n} + \frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial r_{2n}}\Delta r_{2n} - F_n^k(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) - \mathbf{J}_{F_n}\Delta\vec{\beta}_s \tag{3.34}$$

And the vector $\vec{r}_n$ and vector $\vec{q}_n$ are both local parameters and should be optimized per patches separately. So it can be integrated into step "Optimize $\vec{Q}^k$" in Figure 3.4, then the eq. (3.34) changes to

$$e_{nij}^{(k+1)} = R_n^k(I_1) - F_n^k(\vec{\beta}^{(k)}, \boldsymbol{x}_{ij}) - \mathbf{J}_{\mathbf{F_n+R_n}}\Delta\vec{\beta}_o \tag{3.35}$$

with the elements $\mathbf{J}_{\mathbf{F_n+R_n}}$ and $\vec{\beta}_o$. And $\mathbf{J}_{\mathbf{F_n+R_n}}$ is composed of two parts, one is $\mathbf{J}_{F_n}$, the rest two columns are $\frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial a_n}$ and $\frac{\partial R_n^k(I_1(\boldsymbol{x}_{ij}))}{\partial b_n}$. Correspondingly, $\Delta\vec{\beta}_o$ has also two parts, one is $\Delta\vec{\beta}_s$, the rest rows are Δr_1 and Δr_2.

The radiometric correction is finished with the step "Optimize $\vec{Q}^k$" at the same time in this way.

3.2.4 Rectified Stereo Images

For now, let's assume that we have $H_\infty = I_3$ and $\boldsymbol{e} = (1, 0, 0)^T$. This corresponds to the case rectified stereo images. In this case, the iterative process will be simplified to only step "Optimize $\vec{Q}^k$" with constant H_∞ and $\boldsymbol{e}$. This is a special case of the algorithm.

When $H_\infty = I_3$ and $\boldsymbol{e} = (1, 0, 0)^T$, the H_n (eq. (3.9)) is changed to

$$H_n = I_3 + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \vec{q}_n^T \tag{3.36}$$

Rewrite H_n in matrix form:

$$H_n = \begin{bmatrix} 1+q_{n1} & q_{n2} & q_{n3} \\ & 1 & \\ & & 1 \end{bmatrix} \tag{3.37}$$

From this equation, we can get the conclusion, that the corresponding points must lie on the same row as in the other images. Just like Section 2.3 mentioned, the rectified stereo images have epipolar constraint. The corresponding point must lie on the epipolar lines which is the line $\boldsymbol{y}_{ij_2} = x_{ij_2}$ in rectified stereo images. So the search of corresponding point is restricted to this line. And from our algorithm, we get the same conclusion.

The specific mathematics calculation process of rectified images and the result are shown below. For rectified stereo images, the $\vec{\beta}_n$ (eq. (3.17)) is simplified to

$$\vec{\beta}_n = \vec{q}_n \tag{3.38}$$

where we first don't consider radiometric correction and then introduce it afterwards again. Correspondingly,

$$F_n(\vec{\beta}_n, \boldsymbol{x}_{ij}) = F_n(\vec{q}_n, \boldsymbol{x}_{ij}) \tag{3.39}$$

And for patch n, the Jacobian matrix $\mathbf{J}_{F_n}$:

$$\mathbf{J}_{F_n} = \begin{pmatrix} \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{11})}{\partial \vec{q}_n} \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{12})}{\partial \vec{q}_n} \\ \vdots \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{ij})}{\partial \vec{q}_n} \end{pmatrix} \tag{3.40}$$

With eq. (3.31),

$$\mathbf{J}_{F_n} = \begin{pmatrix} \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \boldsymbol{e}\boldsymbol{x}_{11}^T \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \boldsymbol{e}\boldsymbol{x}_{12}^T \\ \vdots \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \boldsymbol{e}\boldsymbol{x}_{ij}^T \end{pmatrix} \tag{3.41}$$

Then $\vec{e}$ can be expressed as

$$\vec{e}_n = \begin{pmatrix} I_1(N(\boldsymbol{x}_{11})) - F_n(\vec{q}_n, \boldsymbol{x}_{11}) \\ I_1(N(\boldsymbol{x}_{12})) - F_n(\vec{q}_n, \boldsymbol{x}_{12}) \\ \vdots \\ I_1(N(\boldsymbol{x}_{ij})) - F_n(\vec{q}_n, \boldsymbol{x}_{ij}) \end{pmatrix} \tag{3.42}$$

So the $\vec{q}_n$ can be iterative refined by

$$\vec{q}_n^{(k+1)} = \vec{q}_n^{(k)} + \left(\mathbf{J}_{F_n}^T \mathbf{J}_{F_n}\right)^{-1} \mathbf{J}_{F_n}^T \vec{e}_n \tag{3.43}$$

until the stopping criteria is reached.

At same time, as mentioned in Subsection 3.2.3, add the radiometric correction. Here the linear function $R_n(I_1) = r_{1n} I_1 + r_{2n}$ will represent the radiometric correction. The $\mathbf{J}_{\mathbf{F_n}}$ (eq. (3.41)) becomes

$$\mathbf{J}_{\mathbf{F_n+R_n}} = \begin{pmatrix} \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{11})}{\partial \vec{q}_n} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{11})))}{\partial r_{1n}} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{11})))}{\partial r_{2n}} \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{12})}{\partial \vec{q}_n} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{12})))}{\partial r_{1n}} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{12})))}{\partial r_{2n}} \\ \vdots & \vdots & \vdots \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{ij})}{\partial \vec{q}_n} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{1n}} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{2n}} \end{pmatrix} \tag{3.44}$$

$\vec{e}_n$ (eq. (3.42)) and $\vec{\beta}_n$ change to

$$\vec{e}_n = \begin{pmatrix} R_n(I_1(N(\boldsymbol{x}_{11}))) - F_n(\vec{q}_n, \boldsymbol{x}_{11}) \\ R_n(I_1(N(\boldsymbol{x}_{12}))) - F_n(\vec{q}_n, \boldsymbol{x}_{12}) \\ \vdots \\ R_n(I_1(N(\boldsymbol{x}_{ij}))) - F_n(\vec{q}_n, \boldsymbol{x}_{ij}) \end{pmatrix} \tag{3.45}$$

and

$$\vec{\beta}_n = \begin{bmatrix} \vec{q}_n \\ r_1 \\ r_2 \end{bmatrix} \tag{3.46}$$

The result updates according to

$$\vec{\beta}_n^{(k+1)} = \vec{\beta}_n^{(k)} + \left(\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}\right)^{-1} \mathbf{J}_{F_n+R_n}^T \vec{e}_n \tag{3.47}$$

The algorithm was implemented in Python and applied it to example rectified stereo images. And one of implementation results is shown here as an example. More evaluation results is listed in Chapter 4. For this example, the template image I_1 and target image I_2 are shown in Figure 3.6. There are quite some planes in the image, for examplethe billboard next to the window on the left wall, the billboard on left balcony and the word on the right wall. The billboard next to the window on the left wall and the word on the right wall are chosen to test the algorithm.

Before running the program, an initialization of parameter $\vec{\beta}_n$ is needed. a and b will start with 1 and 0. The reason is that the radiometric distortion is not too much, a and b could start with no radiometric distortion. Parameter $\vec{q}_n$ is initialized as $(0, 0, q_{n_3})$, where q_{n_3} was estimated by hand. In a future version of the algorithm this will be automated.

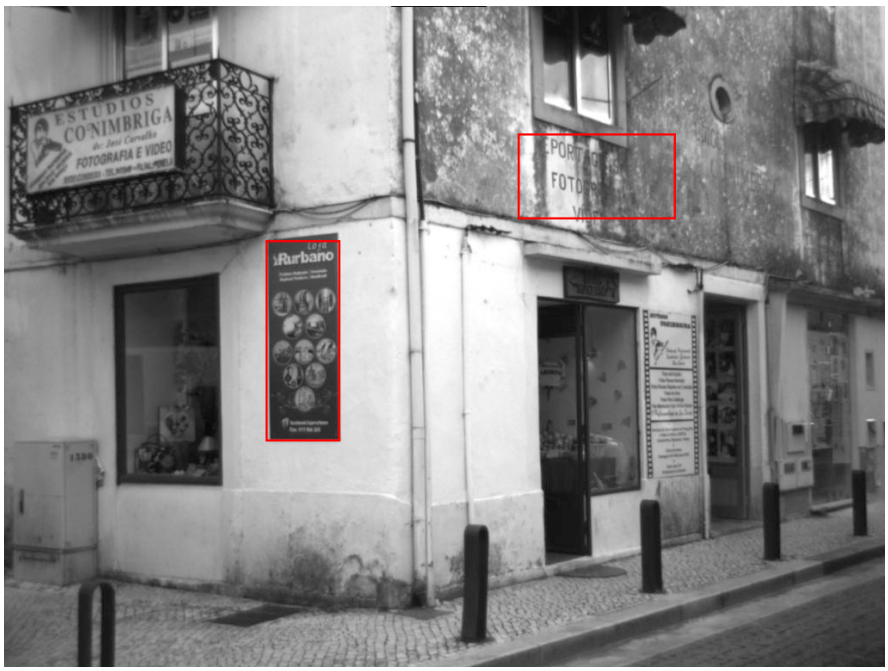
After initializing β_n, it is used to warp plane patch n in the target image to the template image, the result of the first plane patch (billboard on the left wall) is shown in Figure 3.7. Among them, Figure 3.7c shows the difference of this two image patches. It is built by subtract the gray value of the corresponding point. And the difference will be used as gray value of this position to build the difference image. Because the difference of most of pixels is too small, the value is re-scaled by a factor (It's 50 in the thesis) to make it more obvious. What's more, $difference = I_2 - I_1$ and it is shown in blue for positive value and in red for negative value. And the result of the word on right wall is shown in Figure 3.8

In order to quantify the accuracy of result, the definition of PSNR should be introduced here.(Here the formula of PSNR is used but it doesn't have the meaning for noise, so that the result is not an actual PSNR.)

Definition 3.7 (Peak signal-to-noise ratio) *Peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a very wide dynamic range, PSNR is usually expressed in terms of the logarithmic decibel scale [20a]*

PSNR is most easily defined via the mean squared error (MSE). Given a noise-free m×n monochrome image I and its noisy approximation K, MSE is defined as:

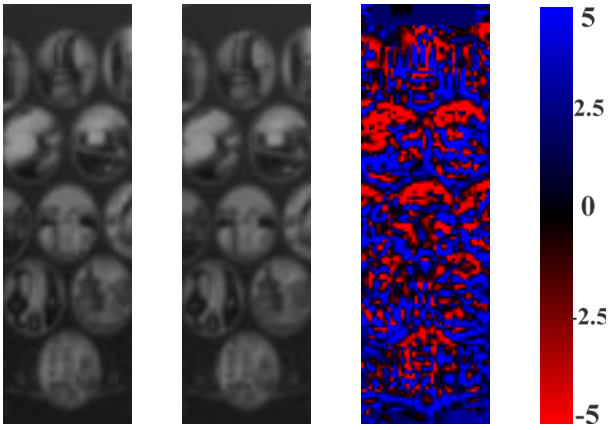
$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{3.48}$$



(a) Template image

(b) Target image

Figure 3.6: Example of a rectified stereo image pair

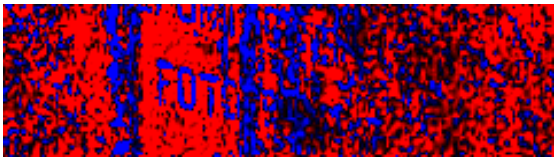


(a) Template image (b) Warped image (c) Difference image (d)

Figure 3.7: Result of plane patch of billboard on the left wall

(a) Word on right wall in template image

(b) Word on right wall in warped image



(c) Difference image of Word on right wall

(d) Legend

Figure 3.8: Result of plane patch of word on right wall

Patch	**PSNR(dB)**
Billboard on the left wall	35.8293
Word on right wall	32.0190

Table 3.2: PSNR of different patches for rectified images

The PSNR (in dB) is defined as:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned} \tag{3.49}$$

Here, MAXI is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255.

PSNR is most commonly used to measure the similarity of corresponding images. The PSNR of these three patches is shown in Table 3.2. The typical value for the PSNR for image registration are between 30 and 50 dB, provided the bit depth is 8 bits, where higher is better. As the difference images show, the results are good, but not perfect. The reason is that the example image is not perfect rectified, so optimization of H_∞ and $\boldsymbol{e}$ is also needed, just as what is done in next section.

3.2.5 Unrectified Images

Research is extended from special cases to general cases. In last section, the result of the algorithm on rectified stereo images is shown. Next, the algorithm will be applied to normal case, unrectified images, which appears more often in the practical applications. There is no special parameters for normal case, so the whole iterative process shown in Figure 3.4 will be implemented. The process is divided into two parts: Optimize $\vec{Q}^k, \vec{R}^k$ per patch separately with constant $H_\infty^{(k)}, \boldsymbol{e}^{(k)}$ and Optimize $H_\infty^{(k)}, \boldsymbol{e}^{(k)}$ for all patches with constant $\vec{Q}^{(k)}, \vec{R}^{(k)}$, where $\vec{Q}^{(k)} = (\vec{q}_1^T, \vec{q}_2^T, \cdots, \vec{q}_n^T)^T$ and $\vec{R}^{(k)} = (\vec{r}_1^T, \vec{r}_2^T, \cdots, \vec{r}_n^T)^T$ with $\vec{r}_n^T = (r_{n1}, r_{n2})$. The first steps Optimize $\vec{Q}^k, \vec{R}^{(k)}$ will be done just like the process in Subsection 3.2.4, the only difference is that H_∞^k and $\boldsymbol{e}^k$ are not always I_3 and $(1,0,0)^T$ but result of the last iterative process $H_\infty^k, \boldsymbol{e}^k$.

So in this section, we will mainly discuss the second part. Detailed calculation process for the second part is shown next. After the step Optimize $\vec{Q}^k, \vec{R}^{(k)}$, the $\vec{q}_n^T$ and $\vec{r}_n^T$ for all patches are got and can be regarded as constant values in this step. So there are only parameter H_∞

and $\boldsymbol{e}$ to be optimized. The $\vec{\beta}$ (eq. (3.17)) becomes

$$\vec{\beta} = \begin{pmatrix} \mathrm{vec}(H_\infty) \\ \boldsymbol{e} \end{pmatrix} \tag{3.50}$$

the Jacobian matrix $\mathbf{J}_{F_n}$ for patch n:

$$\mathbf{J}_{F_n} = \begin{pmatrix} \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{11})}{\partial \mathrm{vec}(H_\infty)} & \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{11})}{\partial \boldsymbol{e}} \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{12})}{\partial \mathrm{vec}(H_\infty)} & \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{12})}{\partial \boldsymbol{e}} \\ \vdots & \vdots \\ \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{ij})}{\partial \mathrm{vec}(H_\infty)} & \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{ij})}{\partial \boldsymbol{e}} \end{pmatrix} \tag{3.51}$$

where $\boldsymbol{x}_{ij} = (\boldsymbol{x}'_{ij}, 1)^T$ and $\boldsymbol{x}_{ij}$ means the i row j column in every patch. They are not the same for different patches. Combine all $\mathbf{J}_{F_n}$, the total $\mathbf{J}_F$ is

$$\mathbf{J}_F = \begin{pmatrix} \mathbf{J}_{F_1} \\ \mathbf{J}_{F_2} \\ \vdots \\ \mathbf{J}_{F_n} \end{pmatrix} \tag{3.52}$$

The total $\mathbf{J}_F$ matrix can be calculated explicit by bringing $\frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n}$ (eq. (3.24)), $\frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{ij})}{\partial \mathrm{vec}(H_\infty)}$ (eq. (3.29)) and $\frac{\partial T_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{ij})}{\partial \boldsymbol{e}}$ (eq. (3.30)) into the equation above.

The next step is calculating the total mapping error $\vec{E}$. Just like mentioned in Subsection 3.2.4 (eq. (3.45)), the mapping error $\vec{e}_n$ can be expressed as

$$\vec{e}_n = \begin{pmatrix} R_n(I_1(N(\boldsymbol{x}_{11}))) - F_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{11}) \\ R_n(I_1(N(\boldsymbol{x}_{12}))) - F_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{12}) \\ \vdots \\ R_n(I_1(N(\boldsymbol{x}_{ij}))) - F_n(H_\infty, \boldsymbol{e}, \boldsymbol{x}_{ij}) \end{pmatrix} \tag{3.53}$$

Put all $\vec{e}_n$ together in $\vec{E}$:

$$\vec{E} = \begin{pmatrix} \vec{e}_1 \\ \vec{e}_2 \\ \vdots \\ \vec{e}_n \end{pmatrix} \tag{3.54}$$

Finally $\vec{\beta}$ is iteratively optimized by

$$\vec{\beta}^{(k+1)} = \vec{\beta}^{(k)} + \left(\mathbf{J}_F^T \mathbf{J}_F\right)^{-1} \mathbf{J}_F^T \vec{E} \tag{3.55}$$

The next step is to implement the algorithm in Python and apply it to example unrectified stereo images. The algorithm will be evaluated on synthetic test images for which the ground truth is available. The details are shown in Section 4.2. The template image I_1 and target image I_2 are shown in Figure 3.9. There are three planes in the image, which show a deer, a bouquet of flowers and a cat. Therefore I simply call them deer plane, flower plan and cat plane.

Because the testing image pair is synthetically generated, the ground truth is available as well. I will use them to have a good initialization of parameters H_∞, $\boldsymbol{e}$ and $\vec{Q}$, which could also be got with the GPS and INS information or EXIF data in practical application.

Just like what is done for rectified stereo images, the final result H_∞, $\boldsymbol{e}$, $\vec{Q}$ and R are used to warp the target image to the template image shown in Figure 3.10. And the PSNR is shown in Table 3.3. In order to evaluate the result of the proposed algorithm, the root mean squared displacement (RMSD) is introduced here.

Definition 3.8 (Root mean squared displacement) *Because we know the ground truth (homography matrices of different regions of interest H_{ngt}) of the self-built image pairs. The ground truth target $\boldsymbol{y}'_{ij_{gt}} = N(H_{ngt} \cdot \boldsymbol{x}_{ij})$ can be calculated. Then the position $\boldsymbol{y}'_{ij}$ of warped point can be calculated with the estimated homography matrices H_n:*

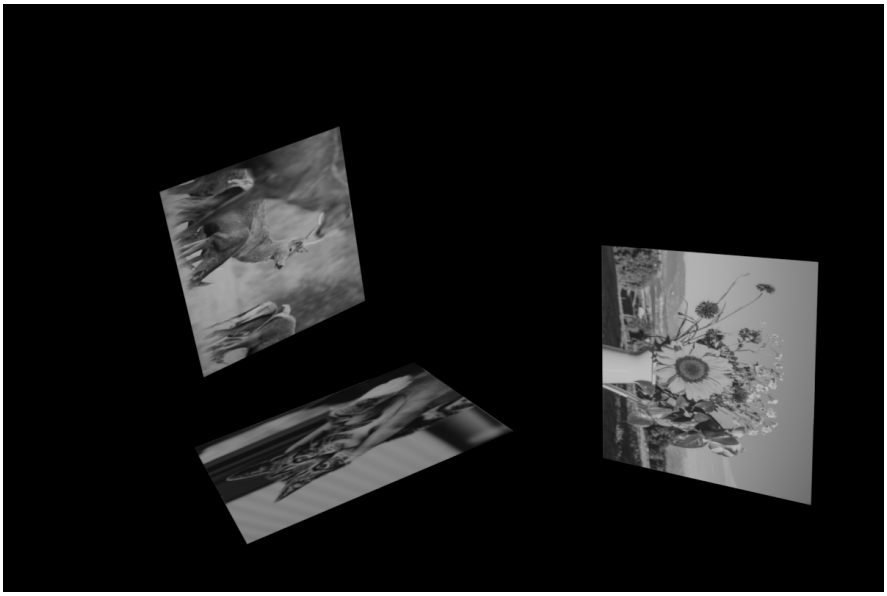
$$\boldsymbol{y}'_{ij} = N(H_n \cdot \boldsymbol{x}_{ij}) \tag{3.56}$$

Root mean squared displacement (RMSD) is defined as

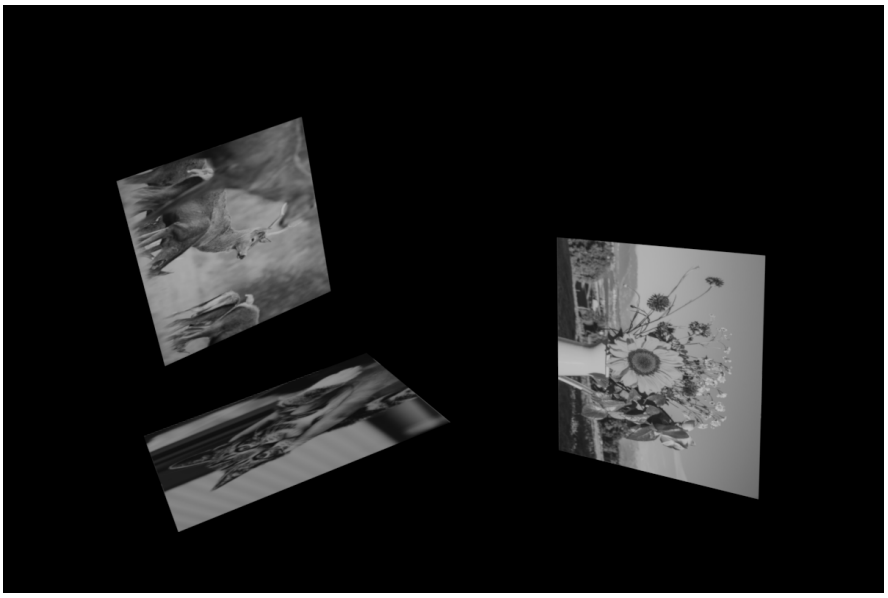
$$\text{RMSD} = \sqrt{\frac{1}{n} \sum_{\boldsymbol{x}_{ij} \in ROI} \|\boldsymbol{y}'_{ij} - \boldsymbol{y}'_{ij_{gt}}\|^2} \tag{3.57}$$

where n means the number of points in the region of interest (patches) and $\|\boldsymbol{y}'_{ij} - \boldsymbol{y}'_{ij_{gt}}\|$ is the distance d between tow points. RMSD is the measure of the average distance between two vector space, here means how much the difference between the estimated result and ground truth.

The root mean squared displacement (RMSD) and maximum displacement of all patches are shown in Table 3.4. From the difference images, we can see visually that there is basically no difference. From the PSNR analysis, the value is relatively high. So the result is acceptable. And because the whole parameters are optimized, there are no large matching error as in the rectified case (Figure 3.8)

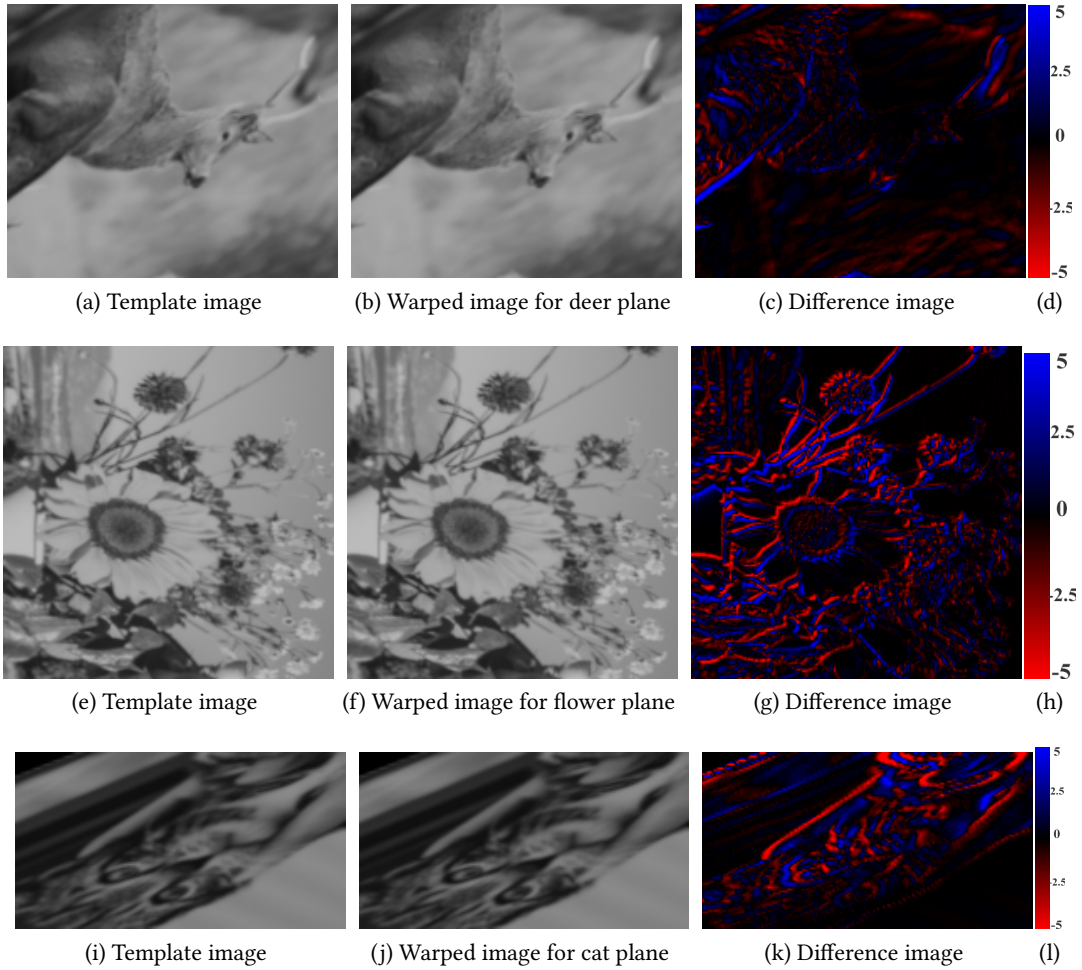


(a) Template image



(b) Target image

Figure 3.9: Example unrectified images



(a) Template image (b) Warped image for deer plane (c) Difference image (d)

(e) Template image (f) Warped image for flower plane (g) Difference image (h)

(i) Template image (j) Warped image for cat plane (k) Difference image (l)

Figure 3.10: Result of unrectified Images

Patch	**PSNR(dB)**
Deer Plane	44.9857
Flower Plane	38.3061
Cat Plane	41.5518

Table 3.3: PSNR of different patches for unrectified images

Patch	**RMSD (pixel)**	**Maximum(pixel)**
Deer Plane	0.186577	0.370456
Flower Plane	0.130046	0.162404
Cat Plane	0.369888	0.874363
All Patches	0.215594	0.874363

Table 3.4: Displacement of different patches for unrectified images

In this chapter I only show one example result of the algorithm for each case. More results are listed evaluation in Chapter 4.

3.3 Program

In this section, I mainly introduce the methods and theories related to programming. For example, during programming, the library OpenCV is used, but some basic information, like which kind of coordinate it uses for function "warpPerspective", is not included in the documentation of it. So I had to figure them out. What's more, how to blur images and calculate image derivatives are also mentioned here. In the end, dyadic program is introduced.

3.3.1 Image Warping

In order to do image registration, the function to warp image is needed. In the thesis, I have chosen the library OpenCV for warping images. How to warp the target image to the template image with the calculated homography matrix is a very important step of implementation.

Because homography is a perspective transformation, OpenCV calls the function "warpPerspective" to warp an image with a homography matrix. The function applies a perspective transformation to an image. The introduction of it in OpenCV documentation [ope] is listed here:

```
1  cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[,
       borderValue]]]])
```

Listing 3.1: Model of warpPerspective

where the parameters:

- **src** - input image
- **M** - 3×3 homography transformation matrix
- **dsize** - size of the output matrix

- **flags** - combination of interpolation methods(INTER_LINEAR or INTER_NEAREST) and the optional flag WARP_INVERSE_MAP, that sets M as the inverse transformation ($dst \rightarrow src$)
- **borderMode** - piexl extrapolation methode (BORDER_CONSTANT or BORDER_REPLICATE).
- **borderValue** - value used in case of a constant border; by default, it equals 0.

The function transforms the source image using the specified matrix:

$$dst(x, y) = src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{31}y + M_{33}}\right) \tag{3.58}$$

when the flag " WARP_INVERSE_MAP" in the function is set, Otherwise, the transformation is first inverted with **invert()** and then put in the formula above instead of **M**.

According to this, the target image I_2 is put into the function "warpPerspective" as **src**. And $H_n = H_\infty + \vec{e} \cdot \vec{q}_n^T$ is set as **M**. And the **flags** are INTER_LINEAR and WARP_INVERSE_MAP. The output of the function is a warped image which is used to show the estimation result and compare with the template image I_1 to get the mapping error $\vec{e}_n$.

But what we optimize in the program are H_∞, $\vec{e}$ and $\vec{q}_n^T$. There isn't explicit homography matrix H_n in program. In order to warp the image with H_∞, $\vec{e}$ and $\vec{q}_n^T$, a new function based on "warpPerspective" is built to warp the image in the program:

```
1  def warp_image(q, e, H_inf, target_img, template_img):
2    H = H_inf + e @ q.T
3
4    width_x = np.array(template_img).shape[1]
5    height_y = np.array(template_img).shape[0]
6
7    warped_img = cv2.warpPerspective(target_img, H, (width_x,
         height_y),
8    flags=cv2.INTER_LINEAR + cv2.WARP_INVERSE_MAP)
9  return warped_img
```

Listing 3.2: warp_image

Although the openCV documentation describes the parameters of functions in detail, the specific description of the coordinate system is not involved. For example, the specific location of the origin, the direction of the coordinate axis, and how to perform homogeneous and dehomogeneous are not clearly indicated. But for image registration technology, accuracy is

very important. Therefore, the above problems need to be explored and the program must be adjusted accordingly to ensure accuracy.

First tests with the function reveal that x-axis points right, y-axis points downwards and z-axis for homogenization and dehomogenization of the homogeneous coordinates. But it can be only seen that the origin of pixel coordinate is somewhere on the top left corner. There are two types of pixel coordinate.

1. The origin is at the middle of the upper left corner pixel of the image with the positive Row axis pointing downwards and the positive Column axis pointing towards the right [MIS].

2. The origin is at the upper left corner of the upper left pixel of the image with the positive Line axis pointing downwards and the positive Column axis pointing towards the right [MIS].

To clarify this problem, a little test program is used. Assumed that a simple image (shown in Figure 3.11a) with only 4 pixel, which is

$$\begin{bmatrix} 100 & 170 \\ 170 & 170 \end{bmatrix}$$

is transformed by a homography matrix

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which is corresponds to scale the template image by factor 2.(In function "warpPerspective", **flag** WARP_INVERSE_MAP is not used in the test.) And if the gray value of a quarter area in the upper left corner in the warped image is 100, this proves that the origin is at the upper left corner of the upper left pixel of the image in the function "warpPerspective". Otherwise, the type 1 is used.

The result should is shown in Figure 3.11b. It's obvious from the result, that the function regards the middle of pixel at upper left corner as the origin of the pixel coordinate. So I will use this definition in the thesis, too.

(a) Testing image (b) Result image

Figure 3.11: Origin of the coordinate system used by "warpPerspective"

3.3.2 Image Pyramid

In cases, images contain much noise. In image processing, it will influence the result. So before implementation of the algorithm, we must reduce the noise in image. There are some low pass filters, which can be used for image blurring. Here I have chosen the most common filter, Gaussian filter to make a Gaussian blur. In addition, it's also used in the thesis to make a Gaussian pyramid of images to reduce the effects of repeating structure and allow for a large range of convergence of the algorithm.

The proposed algorithm is based on the least square method. Therefore, the less blurry the image, the more noise will be introduced into the iterative process, and the greater the interference to the final result. At the same time, the algorithm will converge slowly because of noise influence. To avoid this problem, image pyramid is used in the program.

Go further, when there is repeating structures (texture) in the target plane such as brick floor, the repetitive structure can be almost blurred away with image pyramid and there is enough other structure left that make the algorithm converge.

In the program, the function "pyrDown" from OpenCV is used to build Gaussian image pyramid(An example image pyramid built by this function is shown in Figure 3.12).

```
1  cv2.pyrDown(src[, dst[, dstsize[, borderType]]])
```

Listing 3.3: Image pyramid

With this function, an image pyramid as a set of layers in which the higher the layer, the smaller the size is build.

After building an image pyramid of testing image, the algorithm will first calculate at the highest layer of the pyramid, which is the layer with the lowest resolution. When the preliminary result is obtained, it will be re-scaled into next layer and used as the initialization of next layer. With this initialization, the algorithm runs on the next layer. Repeat this until the last layer or the original image. Then the result has no offset and more accurate. At the same time the algorithm converges faster. Because it takes less time to converge, when the algorithm is applied to the lower resolution image. With a prior result for initialization of the higher resolution image, it also reduces the running time of the algorithm.

(a) Image in 3rd layer

(b) Image in 2nd layer

(c) Image in 1st layer

Figure 3.12: Visual representation of an image pyramid with 3 levels

3.3.3 Image Derivatives

When talking about the derivative of an image, you're actually talking about what's called a discrete derivative, and it's more of an approximation of the derivative. One simple example is that you can take the derivative in the x-direction at pixel $\boldsymbol{x}'_{ij}$ by taking the difference between the pixel values to the left and right of your pixel. It's widely used in edge detection. But in my thesis, only the first derivation of image is needed to approximate the term $\frac{\mathrm{d}I_2(N)}{\mathrm{d}N}$ in eq. (3.23):

$$j_{nu} = \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \frac{\partial T_n(\vec{\beta}, \boldsymbol{x}_{ij})}{\partial \beta_u}$$

In computer version, image derivatives can be computed by using small convolution filters of size 2×2 or 3×3, such as Sobel, Roberts and Prewitt operators. However, a large mask will generally give a better approximation of the derivative and examples of such filters are Gaussian derivatives and Gabor filters. Sometimes high frequency noise needs to be removed and this can be incorporated in the filter so that the Gaussian kernel will act as a band pass filter in the process. The most used derivatives filter kernel is shown in Figure 3.13.

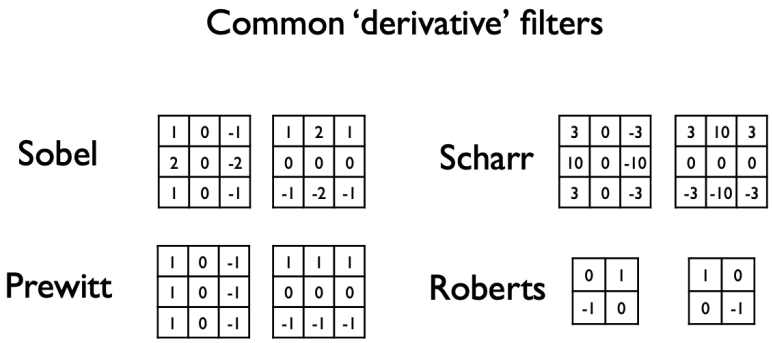


Figure 3.13: Derivative filter

In the Library OpenCV, there is a function "cv2.Sobel" to calculate the first, second, third or mixed derivatives using extended Sobel operator. This is used to calculate the image derivatives in the implementation.

The Sobel operator uses two 3×3 kernel (usually) which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define A as the source image, and G_x and G_y are two images which at each point contain the vertical and horizontal derivative approximations respectively, the computations

are as follows [20b]:

$$G_x = \frac{1}{8}\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \frac{1}{8}\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{3.59}$$

where $*$ here denotes the 2-dimensional signal processing convolution operation.

The function "cv2.Sobel" in OpenCV [ope]shows:

```
cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[,
    borderType]]]]])
```

Listing 3.4: Model of Sobel Filter

However, the Sobel operator, while reducing artifacts associated with a pure central differences operator, does not have perfect rotational symmetry. Scharr looked into optimizing this property. Scharr operators result from an optimization minimizing weighted mean squared angular error in Fourier domain. This optimization is done under the condition that resulting filters are numerically consistent. Therefore, they really are derivative kernels rather than merely keeping symmetry constraints [20b].

There is also a special value $ksize = CV\_SCHARR(-1)$ of function "cv2.Sobel" that corresponds to the 3×3 Scharr filter in function that gives more accurate results than the Soble operator. The Scharr aperture is:

$$\frac{1}{32}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

for the x-derivative, or transposed for the y-derivative.

The function combine Gaussian smoothing and differentiation, so the result is more or less resistant to the noise. Most often, the function is called with (xorder =1, yorder =0) or (xorder =0, yorder =1) to calculate the first x- or y- image derivative.

In the thesis, what needed in program is $\frac{\mathrm{d}I_2(N)}{\mathrm{d}N}$, with

$$\begin{aligned} \boldsymbol{y}'_{ij} &= N(\boldsymbol{y}_{ij}) \\ \boldsymbol{y}_{ij} &= (H_\infty + \boldsymbol{e} \cdot \vec{q}_n^T)\boldsymbol{x}_{ij} \end{aligned}$$

The meaning of this term is the derivative of pixel $\boldsymbol{y}'_{ij}$ in image I_2. To get this, the exact $\boldsymbol{y}'_{ij}$ corresponding to $\boldsymbol{x}'_{ij}$ should be known. So the function "warp_image" in Subsection 3.3.1

combining function "cv2.Sobel" can complete this task. Function "warp_image" calculates $\boldsymbol{y}'_{ij}$ and function "cv2.Sobel" calculates the derivative. An example is shown below:

1. First, use function "cv2.Sobel" function to calculate the first x and y derivative of origin image I_2:

```
ImageDerivativeX = cv2.Sobel(target_img, -1, 1, 0, ksize=-1)
ImageDerivativeY = cv2.Sobel(target_img, -1, 0, 1, ksize=-1)
```

Listing 3.5: Derivative in x and y direction

2. Then the output of last step should be warped with function "warp_image" and H_∞, $\vec{e}$ and $\vec{q}_n^T$:

```
ImageDerivativeX_warp = self.warp_image(q, e, H_inf,
    ImageDerivativeX, template_img)
ImageDerivativeY_warp = self.warp_image(q, e, H_inf,
    ImageDerivativeY, template_img)
```

Listing 3.6: $\frac{\mathrm{d}I_2(N)}{\mathrm{d}N}$

3.3.4 Dyadic Program

Generally speaking, the images that are the objects of our algorithm contain all millions to ten of millions pixels. So when I apply the algorithm, the dimension of eq. (3.52) will reach 20 million even more. In this way, when we calculate pseudo-inverse of $\mathbf{J}_F$, it will require a very large memory space and running time, which greatly increases the computation time of algorithm. In order to avoid this problem, I will optimize it during programming, so as to improve the reaction speed of the program.

The main idea is to use dyadic product to build a dyadic program. In mathematics, specifically multilinera algebra, a dyadic or dyadic tensor is a second order tensor, written in a notation that fists in with vector algebra. The dyadic product takes in two vectors and returns a second order tensor called a "dyadic". A dyadic can be used to contain physical or geometric information, although in general there is no direct way of geometrically interpreting it.

It also has some aspects of matrix algebra, as the numerical components of vector can be arranged into row and column vectors, and those of second order tensors in square matrix. Dyadic expressions may closely resemble the matrix equivalents.

Here I will use the process of calculating eq. (3.47) in step Optimize $\vec{Q}^k, \vec{R}^k$ per patch to

explain the details of using dyadic product. In each cycle, the term

$$\left(\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}\right)^{-1} \mathbf{J}_{F_n+R_n}^T \vec{e}_n$$

should be calculated. Rewrite $\mathbf{J}_{\mathbf{F_n+R_n}}$ in this form:

$$\mathbf{J}_{\mathbf{F_n+R_n}} = \begin{pmatrix} B_1^T \\ B_2^T \\ \vdots \\ B_{i\times j}^T \end{pmatrix} \tag{3.60}$$

where

$$B_{i\times j}^T = \begin{pmatrix} \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{ij})}{\partial \vec{q}_n} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{1n}} & -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{2n}} \end{pmatrix} \tag{3.61}$$

$$= \begin{pmatrix} b_{i\times j}^{(1)} & b_{i\times j}^{(2)} & b_{i\times j}^{(3)} \end{pmatrix} \tag{3.62}$$

with

$$\begin{aligned} b_{i\times j}^{(1)} &= \frac{\mathrm{d}I_2(N)}{\mathrm{d}N} \cdot \frac{\mathrm{d}N(T_n)}{\mathrm{d}T_n} \cdot \frac{\partial T_n(\vec{q}_n, \boldsymbol{x}_{ij})}{\partial \vec{q}_n} \\ b_{i\times j}^{(2)} &= -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{1n}} \\ b_{i\times j}^{(3)} &= -\frac{\partial R_n(I_1(N(\boldsymbol{x}_{ij})))}{\partial r_{2n}} \end{aligned}$$

and $b_{i\times j}^{(1)}$ is a 1×3 vector, $b_{i\times j}^{(2)}$ and $b_{i\times j}^{(3)}$ are both scalar each. So the dimension of $B_{i\times j}^T$ is 1×5 in this case

So the term $\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}$ looks like

$$\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n} = \begin{pmatrix} B_1 & B_2 & B_3 & \cdots & B_{i\times j} \end{pmatrix} \cdot \begin{pmatrix} B_1^T \\ B_2^T \\ \vdots \\ B_{i\times j}^T \end{pmatrix} = \sum_1^{i\times j} B_s \cdot B_s^T \tag{3.63}$$

And M is defined as a temporary represent of $\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}$ and is initialized as zero matrix M_0 with the same dimension of term $B_s \cdot B_s^T$ (For Optimize $\vec{Q}^k, \vec{R}^k$, it's 5×5). So for each pixel in

the plane patch n, M_s can be calculated by accumulating:

$$M_s = M_{s-1} + B_s \cdot B_s^T \tag{3.64}$$

After addition of all pixels in the plane patch n, $M_{i\times j} = \mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}$. In this way, the matrix multiplication is changed to addition with dyadic product $\sum_1^{i\times j} B_s \cdot B_s^T$.

At the same time apply this method to term $\mathbf{J}_{F_n+R_n}^T \vec{e}_n$, and get the result $W_{i\times j}$ (Here it's a 5×1 vector.) with the same form of $M_{i\times j}$. Then the final result can be got:

$$\left(\mathbf{J}_{F_n+R_n}^T \mathbf{J}_{F_n+R_n}\right)^{-1} \mathbf{J}_{F_n+R_n}^T \vec{e}_n = M_{i\times j}^{-1} \cdot W_{i\times j} \tag{3.65}$$

for plane patch n. In each iteration step, only a storage space of matrix with the same dimension of $B_s \cdot B_s^T$ replace a storage space of matrix with $i \times j$ times dimensions. With this optimization, the speed of matrix inversion is also greatly reduced. One example of code implementation for rectified case is shown below: (This function only show a single loop in iterative calculation of Optimize $\vec{Q}^k, \vec{R}^k$.)

```
def improve_parameter(self, q, e, H_inf, r_correct, target_img,
    template_img, *correspondings_field):

  M = zeros((5, 5))
  W = zeros((5, 1))

  # warp_image is a function which I use warpPerspective to build,
     to warp the image.
  warped_img = warp_image(q, e, H_inf, target_img, template_img)

  ImageDerivativeX = Sobel(target_img, -1, 1, 0, ksize=-1)
  ImageDerivativeX_warp = warp_image(q, e, H_inf, ImageDerivativeX,
     template_img)
  ImageDerivativeY = Sobel(target_img, -1, 0, 1, ksize=-1)
  ImageDerivativeY_warp = warp_image(q, e, H_inf, ImageDerivativeY,
     template_img)

  for field in correspondings_field:
    x_min_improve = field[0]
    x_max_improve = field[0] + field[1]
    y_min_improve = field[2]
    y_max_improve = field[2] + field[3]
    for y in range(y_min_improve, y_max_improve, 1):
```

```
20        for x in range(x_min_improve, x_max_improve, 1):
21          # coordinate in template img
22          x_ij = np.array([[x],[y],[1]])
23
24          # dimension fo H is 3*3
25          H = (H_inf + e @ q.T) @ x_ij
26
27          # dimension of D_N is 2*3
28          D_N = np.array([[1, 0, 0], [0, 1, 0]]) / H[2] - np.matmul(H
                [0:2], np.array([[0, 0, 1]])) / (H[2] * H[2])
29
30          # Derivative of I_2 (warped img)
31          D_I_12 = np.array([ImageDerivativeX_warp[y, x],
                ImageDerivativeY_warp[y, x]]).reshape(1, 2)
32
33          # D_I after dehomogenous 1*3
34          D_I_dehomogenous = D_I_12 @ D_N
35
36          # B_1 is the part for Delta p in A_matrix. Dimension is 1 *
                3
37          B_1 = D_I_dehomogenous @ e @ x_ij.T
38
39          # B_2 is the part for Delta a and Delta b in A_matrix.
                Dimension is 1 * 2
40          B_2 = np.array([[template_img[y, x], 1]])
41
42          # B is the factor of variable, dimension is 1*5
43          B = np.concatenate((B_1, -B_2), axis=1)
44
45          # difference between template image and warped image
46          d = r_correct[0, 0] * int(template_img[y, x]) + r_correct[1,
                0] - int(warped_img[y, x])
47
48          # dimension of M_matrix is 5 * 5
49          # dimension of W_matrix is 5 * 1
50          M = M + B.T @ B
51          W = W + d * B.T
52
53    # dimension of q is 5 * 1. pinv is pesudo-inverse
54    Delta = np.linalg.pinv(M) @ W
```

```
55
56     return Delta
```

Listing 3.7: Dyadic product example

4 Evaluation

This chapter presents the evaluation results of my algorithm using two benchmarks: first with the Public Dataset Middlebury [SSZ01] for rectified images and with a syntetic dataset generated by myself for this evaluation. Before that, I also introduce some other public datasets and the process of building own datasets.

4.1 Public Datasets

So far, we have completed the theoretical derivation of our algorithm and it's implementation program is already applied to some example images, some preliminary results are obtained. In this chapter, I will go through a deeper test of the algorithm and get a more comprehensive evaluation.

Testing of course requires testing datasets. Generally speaking, there are two ways to get testing datasets. One is using published public datasets, and the other is to build datasets by own. In this chapter, we focus on public datasets.

The application scenarios of my algorithm is aerial images and there must be plane structures in the images. What's more, the dataset must be images pairs or consecutive images for the same scene. So the dataset must meet at least two conditions:

1. Image pair for the same scene
2. Plane structures

There are many public stereo images dataset. For example, New Tsukuba Stereo Dataset [Nak+96], Middlebury Stereo Vision Datasets [SSZ01], AdelaideRMF [Won+11].

The most scenes in the dataset New Tsukuba Stereo Dataset are office interior scene. This dataset contains 1800 stereo pairs with ground truth disparity maps, occlusion maps and discontinuity maps that will help to further develop the state of the art of stereo matching algorithms and evaluate its performance. It has been generated using photo-realistic computer graphics techniques and modeled after the original "head and lamp" stereo scene released by University of Tsukuba in 1997. The dataset is a 1 minute video sequence and also contains the

3D position and orientation of the camera on each frame, so it can also be used to develop and evaluate camera tracking methods [Nak+96].

But the prerequisites of my algorithm is that there is a plane structure in the image. Although it has some plane structures, most of them are scattered facets like Figure 4.1, or smooth planes, where almost all gray values are the same like the image shown in Figure 4.2. It's not very suitable for my algorithm. So I won't use it.

(a) Left image (b) Right image

Figure 4.1: Example image pair of New Tsukuba Stereo Dataset with scattered facets [Nak+96]

(a) Left Image (b) Right Image

Figure 4.2: Example image pair of New Tsukuba Stereo Dataset with no feature plane [Nak+96]

AdelaideRMF is a data set for robust geometric model fitting (homography estimation and fundamental matrix estimation). It collects a set of image pairs and the key-point correspondences which were obtained by SIFT matching are manually labeled[Won+11]. Unlike New Tsukuba Stereo Dataset, most of the scenes in it are different buildings, so there are a lot of large flat planes in images Figure 4.3, such as the wall of red building.

The purpose of evaluation is to evaluate the result of my algorithm, then ground truth is

(a) Left image

(b) Right image

Figure 4.3: Example image pair of AdelaideRMF Dataset [Won+11]

needed for reference. But in AdelaideRMF, there are only matching features that were obtained by SIFT matching. And they can be used to to estimate the homography matrix. But this result is still an approximate value, not a ground truth. So it's meaningless to compare with this result. In other words, this data set can only be used to show the results of our algorithm, but not to verify the accuracy of our algorithm.

(a) Barn 1

(b) Barn 2

(c) Bull

Figure 4.4: Middlebury Stereo Vision Datasets 2001 Version(1) [SSZ01]

In the end, I introduce Middlebury Stereo Vision Datasets. I mainly used the 2001 Stereo datasets in it with ground truth. There are 6 datasets of piecewise planar scenes Figure 4.4 and Figure 4.5. Because the image pairs in this dataset is all perfect rectified, they can only be use to evaluate the rectified case.

(a) Poster

(b) Swatooth

(c) Venus

Figure 4.5: Middlebury Stereo Vision Datasets 2001 Version(2) [SSZ01]

Besides these stereo image datasets, there are also some available aerial image datasets. But the most datasets don't contain stereo images. Only a small part of them have stereo images, such as ASL Datasets `http://projects.asl.ethz.ch/datasets/` and CMLA Subpixel Stereo Dataset `http://www.ipol.im/pub/pre/187/`. But they either don't have flat plane structure, or they don't have a ground truth. So I don't consider them further.

Because the public datasets, that I could find, are only partly suited for the evaluation of the proposed algorithm. I decided to build my own dataset to verify the algorithm.

4.2 Self-Built Datasets

In order to construct our own dataset, I first need a virtual environment. I choose Blender as the software for the build environment. Blender is used to manage the 3D scene and do the rendering. There can be some exact objects built by Blender in the virtual environment. In our case, we only need the plane patches in the images. So I just put some planes, which are shading with some pictures, in the scene and render the realistic images. The whole dataset is available online at: `https://github.com/Zauberr/Multi-H-Dataset`

The interface of the software is shown in Figure 4.6

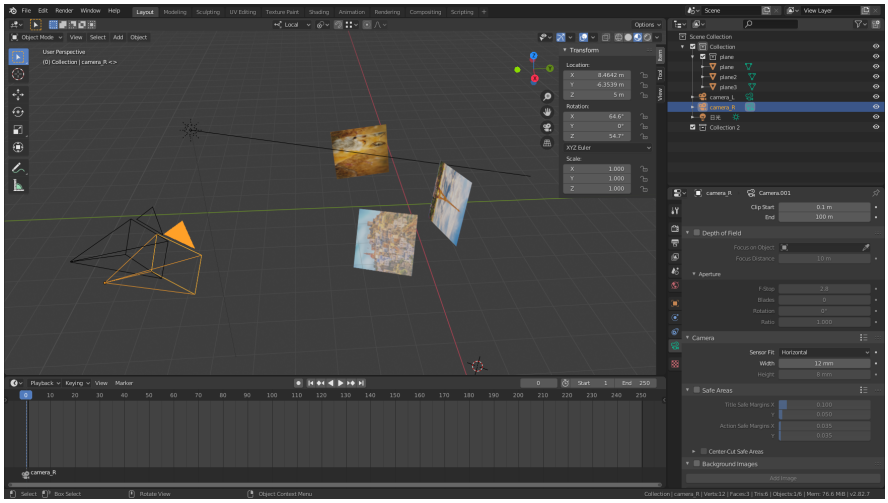
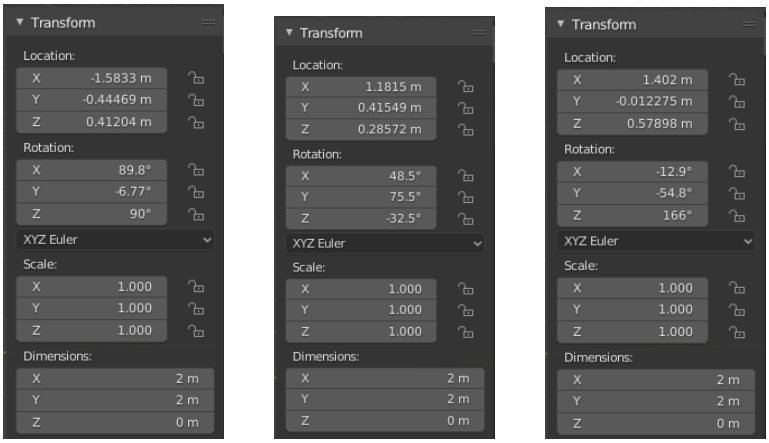


Figure 4.6: Interface of Blender

We can see from it that the central main interface is a constructed virtual scene, in which the world coordinate system is preset, red is the x-axis, green is the y-axis and blue is the z-axis. Each object (rigid body) has its own position coordinates and rotation parameters relative to the world coordinate system. In the toolbar at bottom right corner of the screen, we can set



(a) Fox (b) Eiffel Tower (c) Greece

Figure 4.7: Transformation of example planes

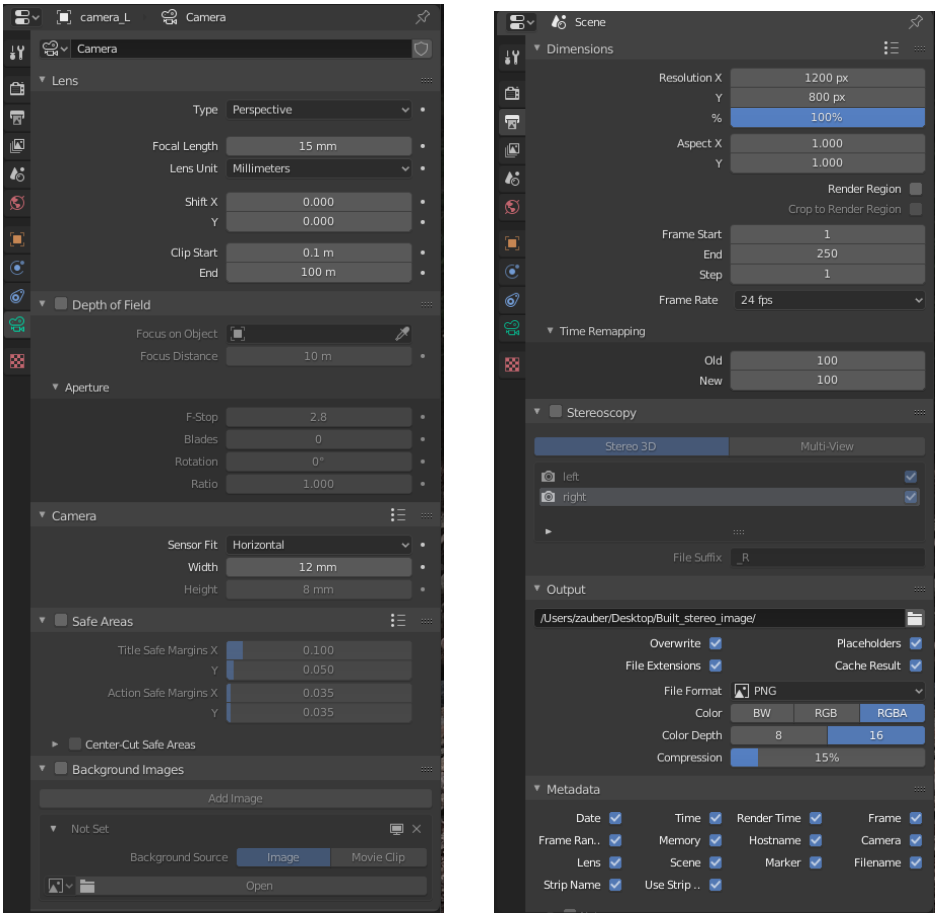
the specific parameters of each object in detail. For example, for the camera we use, we can choose different camera models and different focal lengths, etc.

After understanding the main software situation, we focus on the processing pipeline of dataset. We will use the scene in Figure 4.6 as an example to explain the entire process in detail. First of all, because the algorithm applies to plane structures, we first put three plane structures in the scene and draw different patterns on the surface of each picture. We call the plane showing the fox the "fox plane", showing Eiffel Tower "eiffel plane" and the last one "greece plane". (In the whole datasets, the used images are from `https://pixabay.com`.) The transformations of these three planes are shown in the Figure 4.7.

After successfully building the scene with the images, we need to extract the image data (render the image) from the virtual scene. In this step we use the camera components in the software. We placed a pair of stereo camera equipment consisting of two cameras with identical internal parameters (shown in Figure 4.8) in front of three planes.

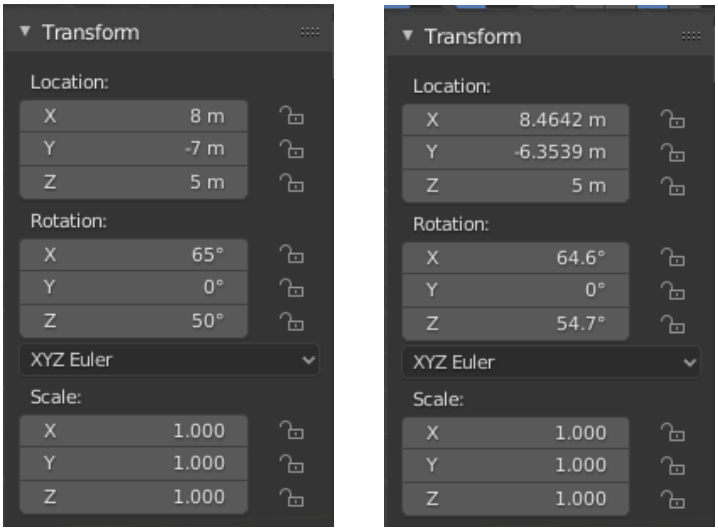
The calculation of imaging process also requires external parameters of the camera, such as rotation and translation. These parameters can also be obtained in the software. The parameters in the example are shown in Figure 4.9

After providing the light source in the scene, we have completed the construction of a simple virtual environment. After that, the scene should be rendered and collect the raw image data, which can be automatic done by the software. Next, we need to get the specific imaging formula corresponding to our image and the homography matrix of the plane in it, which is regarded as ground truth.



(a) (b)

Figure 4.8: Internal parameter of stereo cameras



(a) Left Camera (b) Right Camera

Figure 4.9: External-parameter of stereo cameras

The specific imaging process and formula are exactly the same as described in Section 3.1. Finally, we use the eq. (3.8) to solve homography matrix for each plane. And here I add how to get the parameter matrices needed in eq. (3.8) based on the actual virtual environment parameters.

For convenience, eq. (3.8) is repeated here:

$$H = K_2 R_2^T R_1 K_1^{-1} + K_2 R_2^T (\boldsymbol{c}_1 - \boldsymbol{c}_2) \cdot \frac{\vec{n}_n^T}{d_n} R_1 K_1^{-1}$$

where

- $\boldsymbol{c}_1$,$\boldsymbol{c}_2$: location of the camera center in the world coordinate system
- R_1,R_2: rotation matrix of world coordinate system relative to the camera
- K_1,K_2: calibration matrix of camera
- $\vec{n}_n$: unit outward normal vector of plane
- d_n: distance from the plane to camera center C_1

I use the virtual scene shown previously as an example to solve each parameter separately in the equation. In the process of solving each parameter, the meaning of parameters in the formula is the same as the meaning in each source picture. And the left camera is camera 1.

- $\boldsymbol{c}_1$,$\boldsymbol{c}_2$:
  From Figure 4.9, we can get the transform information for stereo camera pairs. we can use the location information shown to describe $\boldsymbol{c}_1$,$\boldsymbol{c}_2$:

$$\boldsymbol{c} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 8 \\ -7 \\ 5 \end{bmatrix} \tag{4.1}$$

- R_1,R_2:
  From Figure 4.9, we can get the transform information for stereo camera pairs. we can use the Rotation information shown to describe R_1,R_2 [Ber08], and in blender, it uses extrinsic parameters, in other words, space-fixed rotation.

$$\begin{aligned} R &= R_z(Z)R_y(Y)R_x(X) \\ &= R_z(50^\circ)R_y(0^\circ)R_x(65^\circ) \\ &= \begin{bmatrix} \cos Z \cos Y & \cos Z \sin Y \sin X - \sin Z \cos X & \cos Z \sin Y \cos X + \sin Z \sin X \\ \sin Z \cos Y & \sin Z \sin Y \sin X + \cos Z \cos X & \sin Z \sin Y \cos X - \cos Z \sin X \\ -\sin Y & \cos Y \sin X & \cos Y \cos X \end{bmatrix} \end{aligned} \tag{4.2}$$

- K_1,K_2:
  From Figure 4.8, we can get the focal length of camera f, sensor width x, sensor height y and resolution X and Y. The output pixel in the image is set as squared pixel. The scaling factors k_x, k_y are

$$k_x = k_y = \frac{\sqrt{X^2 + Y^2}}{\sqrt{x^2 + y^2}} = \frac{\sqrt{1200^2 + 800^2}}{\sqrt{12^2 + 8^2}} = 100px/mm \tag{4.3}$$

  then K_1,K_2:

$$K = K_1 = K_2 = \begin{bmatrix} fk_x & 0 & \frac{X-1}{2} & 0 \\ 0 & -fk_y & \frac{Y-1}{2} & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{4.4}$$

  where the minus for fk_y and 1 comes from converting coordinate between right-up-backwards coordinate system of Blender and right-down-forward coordinate system of OpenCV.

- $\vec{n}_n$:
  From Figure 4.7, we can get location and rotation of each plane. And the unit outward normal vector of each plane before transformation is set by software $\vec{n}_{init} = (0, 0, 1)^T$. So $\vec{n}_n$:

$$\begin{aligned}\vec{n} &= R_{plane} \cdot \vec{n}_{init} \\ &= \begin{bmatrix} \cos Z \cos Y & \cos Z \sin Y \sin X - \sin Z \cos X & \cos Z \sin Y \cos X + \sin Z \sin X \\ \sin Z \cos Y & \sin Z \sin Y \sin X + \cos Z \cos X & \sin Z \sin Y \cos X - \cos Z \sin X \\ -\sin Y & \cos Y \sin X & \cos Y \cos X \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\end{aligned} \tag{4.5}$$

- d_n:
  Here we regard the right camera as the camera 1. From Figure 4.9b, we know the location of the camera 1 $\boldsymbol{c}_1$ and from Figure 4.7, we can get a point $\vec{t} = (1, -1, 1)^T$ on the plane, which is the same as location of plane. So d_n:

$$d_n = \vec{n}_n^T \boldsymbol{c}_1 - \vec{n}_n^T \vec{t} \tag{4.6}$$

Combine all the equation above, we can finally calculate the ground truth:

- $H_\infty = K_2 R_2 R_1^T K_1^{-1}$
- $\vec{e} = K_2 R_2 (\boldsymbol{c}_1 - \boldsymbol{c}_2)$
- $\vec{q}_n^T = \frac{\vec{n}_n^T}{d_n} R_1^{-1} K_1^{-1}$
- $H_n = H_\infty + \vec{e} \cdot \vec{q}_n^T$

4.3 Testing

After completing the collection and pre-processing of the public datasets and the construction of the test dataset, we could use these images to test my algorithm. In this section, some testing results of the algorithm is shown.

4.3.1 Testing with Middlebury Stereo Vision Datasets

First, we apply my algorithm on the dataset of Middlebury Stereo Vision Datasets 2001 Version. Just like mention before, the images in the dataset are all rectified. So we use it only to test the

rectified case of the algorithm. The results of all image pairs are shown in Figure 4.10 and the PSNR are shown in Table 4.1. The root mean squared displacement (RMSD) of all patches are shown in Table 4.7.

It can be seen from the results that when the algorithm is applied to the rectified stereo images very stable. That is to say, when H_∞ and $\boldsymbol{e}$ are known, the result is usually quite accurate.

Shown	Display Patch					
Figure	(1)	(2)	(3)	(4)	(5)	(6)
Figure 4.10a	37.5768	40.4451	45.9941	42.4099	47.7764	38.3245
Figure 4.10b	46.9869	41.8912	32.6315	41.8892	36.9338	49.0576
Figure 4.10c	51.4393	44.9664	34.969591	47.4049	45.2209	

Table 4.1: PSNR (dB) of different patches in Middlebury Stereo Vision Datasets

Shown	Display Patch					
Figure	(1)	(2)	(3)	(4)	(5)	(6)
Figure 4.10a	2.252596	0.647761	0.184386	0.536102	0.954197	0.895281
Figure 4.10b	0.203423	1.123935	1.523684	0.358562	1.111534	0.102031
Figure 4.10c	0.507921	0.344584	1.353465	0.664646	0.188721	

Table 4.2: Displacement (pixel) of different patches in Middlebury Stereo Vision Datasets

4.3.2 Testing with Self-built Stereo Images

When the algorithm is applied to the image pair without ground truth, the evaluation could only be done with the difference image and PSNR. Next, we will focus on application on the self-built testing images with ground truth. The evaluation can be done with the ground truth.

I start also with the rectified stereo images. The built images is shown in Figure 4.12. And the ground truth of it was calculated in the way mentioned in Section 4.2. Using algorithm on the images, the result is shown in Figure 4.11 and the estimated parameters are in Table 4.5. PSNR is in Table 4.3. The root mean squared displacement (RMSD) and maximum displacement of all patches are shown in Table 4.4.

Patch	**PSNR(dB)**
Mercury Plane	49.1514
Sheep Plane	66.9878
Butterfly Plane	56.9686

Table 4.3: PSNR of different patches for self-built stereo images

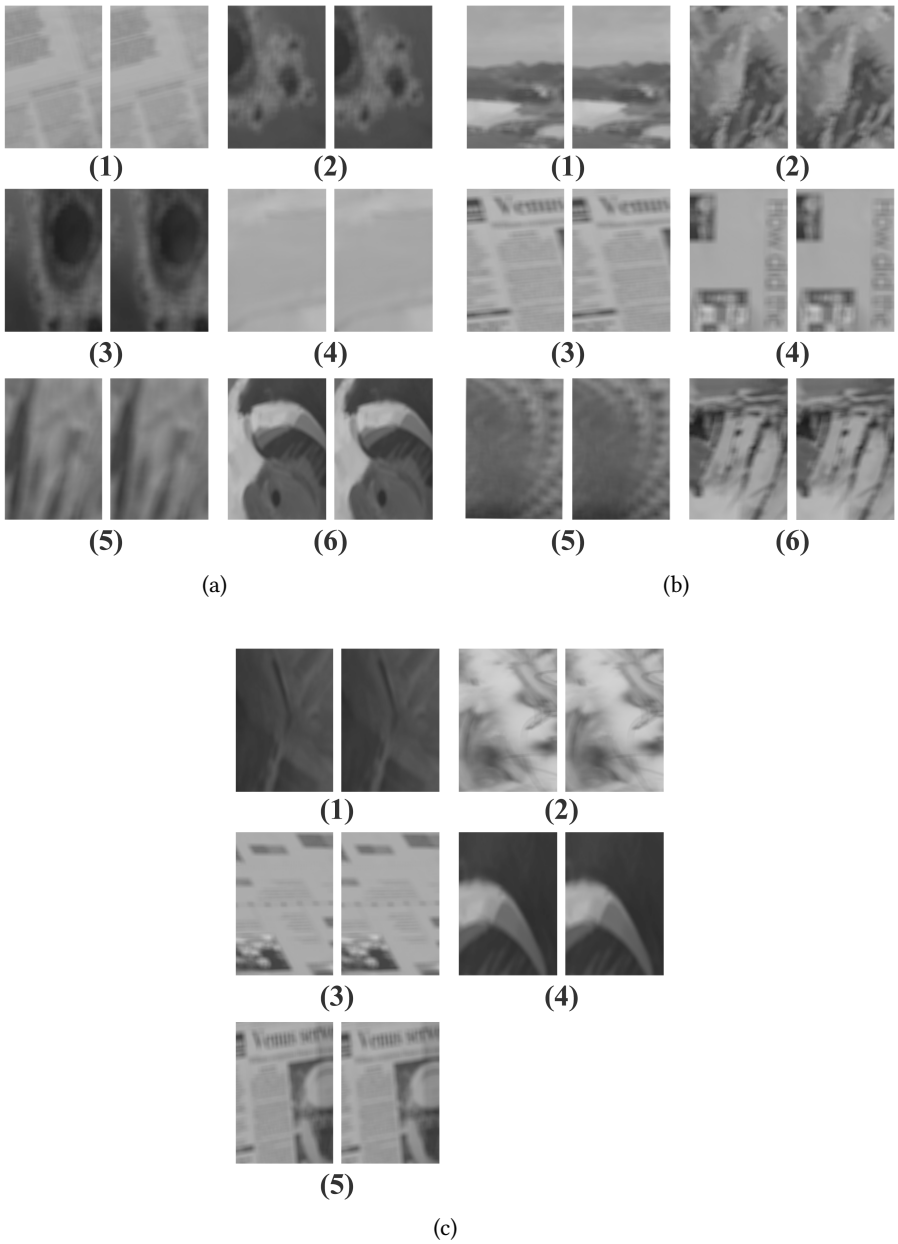
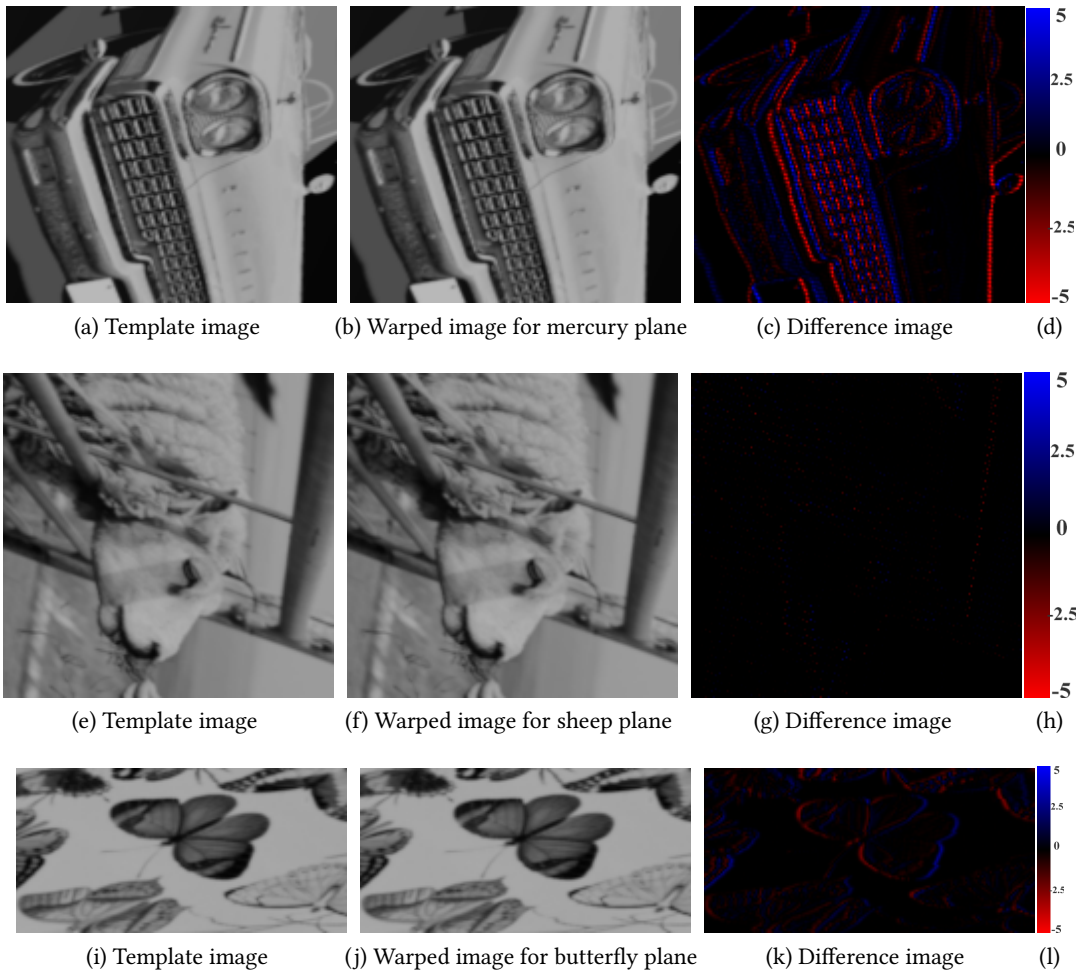


Figure 4.10: Template and transformed plane patches in Middlebury Stereo Vision Datasets



(a) Template image (b) Warped image for mercury plane (c) Difference image (d)

(e) Template image (f) Warped image for sheep plane (g) Difference image (h)

(i) Template image (j) Warped image for butterfly plane (k) Difference image (l)

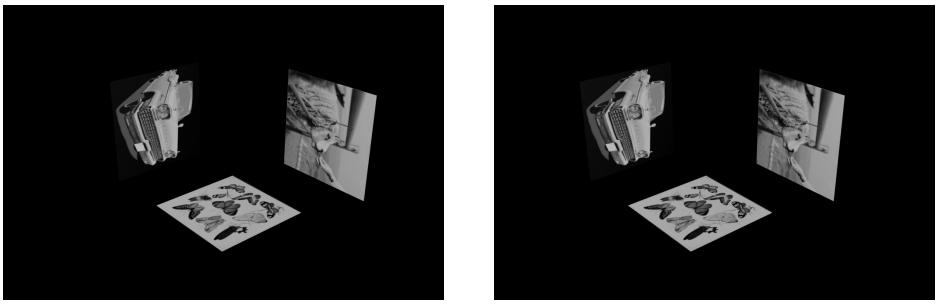
Figure 4.11: Result of self-built stereo images

Patch	**RMSD (pixel)**	**Maximum(pixel)**
Mercury Plane	0.044846	0.045849
Sheep Plane	0.003704	0.004460
Butterfly Plane	0.017901	0.018868

Table 4.4: Displacement of different patches for self-built stereo images

Parameter	Ground Truth	Estimated
H_∞	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
$\boldsymbol{e}$	$\begin{bmatrix} -5.69822333e+02 \\ -4.57549368e-03 \\ 1.52790722e-05 \end{bmatrix}$	$\begin{bmatrix} -5.69822333e+02 \\ -4.57549368e-03 \\ 1.52790722e-05 \end{bmatrix}$
$\vec{q}_1$	$\begin{bmatrix} -5.04147145e-05 \\ -2.53917154e-05 \\ 1.22084579e-01 \end{bmatrix}$	$\begin{bmatrix} -5.04285679e-05 \\ -2.54000998e-05 \\ 1.22014162e-01 \end{bmatrix}$
$\vec{q}_2$	$\begin{bmatrix} 6.80928394e-05 \\ -2.41470029e-05 \\ 4.64782197e-02 \end{bmatrix}$	$\begin{bmatrix} 6.80857568e-05 \\ -2.41376373e-05 \\ 4.64747905e-02 \end{bmatrix}$
$\vec{q}_3$	$\begin{bmatrix} 0.00000000e+00 \\ 1.09855489e-04 \\ 3.28974882e-02 \end{bmatrix}$	$\begin{bmatrix} -1.58691562e-09 \\ 1.09810544e-04 \\ 3.28889686e-02 \end{bmatrix}$
r_{11}		9.96429311e-01
r_{21}		-2.65463250e-05
r_{12}		9.94738600e-01
r_{22}		-4.20122298e-05
r_{13}		9.97044402e-01
r_{23}		-2.07652871e-05

Table 4.5: Ground truth and estimated parameter for self-built stereo images



(a) Template image (b) Target image

Figure 4.12: Self-built stereo images

Patch	**PSNR(dB)**
Brick-wall Plane	29.3573
Voronoi Plane	40.4581
Markov Plane	38.8231

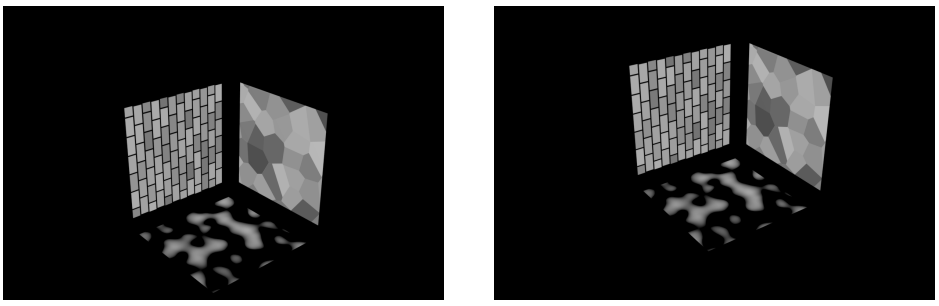
Table 4.6: PSNR of different patches for self-built images (1)

4.3.3 Testing with Self-built Normal Images

In this part, the results of testing with the self-built normal images are shown. I have chosen 3 image pairs to evaluate the algorithm.

First, a scenario with very different camera poses. The self-built images are shown in Figure 4.13. The ground truth and estimated parameters are in Table 4.8. PSNR is in Table 4.6. The root mean squared displacement (RMSD) and maximum displacement of all patches are shown in Table 4.7. The warped images are shown in Figure 4.14. From this result, we know that the result of brick-wall plane is not very good. The reason is that the algorithm sometimes responds to the texture structure not very well. In order to overcome it, we should use the image pyramid.

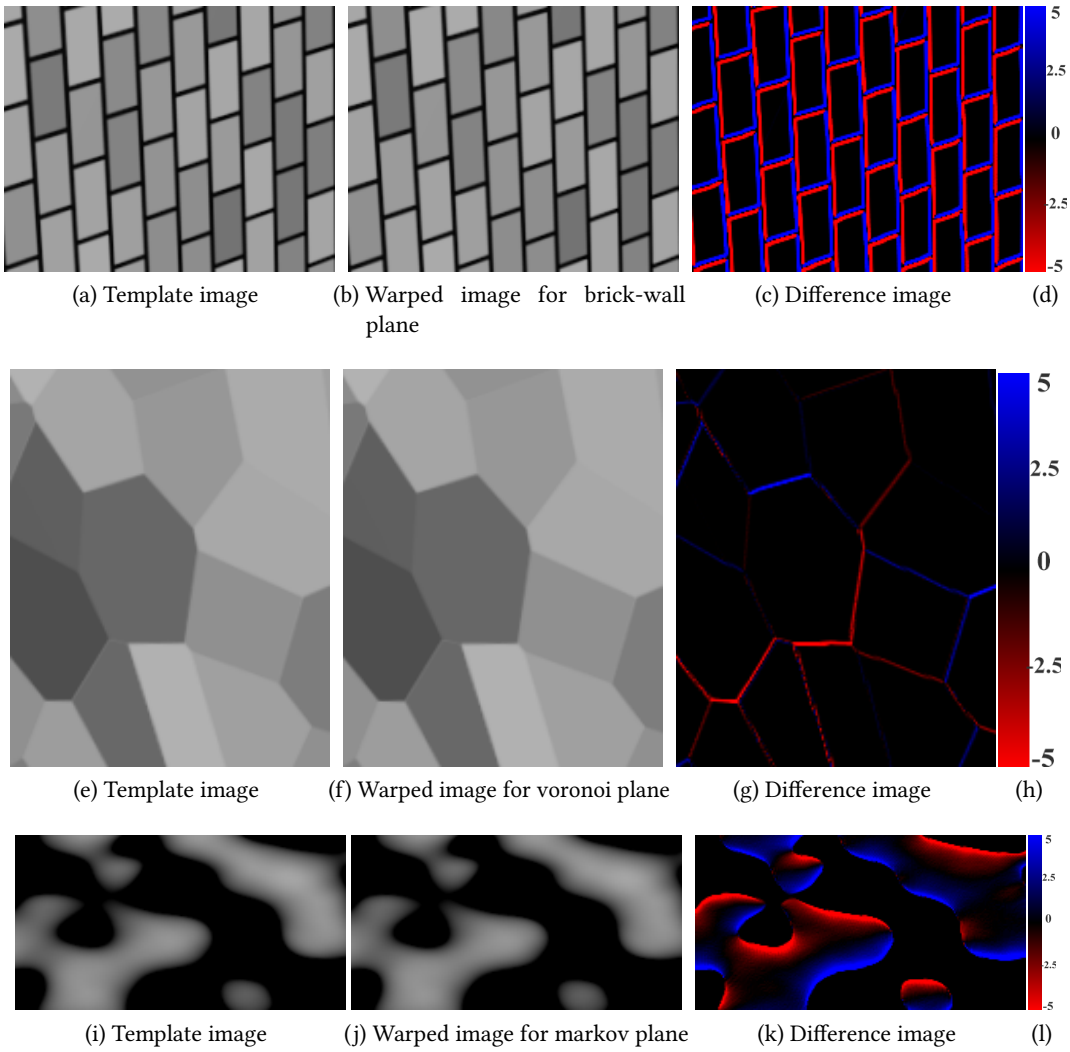
What's more, the camera poses are very different and the algorithm can also find the correct result with suitable initialization. This proves that the algorithm can be applied on any two pictures, as long as they contain the same planes.



(a) Template image (b) Target image

Figure 4.13: Self-built images (1)

The second example images pair comes from two cameras with very different poses, too. The self-built images are shown in Figure 4.15. The ground truth and estimated parameters are in Table 4.11. PSNR is in Table 4.9. The root mean squared displacement (RMSD) and maximum displacement of all patches are shown in Table 4.10. The warped images are shown in Figure 4.16. There are no texture structure in these images. So the result is all perfect.



(a) Template image (b) Warped image for brick-wall plane (c) Difference image (d)

(e) Template image (f) Warped image for voronoi plane (g) Difference image (h)

(i) Template image (j) Warped image for markov plane (k) Difference image (l)

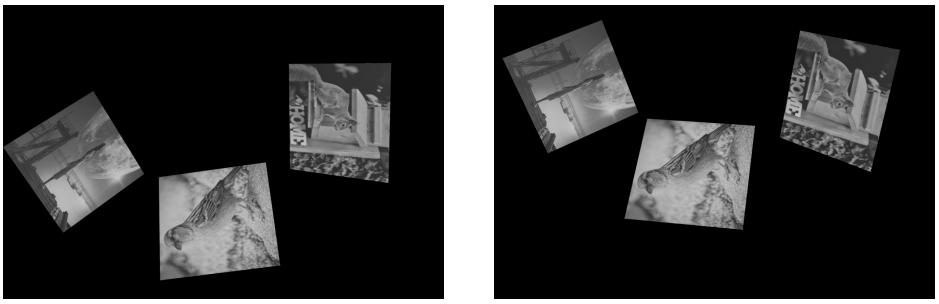
Figure 4.14: Result of self-built normal images (1)

Patch	**RMSD (pixel)**	**Maximum(pixel)**
Brick-wall Plane	0.304703	0.325780
Voronoi Plane	0.208013	0.277421
Markov Plane	0.344370	0.397986
All Patches	0.288475	0.397986

Table 4.7: Displacement of different patches for self-built images (1)

Parameter	Ground Truth			Estimated		
H_{∞}	$\begin{bmatrix} 9.65703446e-01 \\ 2.19264412e-02 \\ -5.08187535e-05 \end{bmatrix}$	$\begin{bmatrix} -8.36839711e-03 \\ 1.01576119e+00 \\ 4.74420190e-05 \end{bmatrix}$	$\begin{bmatrix} 1.39138794e+02 \\ -1.23767980e+02 \\ 1.00606213e+00 \end{bmatrix}$	$\begin{bmatrix} 9.65839836e-01 \\ 2.19485280e-02 \\ -5.05974374e-05 \end{bmatrix}$	$\begin{bmatrix} -8.41589335e-03 \\ 1.01520898e+00 \\ 4.74229252e-05 \end{bmatrix}$	$\begin{bmatrix} 1.39060980e+02 \\ -1.23718912e+02 \\ 1.00579468e+00 \end{bmatrix}$
$\boldsymbol{e}$		$\begin{bmatrix} -7.24246259e+02 \\ -1.64515557e+01 \\ 3.81296513e-02 \end{bmatrix}$			$\begin{bmatrix} -7.24204197e+02 \\ -1.64534715e+01 \\ 4.01498913e-02 \end{bmatrix}$	
$\vec{q}_1$		$\begin{bmatrix} -5.96261061e-05 \\ -3.00310957e-05 \\ 1.44390941e-01 \end{bmatrix}$			$\begin{bmatrix} -5.96507588e-05 \\ -3.02164145e-05 \\ 1.44743346e-01 \end{bmatrix}$	
$\vec{q}_2$		$\begin{bmatrix} 8.15476596e-05 \\ -2.89183354e-05 \\ 5.56620941e-02 \end{bmatrix}$			$\begin{bmatrix} 8.14932969e-05 \\ -2.88990747e-05 \\ 5.57176245e-02 \end{bmatrix}$	
$\vec{q}_3$		$\begin{bmatrix} 0.00000000e+00 \\ 1.15523713e-04 \\ 3.45949029e-02 \end{bmatrix}$			$\begin{bmatrix} 1.89395051e-07 \\ 1.14948256e-04 \\ 3.49029342e-02 \end{bmatrix}$	
r_{11}					9.93390769e-01	
r_{21}					-4.61817515e-05	
r_{12}					9.97370299e-01	
r_{22}					-1.80821342e-05	
r_{12}					9.96350068e-01	
r_{22}					-3.17773268e-05	

Table 4.8: Ground truth and estimated parameter (1)

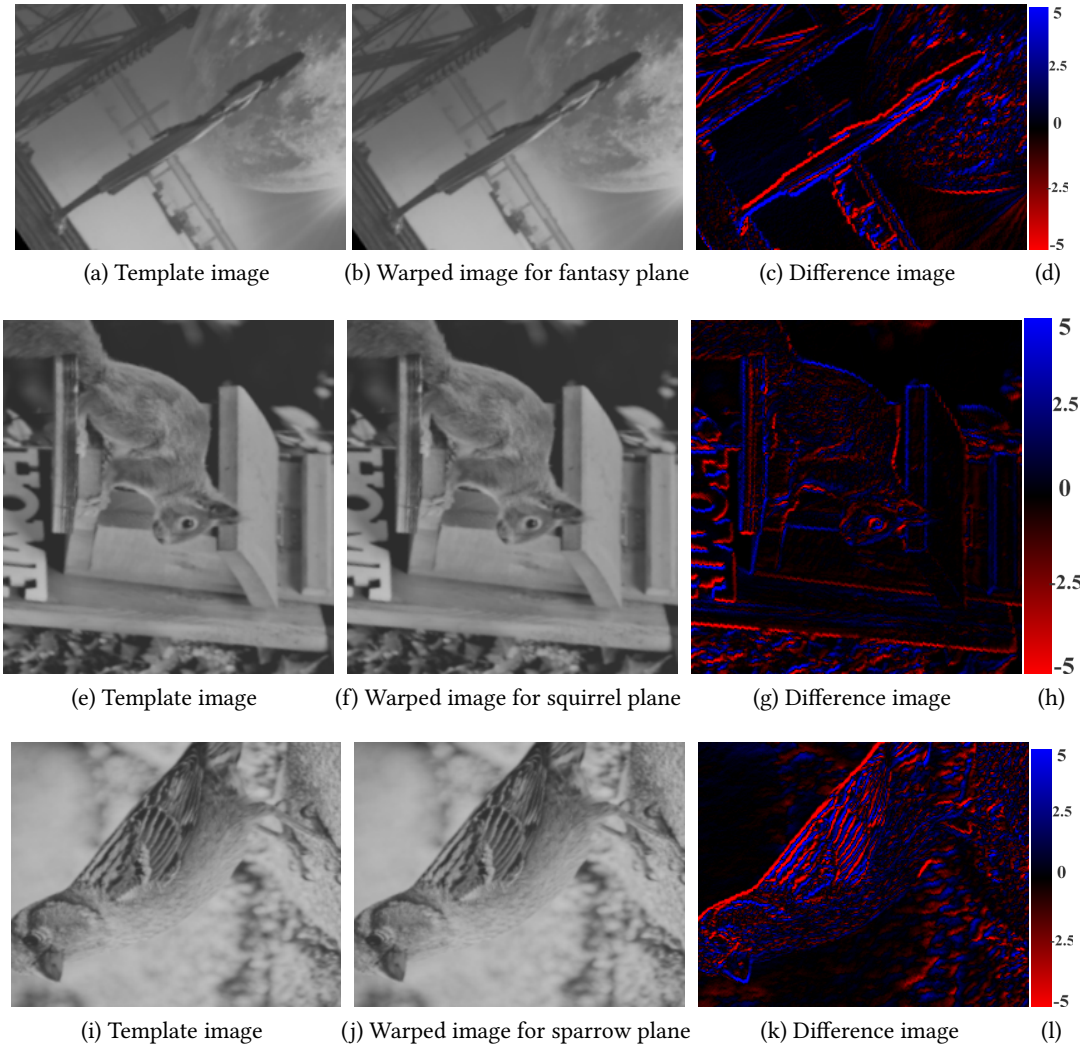


(a) Template image (b) Target image

Figure 4.15: Self-built images (2)

Patch	**PSNR(dB)**
Fantasy Plane	36.1911
Squirrel Plane	40.2400
Sparrow Plane	40.4210

Table 4.9: PSNR of different patches for self-built images (2)



(a) Template image (b) Warped image for fantasy plane (c) Difference image (d)

(e) Template image (f) Warped image for squirrel plane (g) Difference image (h)

(i) Template image (j) Warped image for sparrow plane (k) Difference image (l)

Figure 4.16: Result of self-built normal images (2)

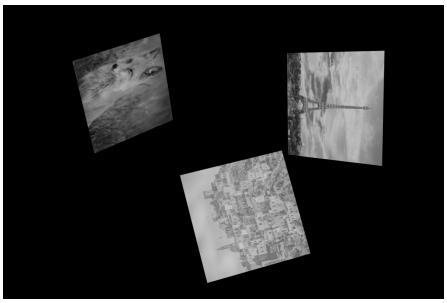
Patch	**RMSD (pixel)**	**Maximum(pixel)**
Fantasy Plane	0.291320	0.393618
Squirrel Plane	0.131122	0.154218
Sparrow Plane	0.189493	0.207162
All Patches	0.202340	0.393618

Table 4.10: Displacement of different patches for self-built images (2)

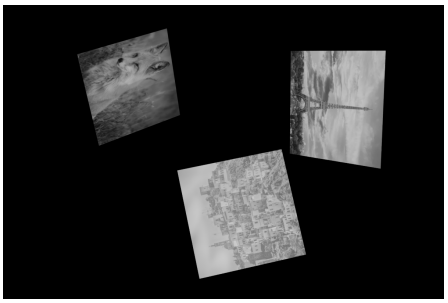
Parameter	Ground Truth			Estimated		
H_∞	$\begin{bmatrix} 9.52824171e-01 \\ 1.88784031e-01 \\ -4.01833405e-05 \end{bmatrix}$	$\begin{bmatrix} -1.61137624e-01 \\ 1.00060579e+00 \\ 6.27021960e-05 \end{bmatrix}$	$\begin{bmatrix} 2.06105992e+02 \\ -2.35871911e+02 \\ 9.92769988e-01 \end{bmatrix}$	$\begin{bmatrix} 9.51958568e-01 \\ 1.88447546e-01 \\ -3.99424923e-05 \end{bmatrix}$	$\begin{bmatrix} -1.61028691e-01 \\ 9.99411077e-01 \\ 6.26078118e-05 \end{bmatrix}$	$\begin{bmatrix} 2.05291642e+02 \\ -2.35358036e+02 \\ 9.90950379e-01 \end{bmatrix}$
$\boldsymbol{e}$		$\begin{bmatrix} -1.21556156e+03 \\ -4.60188500e+01 \\ -7.75948410e-02 \end{bmatrix}$			$\begin{bmatrix} -1.21004890e+03 \\ -4.60502141e+01 \\ -7.27863792e-02 \end{bmatrix}$	
$\vec{q}_1$		$\begin{bmatrix} -3.18191743e-05 \\ -3.30706122e-06 \\ 1.00547129e-01 \end{bmatrix}$			$\begin{bmatrix} -3.20962545e-05 \\ -4.33730691e-06 \\ 1.00806139e-01 \end{bmatrix}$	
$\vec{q}_2$		$\begin{bmatrix} 5.10113553e-05 \\ -1.23551078e-05 \\ 5.98174870e-02 \end{bmatrix}$			$\begin{bmatrix} 5.10028797e-05 \\ -1.23051622e-05 \\ 5.98152421e-02 \end{bmatrix}$	
$\vec{q}_3$		$\begin{bmatrix} 1.06495815e-05 \\ 4.19802260e-05 \\ 7.33070552e-02 \end{bmatrix}$			$\begin{bmatrix} 1.05603440e-05 \\ 4.18628465e-05 \\ 7.33696013e-02 \end{bmatrix}$	
r_{11}					1.01451717e+00	
r_{21}					1.17515309e-04	
r_{12}					9.86832299e-01	
r_{22}					-1.10771571e-04	
r_{12}					9.96963583e-01	
r_{22}					-1.98783474e-05	

Table 4.11: Ground truth and estimated parameter (2)

Finally, I show a result of a pair of unrectified stereo images. The self-built images are shown in Figure 4.17. The ground truth and estimated parameters are in Table 4.14. PSNR is in Table 4.12. The root mean squared displacement (RMSD) and maximum displacement of all patches are shown in Table 4.13. The warped images are shown in Figure 4.18.

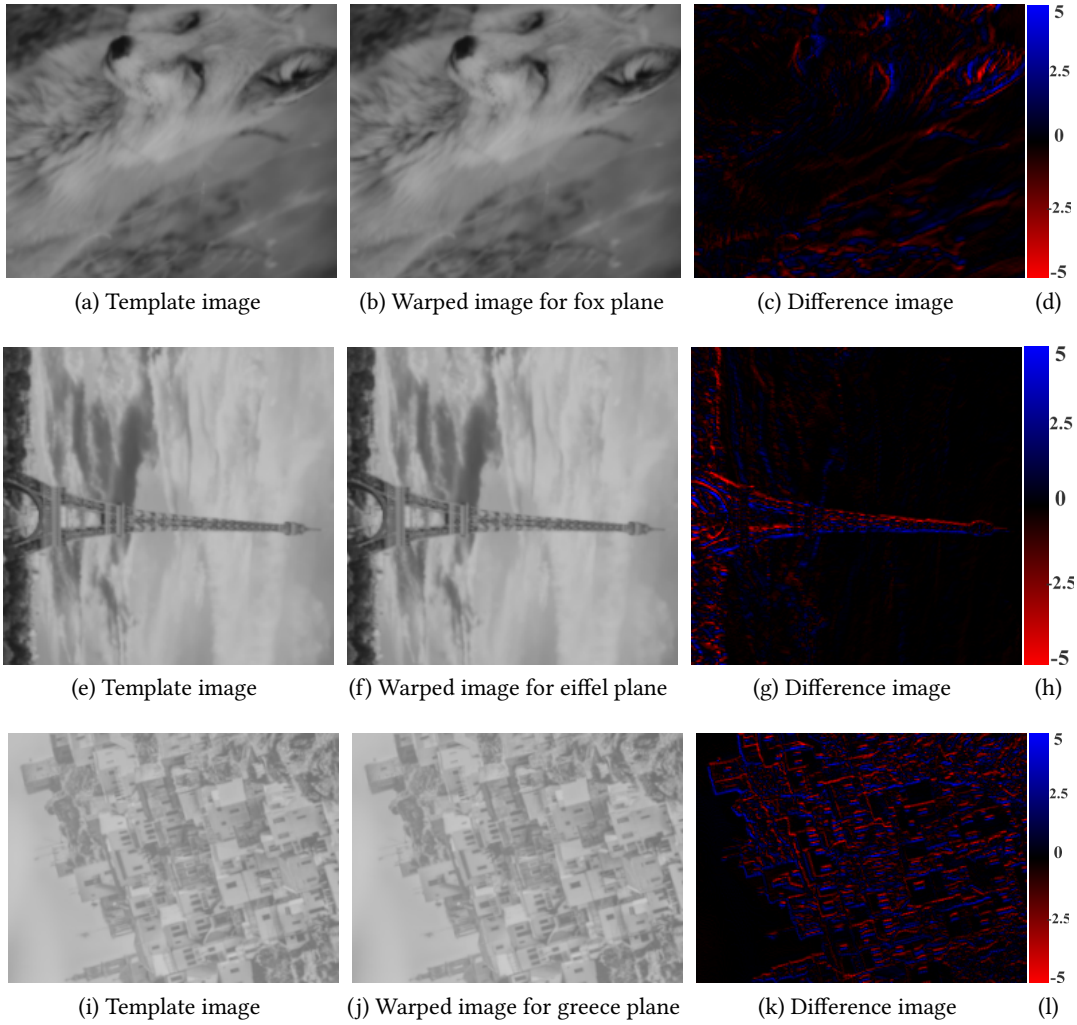


(a) Template image



(b) Target image

Figure 4.17: Self-built images (3)



(a) Template image (b) Warped image for fox plane (c) Difference image (d)

(e) Template image (f) Warped image for eiffel plane (g) Difference image (h)

(i) Template image (j) Warped image for greece plane (k) Difference image (l)

Figure 4.18: Result of self-built normal images (3)

Patch	**PSNR(dB)**
Fox Plane	46.1314
Eiffel Plane	44.7784
Greece Plane	32.3565

Table 4.12: PSNR of different patches for self-built images (3)

Patch	**RMSD (pixel)**	**Maximum(pixel)**
Fox Plane	0.126497	0.226566
Eiffel Plane	0.087420	0.162986
Greece Plane	0.136024	0.194004
All Patches	0.117115	0.226566

Table 4.13: Displacement of different patches for self-built images (3)

Parameter	**Ground Truth**	**Estimated**
H_∞	$\begin{bmatrix} 9.67030208e-01 & -3.13227134e-02 & 1.42036819e+02 \\ 1.54081860e-02 & 1.00157007e+00 & -1.94949553e+01 \\ -4.93452978e-05 & 5.50999444e-06 & 1.02462584e+00 \end{bmatrix}$	$\begin{bmatrix} 9.89508997e-01 & -3.14215411e-02 & 1.41579233e+02 \\ 1.61180210e-02 & 1.02097858e+00 & -2.02384459e+01 \\ -4.89309883e-05 & 6.50245995e-06 & 1.03992663e+00 \end{bmatrix}$
$\boldsymbol{e}$	$\begin{bmatrix} -1.19028099e+03 \\ -1.55712365e+00 \\ 4.98674731e-03 \end{bmatrix}$	$\begin{bmatrix} -1.19003604e+03 \\ -1.62088391e+00 \\ 3.23745088e-02 \end{bmatrix}$
$\vec{q}_1$	$\begin{bmatrix} -4.45934748e-05 \\ -2.22353073e-05 \\ 1.08137896e-01 \end{bmatrix}$	$\begin{bmatrix} -4.18794095e-05 \\ 2.10081204e-05 \\ 1.07554103e-01 \end{bmatrix}$
$\vec{q}_2$	$\begin{bmatrix} 4.89262638e-05 \\ -1.18500922e-05 \\ 5.73724445e-02 \end{bmatrix}$	$\begin{bmatrix} 5.09498850e-05 \\ -1.22376789e-05 \\ 5.80580159e-02 \end{bmatrix}$
$\vec{q}_3$	$\begin{bmatrix} -1.00128972e-05 \\ 1.09754268e-05 \\ 9.83050209e-02 \end{bmatrix}$	$\begin{bmatrix} -7.76330602e-06 \\ 1.09390874e-05 \\ 9.86232716e-02 \end{bmatrix}$
r_{11}		9.99853298e-01
r_{21}		-1.25598370e-06
r_{12}		9.94348907e-01
r_{22}		-3.45616975e-05
r_{12}		1.02688585e+00
r_{22}		1.64952502e-04

Table 4.14: Ground truth and estimated parameter (3)

4.3.4 Comparison and analysis

So far, the proposed algorithm is applied to the self-built images and Middlebury Stereo Vision Datasets. The specific test results obtained are listed in detail in Subsection 4.3.1, Subsection 4.3.2, Subsection 4.3.3, Subsection 3.2.4 and Subsection 3.2.5. For datasets with ground truth, we can analyze our algorithm by comparing the estimated result with the ground truth.

I summarize all the results of image pairs with ground truth in Table 4.15 and Table 4.16.

Image Pair	Patch	**RMSD (pixel)**	**Maximum(pixel)**	**PSNR(dB)**
Figure 4.10a	(1)	2.252596	3.125	37.5768
	(2)	0.647761	0.975361	40.4451
	(3)	0.184386	0.324421	45.9940
	(4)	0.536102	0.859401	42.4099
	(5)	0.954197	2.257174	47.7764
	(6)	0.895281	1.001767	38.3245
Figure 4.10b	(1)	0.203423	0.441505	46.9869
	(2)	1.123935	2.192497	41.8912
	(3)	1.523684	2.625	32.6315
	(4)	0.358562	0.483073	41.8892
	(5)	1.111534	1.700925	36.9338
	(6)	0.102031	0.290172	49.0576
Figure 4.10c	(1)	0.507921	0.883605	51.4393
	(2)	0.344584	0.689927	44.9664
	(3)	1.353465	3.046611	34.9695
	(4)	0.664646	1.188368	47.4049
	(5)	0.188721	0.554017	45.2209
Figure 4.11	Mercury Plane	0.044846	0.045849	49.1514
	Sheep Plane	0.003704	0.004460	66.9878
	Butterfly Plane	0.017901	0.018868	56.9686

Table 4.15: Result of rectified stereo testing images

From Table 4.15, we can see that the results of applying the proposed algorithm to self-built stereo images are significantly better than the Middlebury Stereo Vision datasets. The difference is that we have the 100 % right ground truth of self-build stereo images, but we only have a disparity map scaled by a factor of 8, which means that a value of 80 in disparity map presents that the corresponding pixel in template image is 10 pixels to the right of target image. In this case, the pixel value in disparity map is only integer, the final ground truth got is not 100 % accurate. Even so, our average error is only less than half or at most one pixels. What's more, the rectangular areas we manually selected in Middlebury Stereo Vision Datasets may

not all be the same plane, which will also cause inaccurate results.

Image Pair	Patch	**RMSD (pixel)**	**Maximum(pixel)**	**PSNR(dB)**
Figure 3.10	Deer Plane	0.186577	0.370456	44.9857
	Flower Plane	0.130046	0.162404	38.3061
	Cat Plane	0.369888	0.874363	41.5518
	All Patches	0.215594	0.874363	-
Figure 4.14	Brick-wall Plane	0.304703	0.325780	29.3573
	Voronoi Plane	0.208013	0.277421	40.4581
	Markov Plane	0.344370	0.397986	38.8231
	All Patches	0.288475	0.397986	-
Figure 4.16	Fantasy Plane	0.291320	0.393618	36.1911
	Squirrel Plane	0.131122	0.154218	40.2400
	Sparrow Plane	0.189493	0.207162	40.4210
	All Patches	0.202340	0.393618	-
Figure 4.18	Fox Plane	0.126497	0.226566	46.1314
	Eiffel Plane	0.087420	0.162986	44.7784
	Greece Plane	0.136024	0.194004	32.3565
	All Patches	0.117115	0.226566	-

Table 4.16: Result of unrectified testing images

I show this four unrectified self-built image pairs because they all have their own characteristics and can be used to represent most of the application scenarios. The angle between the cameras of Figure 3.10 is very small. Comparing to this, the angle between the cameras of Figure 4.18 is a little bit bigger. The Results of these are both acceptable. This proves that the pose of the cameras has little or no influence of the proposed algorithm.

The angles between cameras of Figure 3.10 and Figure 4.14 are both small. But the plane objects in the images are not the same type. Figure 3.10 has synthetic textures whereas the other has real images. And there is also a repeating structure in brick-wall plane in Figure 3.10. It's obviously that the result of repeating structure is not good as another. And the contains of the plane objects has small influence.

Figure 4.14 has standard orthogonal planes whereas the other three have arbitrary angle between them. More precisely, Figure 3.10 has near orthogonal planes. Figure 4.16 and Figure 4.18 have near parallel planes in contrast. But the results of the proposed algorithm are almost the same. We can conclude that the pose or position of target plane doesn't influence the result of the proposed algorithm.

5 Conclusion

In this master thesis, a new algorithm to improve image registration for a piecewise planar world has been presented, which is based on multiple homography estimation and Gauss-Newton method. Furthermore, for a more comprehensive and systematic evaluation of this algorithm, a new dataset has been created entirely computer generated in 3D using virtual engine software Blender. Finally, the algorithm was evaluated on some public dataset and the self-built dataset both.

We considered two cases in the evaluation, one where the relative orientation between the images is known and one where it has to be determined. In the first case, the evaluation has shown that the algorithm has a very good performance when H_∞ and $\boldsymbol{e}$ are known i.e. the algorithm is very suitable for rectified images.

The evaluation result of the normal unrectified images shows that the final algorithm has achieved the set goal: estimation of multiple homography of plane patches. Through the analysis of the evaluation results, we can conclude that the algorithm can be used in any pair of images, as long as there is the same plane in the images. In addition, a proper initialization has a significant impact on the results of the algorithm.

In general, the classical homography transformation can only be used for images registration with one large plane. We have successfully implemented the multiple homography to register the sub-patches of images, even if there is more than one plane in the image.

The final result of multiple homography matrices is linked by the global parameters and local parameters. The global parameters contain the information of fundamental matrix, which means that we can also get the fundamental matrix of images from the algorithm. Therefore, the algorithm can also be used to follow camera path.

Finally, a suitable, correct and credible dataset is successfully built in the thesis. It was used to evaluate the algorithm and uploaded as a public dataset for the future work. In the dataset, there is not only the built images and ground truth but also the source code and 3D environment, which can be used by anyone to build the new images for other research work.

5.1 Future Work

Aliasing Handling The evaluation of Figure 4.13 shows that the current implementation of the algorithm performs not very good even though the image pyramid is used to improve the accuracy, when there exist high frequency variations in the image. Some effort should be made to investigate this issue.

Robust Estimation Currently, the algorithm estimates the parameters from every pixel and is sensitive to noise. In order to overcome this problem, the image blurring method is used here. But after getting the estimation of blurred images, people should still apply the algorithm on original images with the result of blurred images, where the effect of noise reappeared. So an optimization of the algorithm to reduce the interference of noise should be done in the future work.

Better Initialization So far, the initialization of parameters is got manually for rectified images or from EXIF data of normal images. But when there are no EXIF data, it's impossible to get the initialization of normal images. So a new step to find an approximate initialization should be added to the algorithm as pre-processing step. And taking the feature points such as Harris corner as candidates for plane patch and estimating the homography matrix roughly are a possible way to get a sufficient initialization .

Post Processing The current way to warp the image is directly using the function warpPerspective in library OpenCV (Subsection 3.3.1), and the interpolation methods are only linear or nearest. So the warping result is not very accurate even though the estimation of parameters is very good. It will lead to erroe in the final result. Therefore, a better post processing or mapping function deserves further investigation to improve the final estimations.

Bibliography

[20a] "Peak Signal-to-Noise Ratio." In: *Wikipedia* (Apr. 2020).

[20b] "Sobel Operator." en. In: *Wikipedia* (June 2020).

[Ack84] F. Ackermann. "Digital Image Correlation: Performance and Potential Application in Photogrammetry." In: *The Photogrammetric Record* 11.64 (1984), pp. 429–439.

[BDK] Simon Baker, Ankur Datta, and Takeo Kanade. *Parameterizing Homographies*.

[Ber08] Paul Berner. *Technical Concepts Orientation, Rotation, Velocity and Acceleration, and the SRM*. Version 2.0. June 2008.

[BMH16] Daniel Barath, Jiri Matas, and Levente Hajder. "Multi-H: Efficient Recovery of Tangent Planes in Stereo Images." In: *Procedings of the British Machine Vision Conference 2016*. York, UK: British Machine Vision Association, 2016, pp. 13.1–13.13.

[Bro92] Lisa Gottesfeld Brown. "A Survey of Image Registration Techniques." In: *ACM Computing Surveys* 24.4 (Dec. 1992), pp. 325–376.

[Cho+09] Wojciech Chojnacki et al. "Multi-Projective Parameter Estimation for Sets of Homogeneous Matrices." In: *DICTA 2009 - Digital Image Computing: Techniques and Applications*. Jan. 2009, pp. 119–124.

[Cho+10] Wojciech Chojnacki et al. "Multiple Homography Estimation with Full Consistency Constraints." In: *2010 International Conference on Digital Image Computing: Techniques and Applications*. Dec. 2010, pp. 480–485.

[Cho+15] Wojciech Chojnacki et al. "Enforcing Consistency Constraints in Uncalibrated Multiple Homography Estimation Using Latent Variables." In: *Machine Vision and Applications* 26.2 (Apr. 2015), pp. 401–422.

[Col] Robert Collins. *Planar Homographies*. www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf.

[CS09] Pei Chen and David Suter. "Rank Constraints for Homographies over Two Views: Revisiting the Rank Four Constraint." en. In: *International Journal of Computer Vision* 81.2 (Feb. 2009), pp. 205–225.

[Deu11] Peter Deuflhard. “Least Squares Problems: Gauss-Newton Methods.” In: *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*. Ed. by Peter Deuflhard. Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer, 2011, pp. 173–231.

[Dub07] Elan Dubrofsky. “Homography Estimation.” In: (2007).

[Erd17] Bastian Erdnüß. “Das Baryzentrische Verhältnis Als Invariante Der Projektiven Geometrie.” In: *tm - Technisches Messen* 84.7-8 (Aug. 2017), pp. 479–492.

[Erd18] Bastian Erdnüß. “Measuring in Images with Projective Geometry.” In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1 (Sept. 2018), pp. 141–148.

[Gru85a] Armin Gruen. “Adaptive Least Squares Correlation: A Powerful Image Matching Technique.” In: *South African Journal of Photogrammetry, Remote Sensing and Cartography* Vol. 14 (Mar. 1985), pp. 175–187.

[Gru85b] A. W. Gruent. “Data Processing Methods for Amateur Photographs.” In: *The Photogrammetric Record* 11.65 (1985), pp. 567–579.

[Gru86] A W Gruen. *Adaptive Least Squares Correlation: A Powerful Image Matching techniqueAdapt Least Square Correlation*. 1986.

[HB03] P. Hellier and C. Barillot. “Coupling Dense and Landmark-Based Approaches for Nonrigid Registration.” In: *IEEE Transactions on Medical Imaging* 22.2 (Feb. 2003), pp. 217–227.

[HIG02] Heiko Hirschmüller, Peter R. Innocent, and Jon Garibaldi. *Real-Time Correlation-Based Stereo Vision with Reduced Border Errors*. 2002.

[Hir05] H. Hirschmuller. “Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. June 2005, 807–814 vol. 2.

[Hir08] Heiko Hirschmuller. “Stereo Processing by Semiglobal Matching and Mutual Information.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (Feb. 2008), pp. 328–341.

[Hir11] Heiko Hirschmüller. “Semi-Global Matching - Motivation, Developments and Applications.” In: *Photogrammetric Week 11*. Ed. by Dieter Fritsch. Stuttgart, Germany: Wichmann, Sept. 2011, pp. 173–184.

[HS09] Heiko Hirschmuller and Daniel Scharstein. "Evaluation of Stereo Matching Costs on Images with Radiometric Differences." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.9 (Sept. 2009), pp. 1582–1599.

[HZ04] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2004.

[JG12] Jyoti Joglekar and Shirish S. Gedam. "Area Based Image Matching Methods – A Survey." In: 2012.

[JI16] N. Jayanthi and S. Indu. "Comparison of Image Matching Techniques." In: *INTERNATIONAL JOURNAL OF LATEST TRENDS IN ENGINEERING AND TECHNOLOGY* 7.3 (2016).

[Kan98] Kenichi Kanatani. "Optimal Homography Computation with a Reliability Measure." In: *MVA*. 1998.

[KHD01] R. Kaucic, R. Hartley, and N. Dano. "Plane-Based Projective Reconstruction." In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. Vancouver, BC, Canada: IEEE Comput. Soc, 2001, pp. 420–427.

[KPS17] Ebrahim Karami, Siva Prasad, and Mohamed Shehata. "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images." In: *arXiv:1710.02726 [cs]* (Oct. 2017). arXiv: 1710.02726 [cs].

[Lóp+05] G. López-Nicolás et al. "Computing Homographies from Three Lines or Points in an Image Pair." en. In: *Image Analysis and Processing – ICIAP 2005*. Ed. by Fabio Roli and Sergio Vitulano. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 446–453.

[Low99] D.G. Lowe. "Object Recognition from Local Scale-Invariant Features." In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Kerkyra, Greece: IEEE, 1999, 1150–1157 vol.2.

[Lüt96] Helmut Lütkepohl. "The Handbook of Matrices." In: 1996.

[MIS] MISB. *Photogrammetry Metadata Set for Digital Motion Imagery*.

[MV92] Ezio Malis and Manuel Vargas. "Deeper Understanding of the Homography Decomposition for Vision-Based Control." en. In: *Bulletin of Sociological Methodology/Bulletin de Méthodologie Sociologique* 37.1 (Dec. 1992), pp. 55–57.

[Nak+96] Y. Nakamura et al. "Occlusion Detectable Stereo-Occlusion Patterns in Camera Matrix." In: *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. June 1996, pp. 371–378.

[ope] opencv dev team. *OpenCV 2.4.13.7 Documentation.* https://docs.opencv.org/2.4/index.html.

[R E01] Philip R. Evans. "Rotations and Rotation Matrices." In: *Biological Crystallography* (June 2001).

[Rub+11] Ethan Rublee et al. "ORB: An Efficient Alternative to SIFT or SURF." In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571.

[SA96] Amnon Shashua and Shai Avidan. "The Rank 4 Constraint in Multiple (≥3) View Geometry." en. In: *Computer Vision — ECCV '96*. Ed. by Bernard Buxton and Roberto Cipolla. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 196–206.

[Sch+17] Thomas Schops et al. "A Multi-View Stereo Benchmark with High-Resolution Images and Multi-Camera Videos." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 2538–2547.

[SCv15] Zygmunt L. Szpak, Wojciech Chojnacki, and Anton van den Hengel. "Robust Multiple Homography Estimation: An Ill-Solved Problem." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2132–2141.

[SSZ01] D. Scharstein, R. Szeliski, and R. Zabih. "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms." In: *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*. Kauai, HI, USA: IEEE Comput. Soc, 2001, pp. 131–140.

[Str20] Didier Stricker. *2D Projective Transformations (Homographies)*. 2020.

[Sze] Richard Szeliski. *Computer Vision: Algorithms and Applications*.

[Wan+19] Yingqian Wang et al. "Flickr1024: A Large-Scale Dataset for Stereo Image Super-Resolution." In: *arXiv:1903.06332 [cs]* (Aug. 2019). arXiv: 1903.06332 [cs].

[Won+11] Hoi Wong et al. "Dynamic and Hierarchical Multi-Structure Geometric Model Fitting." In: *Proceedings of the IEEE International Conference on Computer Vision*. Nov. 2011, pp. 1044–1051.

[Yeh11] H. C. Yeh. "Hardware Components for Real-Time Stereo Matching: Acceleration of 3D HD TV with FPGAs." In: (2011).

[Yue+11] Wang Yue-Zong et al. "Image Match Algorithm through Correlation Coefficient Computation in Micro Stereovision." In: *2011 IEEE International Conference on Mechatronics and Automation*. Aug. 2011, pp. 2333–2337.

[ZF03] Barbara Zitová and Jan Flusser. “Image Registration Methods: A Survey.” In: *Image and Vision Computing* 21 (Oct. 2003), pp. 977–1000.

[ZI02] L. Zeinik-Manor and M. Irani. “Multiview Constraints on Homographies.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.2 (Feb. 2002), pp. 214–223.

List of Tables

List of Figures

Theory List

Listings