Efficient Optimization for Robust Bundle Adjustment

handed in
MASTER'S THESIS

Master of Science Zhongnan Qu

born on the 05.05.1992
living in:
Helene-Mayer-Ring 7A/0414
80809 Munich
Tel.: +4915224192752

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	Prof. Dr. Dongheui Lee, Shile Li, Prof. Dr. Daniel Cremers, Dr. Tao Wu, Nikolaus Demmel, Emanuel Laude
Start:	01.10.2017
Intermediate Report:	31.12.2017
Delivery:	31.03.2018

In your final hardback copy, replace this page with the signed exercise sheet.

Before modifying this document, READ THE INSTRUCTIONS AND GUIDELINES!

Abstract

Recently, bundle adjustment has been the key component in structure from motion problem. Bundle adjustment jointly optimizes the camera parameters and feature points parameters according to an objective function of reprojection errors. However, some widely used methods in bundle adjustment, such as, Levenberg-Marquardt step solved by Cholesky decomposition, truncated Newton's method, still face some challenges in robustness, accuracy, and efficiency. In this thesis, we provide a novel damped inexact Newton's method (DINM), which combines the advantages from both inexact Newton's method and truncated Newton's method. Furthermore, we introduce the adaptiveness, local parameterization, and reweighted potential function in our DINM algorithm. After testing on different types of datasets, the results show that in comparison with the baseline implementations, our algorithms not only obtain a faster and deeper reduction in objective function curves and gradient norm curves, a result more closed to the ground truth, but also own a higher robustness.

Contents

1 Introduction **5**
1.1 Related Work . . . 7
1.2 Problem Statement . . . 8
1.3 Outline . . . 10

2 Relevant Foundation **11**
2.1 Bundle Adjustment . . . 11
2.1.1 Geometric Transformation . . . 11
2.1.2 Overview of Bundle Adjustment . . . 15
2.2 Numerical Optimization Algorithm . . . 17
2.2.1 Line Search and Trust Region . . . 18
2.2.2 Gradient Descent Algorithm . . . 19
2.2.3 Newton's Method . . . 19
2.2.4 Quasi-Newton's Method . . . 20
2.2.5 Gauss Newton Algorithm . . . 20
2.2.6 Levenberg-Marquardt Algorithm . . . 21
2.3 Dataset . . . 22

3 Damped Inexact Newton's Method in Bundle Adjustment **25**
3.1 Matrix Form in Bundle Adjustment . . . 25
3.1.1 Reprojection Error Vector . . . 25
3.1.2 Partial Derivative . . . 25
3.1.3 Jacobian Matrix . . . 30
3.1.4 Verification of Jacobian Matrix . . . 30
3.1.5 Gradient Vector . . . 32
3.1.6 Hessian Matrix . . . 32
3.2 Baseline Implementations of Bundle Adjustment . . . 35
3.2.1 Schur Complement . . . 37
3.2.2 Cholesky Decomposition . . . 39
3.2.3 Baseline Implementations . . . 42
3.3 Damped Inexact Newton's Method . . . 42
3.3.1 Conjugate Gradient . . . 44
3.3.2 Preconditioned Conjugate Gradient . . . 46

3.3.3 Inexact Newton's Method 46
3.3.4 Truncated Newton's Method 54
3.3.5 Inexact Newton's Method with Damping 57

4 Further Improvements **75**
4.1 Adaptive DINM 75
4.2 Local Parameterization 78
4.3 Iteratively Reweighted Least Squares (IRLS) 82
4.3.1 Reweighted Potential Function 82
4.3.2 Iteratively Reweighted Least Squares with Local Parameterization 85

5 Experiments and Evaluation **89**
5.1 Synthetic Bundle Adjustment Dataset 89
5.1.1 Feature Points 89
5.1.2 Camera and Image Frames 89
5.1.3 Noises and Outliers 92
5.1.4 Initial Value 92
5.2 Experiment implementations in Matlab 92
5.3 Evaluation 93
5.3.1 Reprojection Error Plots 94
5.3.2 Performance Comparison 96

6 Discussion **109**

7 Conclusion **111**

A Appendix **113**
A.1 Matlab Code 113
A.2 Results Data 113

List of Figures **115**

Bibliography **119**

Chapter 1

Introduction

With the development of camera technique, a sequence of high resolution photos can be easily collected in more and more scenes. The camera captures a 2D image frame through the vision field of the lens or lens group, which is actually a projection process. In this process, the instantaneous real 3D object or 3D scene in the vision field is projected onto a 2D plane proportionally [wikf]. The location and the boundary of this 2D plane are defined by the camera parameters.

Furthermore, if there exists relative motion between the camera and the real 3D objects, the 3D objects are projected onto the 2D image frames from different view directions. With such a 2D image sequence, we want to estimate the 3D structure of the original objects, which is called Structure from Motion (SfM) [The]. SfM is a photogrammetric range imaging technique for reconstructing 3D structures from 2D projected images collections based on the moving camera or moving objects. SfM is a important technique in many applications of computer vision, such as, 3D geometry reconstruction, simultaneous localization and mapping (SLAM), etc. [wikh] In the last decades, researchers have developed lots of methods to solve SfM problems, such as, maximum likelihood estimation (MLE) [GGS+07], Kalman filter [HD07], particle filter [SEG+05], hidden markov model (HMM) [HB14], etc. SfM presents a problem, whose input is 2D image frames sequence with the recognized (observed) feature points of the 3D objects, and the output is the reconstructed 3D positions of these feature points.

Bundle adjustment is a key component in the most recent SfM problem. Based on the projected images collections from different viewpoints, bundle adjustment jointly refine the 3D coordinates of the scene geometry (feature points), the transformation parameters from the relative motion between the camera and the scene, and the optical characteristics of the camera, according to an optimal criterion related to the reprojection errors of feature points. Bundle adjustment is almost always used as the last step of every feature-based 3D reconstruction algorithm [wika].

Bundle adjustment has a high flexibility. It gracefully handles a very wide variety of different 3D feature and camera types (points, lines, curves, surfaces, exotic cameras), scene types (including dynamic and articulated models, scene constraints),

information sources (2D features, intensities, 3D information, priors) and error models [TMHF99].
The process of bundle adjustment is considered that there exist bundles of light rays from each feature points to each camera center, and all parameters of these feature points and cameras are adjusted together simultaneously and iteratively to obtain the optimal bundles [TMHF99]. This process is plotted in Figure 1.1.

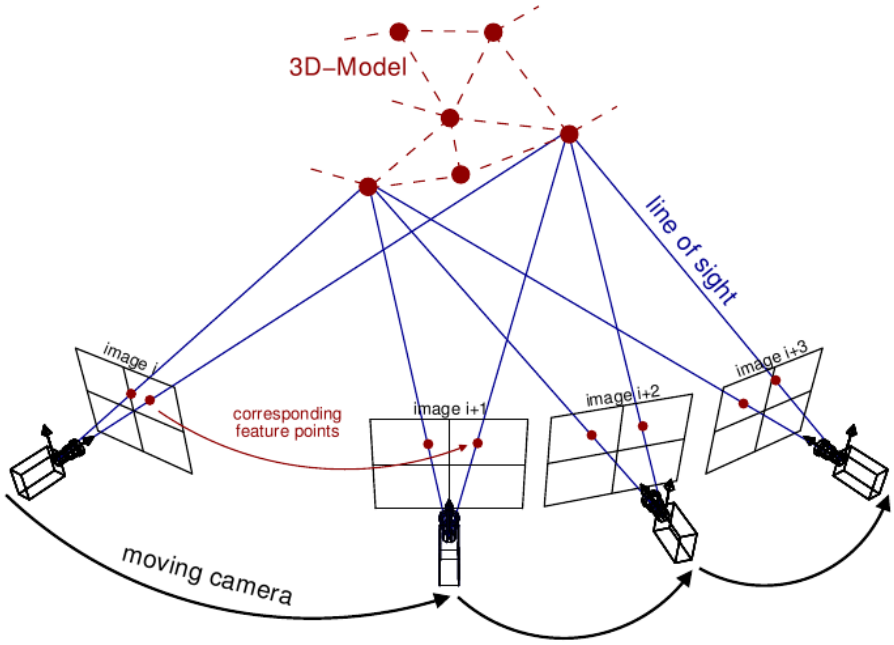

Figure 1.1: Example of bundle adjustment [The].

Due to the highly developed computer technology, bundle adjustment is solved by numerical optimization methods. Optimization algorithm resolve the problem of finding variable values which optimize the defined objective function (cost function) of the system with or without constraints. The process of identifying objective, variables, and constraints for a given problem is known as modeling [NW06]. After constructing of an appropriate model, the optimal value is approached iteratively by numerical step with the help of computer. There is no universal optimization algorithm but rather a collection of algorithms, each of which is tailored to a particular type of optimization problem [NW06]. In general, the objective function in bundle adjustment consists of the reprojection errors in square form.
Reprojection errors in bundle adjustment come from all deviations between the observed feature points coordinates in the projected images and the reprojected feature points coordinates for each feature point in each image (if exists). Since the relative motion continues along the images collection, each image frame is of different camera parameters and not all feature points can appear in each image frame.

1.1 Related Work

Bundle adjustment was originally conceived in the field of photogrammetry during the 1950s and has increasingly been used by computer vision researchers until recent years [wika]. In the long time, some common misconceptions have limited the development of bundle adjustment. For example, many researchers do not deliberate the special structure of bundle adjustment, and solve the bundle adjustment in a general optimization routine of linear algebra, which leads to a extremely slow optimization than expected; some researchers have misunderstood the accuracy of the reconstructed feature points since they treat the uncertainty of camera parameters and feature points separately. [TMHF99] In 1999, B. Triggs et al have clarified these misconceptions, and provided a systematical summarization about bundle adjustment, including the particular model, the parameterization, and some new optimization strategies in bundle adjustment. They also analyze the accuracy and the efficiency of several implementations [TMHF99]. Their great efforts promotes the development of bundle adjustment in brand-new perspectives.

K. Konolige and M. Agrawal propose a framework for visual imagery in [KA08]. They match visual frames with large numbers of point features, but only keep relative frame pose information. This skeleton is a reduced nonlinear system that is a faithful approximation of the larger system and can be used to solve large loop closures quickly. [KA08]

S. Agarwal1 et al have proposed a new bundle adjustment implementation for the large-scale datasets in 2010. They design a new truncated Newton's method for solving large-scale bundle adjustment problems with tens of thousands of images. They introduce conjugate gradients for calculating the Newton step with some preconditioners instead of decomposition. They evaluate the performance of six different bundle adjustment algorithms, and show that their truncated Newton method with relatively simple preconditioners yields an advanced performance for large-scale bundle adjustment [ASSS10].

C. Wu et al exploit hardware parallelism for efficiently solving large-scale 3D reconstruction problems. They use multicore CPUs as well as multicore GPUs. Their results indicate that overcoming the memory and bandwidth limitations not only leads to more space efficient algorithms, but also to surprising savings in runtime [WACS11].

Recently, researchers also introduce some new models in bundle adjustment. The most previous methods for Bundle Adjustment are either centralized or operate incrementally. K. N. Ramamurthy et al address bundle adjustment with a new distributed algorithm using alternating direction method of multipliers (ADMM). They analyze convergence, numerical performance, accuracy, and scalability of the distributed implementation. The runtime of their implementation scales linearly with the number of observed points [RLA+17].

Besides, J. Engel et al also develop a Direct Sparse Odometry (DSO) based on a novel, highly accurate sparse and direct structure and motion formulation. It com-

bines a fully direct probabilistic model (minimizing a photometric error) with consistent, joint optimization of all model parameters, including geometry-represented as inverse depth in a reference frame-and camera motion. The proposed model integrates a full photometric calibration, accounting for exposure time, lens vignetting, and non-linear response functions. The experiments show that their approach significantly outperforms state-of-the-art methods both in terms of tracking accuracy and robustness [EKC18].

1.2 Problem Statement

Bundle adjustment problem requires a dataset with the observed feature points in each frame, and initial values for all arguments (3D feature points positions and cameras' parameters) as inputs; and bundle adjustment outputs the optimal arguments minimizing a chosen objective function, which is generally related to reprojection errors. Bundle adjustment actually finds a proper optimization method of this above process.
The two state-of-the-art strategies in bundle adjustment is listed below, which are presented in [TMHF99] and in [ASSS10] respectively. These both implementations are also treated as the baseline implementations in this thesis.

- LMA-CD: in [TMHF99], they propose a bundle adjustment implementation with exact **Levenberg-Marquardt Algorithm (LMA)** step in each iteration solved by dense **Cholesky decomposition**. This implementation integrates the special structure of Jacobian matrix and Hessian matrix. The details are discussed in Section 3.2.2. One of its typical variants presenting a good performance is selected as the baseline implementation: couple with Schur complement of $\boldsymbol{C}$. $\boldsymbol{C}$ is a sub-matrix of Hessian matrix, seeing Section 3.2.1.

- BAL: in [ASSS10], they test six different implementations for **bundle adjustment in the large**. Among those implementations, their proposed new truncated Newton's method coupled with Schur complement of $\boldsymbol{C}$ performs a better result than others. This implementation solves an inexact LMA step by conjugate gradients with preconditioner of $\boldsymbol{B}$. $\boldsymbol{B}$ is a sub-matrix of Hessian matrix, seeing Section 3.2.1. Besides, they use the diagonal matrix of Hessian matrix as the damping matrix in LMA, seeing Section 2.2.6.

Even if lots of researchers have proposed many strategies of bundle adjustment for different scenes, bundle adjustment is still far from a solved problem. In general, bundle adjustment is a highly non-linear, non-convex optimization problem with curse of dimensionality. Besides, it also faces bad condition, near singular problem in matrix operation during optimization. All above problems lead to that even some popular implementations still have shortages.

"LMA-CD" is hard to beat in small-scale and/or sparse dataset [ASSS10], i.e. an exact LMA step solved by Cholesky decomposition demonstrates a great performance in both accuracy and efficiency. Here, the dataset is sparse means there are only a small part of feature points appearing in each image frame. However, Cholesky decomposition is only suitable for small matrix. When facing a dense or large-scale dataset, "LMA-CD" always consumes extremely large computation time. Besides, "LMA-CD" is not robust enough. The matrix in bundle adjustment is always bad conditioned or near singular, so that even for small-scale dataset, the decomposition costs significant time or gets a very inexact solution in some optimization step, and sometimes it even yields a solution with NaN or Inf. The reason of this situation come from the following sides, some parameters degenerate after several optimization steps; some parameters, such as rotation angles, are much more sensitive than others; after several steps, the Hessian matrix is near singular and bad conditioned. "BAL" is much more efficient in both memory and time consumption than "LMA-CD" when facing a large-scale and/or dense dataset. In addition, the decomposition solver is replace by preconditioned conjugate gradients, which can still compute an acceptable solution even if the matrix of equation system is bad conditioned or near singular. However, the selection of preconditioner in "BAL" is confused. Besides, in testing, we find that its accuracy can not maintain the same level for different sorts of datasets. Especially, when tested on some synthetic datasets, its performance is extremely bad, seeing Section 5.3.

Thus, in this thesis, we are desired to design a new implementation of bundle adjustment with higher performance. The higher performance is defined in the following sides, which are also the general criteria of judging the performance of numerical optimization methods.

Accuracy: the criterion of accuracy comes from two parts, the order of magnitude of the objective function and the order of magnitude of gradient norm (of the objective function). How small the gradient norm reduces to after the optimization indicates how close this solution and the objective function are away from the local optimal. The order of the objective function indicates how close this solution is away from a global optimal. In this thesis, we only focus on seeking a local optimal with a given initial value. Thus, the order of the gradient norm is more important to us. Besides, how closed the optimization trajectory approaches the ground truth is also a metric to determine the accuracy.

Efficiency: the efficiency of algorithm also comes from two parts, memory demanding and time demanding, which are dependent on how much memory and how much runtime are spent during the optimization respectively.

Robustness: the robustness of implementations measure the effectiveness of the algorithm facing different types of datasets. Besides, it also shows the effectiveness when there exist some errors in the system.

In this thesis, we introduce a novel damped inexact Newton's method (DINM), which combines the advantages of truncated Newton's method and inexact Newton's method. Then, based on this DINM algorithm, we further implement some

improvements, e.g. adaptiveness, local parameterization, reweighted objective function, and propose three advanced algorithms. After testing the implementations on real datasets provided by [ASSS] and our synthetic datasets, the results show that our algorithms have a huge improvement in accuracy, efficiency, and robustness, in comparing with two baseline implementations.

1.3 Outline

Here, we have an outline of the following thesis. In Chapter 2, some basic knowledge about bundle adjustment and some general numerical optimization methods are introduced; in Chapter 3, firstly, the baseline implementations of bundle adjustment is presented; then, we demonstrate two popular Newton's methods with our improvements; finally, we combine the advantages from both methods and propose a damped inexact Newton's method in bundle adjustment; in Chapter 4, we introduce the adaptiveness, the local parameterization, and the reweighted potential function in damped inexact Newton's method to further improve the performance; the comparing results between our implementations and baseline implementations are presented in Chapter 5 on real datasets and our synthetic datasets; in Chapter 6, we analyze and discuss the evaluations presented in Chapter 5; at last, we make a summary of the whole thesis in Chapter 7.

Chapter 2

Relevant Foundation

In this chapter, we discuss some basic knowledge in bundle adjustment, e.g. geometric transformation, objective function. Besides, we also briefly present some widely used numerical optimization algorithms.

2.1 Bundle Adjustment

2.1.1 Geometric Transformation

In bundle adjustment implementations, the first step is to study the geometric relations between a 3D scene and the observed 2D projections, i.e. the transformation from the 3D point coordinates in the real space to its 2D projection on the camera image. This geometric transformation consists of three steps, rigid-body motion of camera system, perspective projection, and radial distortion.

Robot Kinematics

Robot kinematics study the relationship between the dimensions and connectivity of kinematic chains and the position, velocity and acceleration of each of the links in the robotic system. The non-linear kinematics equations map the joint parameters to the configuration of the robot system. These non-linear kinematics equations are used in the first step of the geometric transformation, i.e. rigid-body motion. In this step, the point coordinates in the original fixed inertial system are transformed into its coordinates in the moved camera system. The coordinate transformation between both systems is composed of two types of rigid-body motion, rotation and translation.
The original fixed inertial system is named by O, and the camera system is named by C. The original points in both coordinate system are also expressed as O and C respectively. The vector expression consists of the start point, the end point, and the coordinate system in which its form is expressed, e.g. ${}_C\boldsymbol{r}_{CP}$ presents the vector from origin C to point P demonstrated in the coordinate system C. The coordinate

transformation between system O and system C can be presented as,

$$\begin{pmatrix} {}_C\boldsymbol{r}_{CP} \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{CO} & {}_C\boldsymbol{r}_{CO} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} {}_O\boldsymbol{r}_{OP} \\ 1 \end{pmatrix} \ . \tag{2.1}$$

In (2.1), A_{CO} presents the transformation of the rotation motion. The rotation matrix, A_{CO}, has several features,

$$det(\boldsymbol{A}_{CO}) = 1 \ , \tag{2.2}$$

$$\boldsymbol{A}_{OC} = \boldsymbol{A}_{CO}^{-1} = \boldsymbol{A}_{OC}^{T} \ , \tag{2.3}$$

$$\boldsymbol{A}_{CO} = \begin{pmatrix} {}_C\boldsymbol{n}_{O\mathrm{x}} & {}_C\boldsymbol{n}_{O\mathrm{y}} & {}_C\boldsymbol{n}_{O\mathrm{z}} \end{pmatrix} \ . \tag{2.4}$$

$\boldsymbol{n}_{O\mathrm{x}}$, $\boldsymbol{n}_{O\mathrm{y}}$, and $\boldsymbol{n}_{O\mathrm{z}}$ present three unit basis vectors in O system. The 3D coordinates transformation between system O and system C is demonstrated in Figure 2.1.

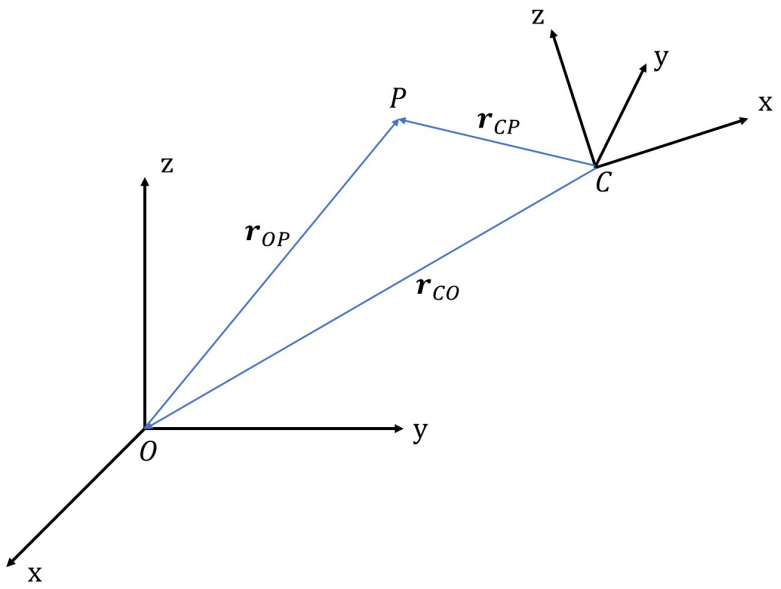

Figure 2.1: 3D coordinates (point P) transformation between coordinate system O and coordinate system C.

There are lots of rotation parameter forms describing the rigid-body rotation transformation, e.g. Euler angles, Tait-Bryan angles [Rix17]. In general, each form needs three independent parameters. In computer vision, Rodrigues' Formula is widely used to demonstrate the rotation matrix. Rodrigues' Formula is founded based on Theorem of Euler on finite rotations [Rix17],

Theorem 2.1.1 (Theorem of Euler) *if a rigid body undergoes a motion leaving fixed one of its points, then a set of points of the body, lying on a line that passes through that point, remains fixed as well.*

Through Theorem 2.1.1, it is clear that an arbitrary pure rigid-body rotation (one point fixed) can be defined by one rotation through a specific angle around an instant rotation axis, a line of fixed points. Therefore, Rodrigues' Formula uses a 3D rotation vector $\boldsymbol{\phi}$ to define a rotation transformation. Its magnitude, $\|\boldsymbol{\phi}\|$, defines the rotation angle; and its unit orientation vector, $\boldsymbol{\phi}/\|\boldsymbol{\phi}\|$, defines the rotation axis. Obviously, this unit orientation vector has the same form in both system.

$$\boldsymbol{n}_{\boldsymbol{\phi}} = \frac{\boldsymbol{\phi}}{\|\boldsymbol{\phi}\|} = {}_C\boldsymbol{n}_{\boldsymbol{\phi}} = {}_O\boldsymbol{n}_{\boldsymbol{\phi}} = \begin{pmatrix} n_{\phi,\mathrm{x}} \\ n_{\phi,\mathrm{y}} \\ n_{\phi,\mathrm{z}} \end{pmatrix} \ . \tag{2.5}$$

Camera system C can be rotated to original fixed system O around axis $\boldsymbol{n}_{\boldsymbol{\phi}}$ with the angle of $\|\boldsymbol{\phi}\|$; reversely, system O can be rotated to system C around axis $-\boldsymbol{n}_{\boldsymbol{\phi}}$ with the angle of $\|\boldsymbol{\phi}\|$. The rotation satisfies the right-hand rule [wikg]. Since there are only two systems (O and C) in our bundle adjustment, we use R as rotation matrix to replace the previous rotation matrix A_{CO}. Rodrigues' Formula is written as,

$$\boldsymbol{R} = (\mathbf{I} - \boldsymbol{n}_{\boldsymbol{\phi}}\boldsymbol{n}_{\boldsymbol{\phi}}^T)\cos(\|\boldsymbol{\phi}\|) + \widehat{\boldsymbol{n}}_{\boldsymbol{\phi}}\sin(\|\boldsymbol{\phi}\|) + \boldsymbol{n}_{\boldsymbol{\phi}}\boldsymbol{n}_{\boldsymbol{\phi}}^T \ , \tag{2.6}$$

with the cross product matrix,

$$\widehat{\boldsymbol{n}}_{\boldsymbol{\phi}} = \boldsymbol{n}_{\boldsymbol{\phi}}\times = \begin{pmatrix} 0 & -n_{\phi,\mathrm{z}} & n_{\phi,\mathrm{y}} \\ n_{\phi,\mathrm{z}} & 0 & -n_{\phi,\mathrm{x}} \\ -n_{\phi,\mathrm{y}} & n_{\phi,\mathrm{x}} & 0 \end{pmatrix} \ . \tag{2.7}$$

The vector, ${}_C\boldsymbol{r}_{CO}$, presents the transformation of the translation motion. In the thesis, we use $\boldsymbol{t}$ as translation vector instead of ${}_C\boldsymbol{r}_{CO}$, and use $\boldsymbol{x_i}$ to present the 3D coordinates of the i-th feature point, P_i.

$${}_C\boldsymbol{x}_i = {}_C\boldsymbol{r}_{CP_i} \ , \tag{2.8}$$

$${}_O\boldsymbol{x}_i = {}_O\boldsymbol{r}_{OP_i} \ , \tag{2.9}$$

$$\begin{pmatrix} {}_C\boldsymbol{x}_i \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} {}_O\boldsymbol{x}_i \\ 1 \end{pmatrix} \ . \tag{2.10}$$

More specific, (2.10) is written as,

$${}_C\boldsymbol{x}_i = \begin{pmatrix} {}_C x_{i,\mathrm{x}} \\ {}_C x_{i,\mathrm{y}} \\ {}_C x_{i,\mathrm{z}} \end{pmatrix} = \boldsymbol{R} \begin{pmatrix} {}_O x_{i,\mathrm{x}} \\ {}_O x_{i,\mathrm{y}} \\ {}_O x_{i,\mathrm{z}} \end{pmatrix} + \begin{pmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{pmatrix} \ . \tag{2.11}$$

Therefore, with (2.10), the coordinates of point P in system O are transformed to its coordinates in system C. Besides, there is also an inverse transformation of (2.10),

$$\begin{pmatrix} {}_O\boldsymbol{x}_i \\ 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}^T & -\boldsymbol{R}^T\boldsymbol{t} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} {}_C\boldsymbol{x}_i \\ 1 \end{pmatrix} \ . \tag{2.12}$$

Perspective Projection

In the second step of geometric transformation, the 3D coordinates in camera system C are projected onto a 2D image frame. In the thesis, the direction of $\boldsymbol{n}_{C\mathrm{z}}$ is defined as the optical axis of the lens, which is perpendicular to the lens plane and is directed to the inside of the camera [per].
In the thesis, we only consider the pinhole camera model, i.e. only one convex lens is used. The convex lens forms a inverted real image on the film. The actual image from convex lens is located on the plane $(0, 0, f)$. Here, f is the focal length of the convex lens. However, with other optics and elements in the camera, the captured image is converted to an upright projection just like located on the plane $(0, 0, -f)$. In perspective projection, the x and y coordinates of ${}_{\boldsymbol{C}}\boldsymbol{x_i}$ are scaled based on the ratio of z coordinate and focal length. In this case, x-axis and y-axis of the image coincide with x-axis and y-axis of the camera system C. The 2D coordinates of perspective projection of the i-th feature point is written as,

$$\boldsymbol{c}_i = \begin{pmatrix} c_{i,\mathrm{x}} \\ c_{i,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} -f \times \frac{{}_C x_{i,\mathrm{x}}}{{}_C x_{i,\mathrm{z}}} \\ -f \times \frac{{}_C x_{i,\mathrm{y}}}{{}_C x_{i,\mathrm{z}}} \end{pmatrix} . \tag{2.13}$$

An example of real 3D scene and its 2D projection of a camera system is drawn in Figure 2.2.

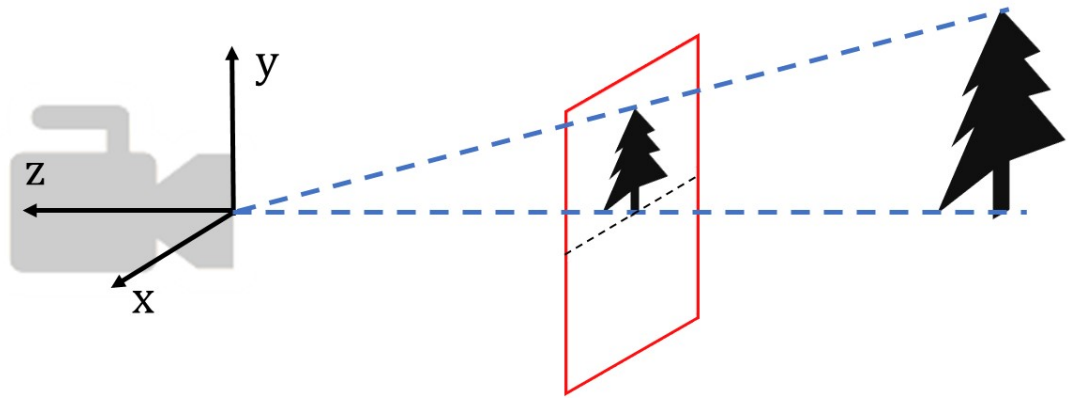

Figure 2.2: Perspective projection in a camera system. The red plane presents the projection plane located at $(0, 0, -f)$. The real 3D object (tree) is proportional projected onto this plane.

Radial Distortion

In geometric optics, distortion is a deviation from rectilinear projection, a projection in which straight lines in a scene remain straight in an image. It is a form of optical aberration. Due to the symmetry of a photographic lens, the most commonly

encountered distortions are radially symmetric, or approximately [wikc]. In order to make the image more similar to its real shape, the radial distortion adjustment is added into camera imaging system. Therefore, the third step of geometric transformation is distortion calibration. In this case, the coordinates from perspective projection is calibrated with a simple 2-parameter model.

$$\boldsymbol{c}_i^{'} = \begin{pmatrix} c_{i,\mathrm{x}}^{'} \\ c_{i,\mathrm{y}}^{'} \end{pmatrix} = (1 + k_1 \times \|\boldsymbol{c}_i\|^2 + k_2 \times \|\boldsymbol{c}_i\|^4) \times \begin{pmatrix} c_{i,\mathrm{x}} \\ c_{i,\mathrm{y}} \end{pmatrix} \quad . \tag{2.14}$$

$\boldsymbol{c}_i^{'}$ is final 2D reprojected coordinates of the i-th feature point.

2.1.2 Overview of Bundle Adjustment

Like we have talked in Chpater 1, bundle adjustment defines an optimal reconstruction problem on 3D feature points coordinates and the camera parameters, e.g. camera pose, focal length, and radial distortion under certain assumptions: if the observed error of image feature is zero-mean Gaussian distribution, bundle adjustment is the Maximum Likelihood Estimator (MLE) [wika].
Considering consistent presentation in the whole thesis, firstly, the basic notations in bundle adjustment are defined as below. Some notations have already been used in the previous sections, which still keep the same.

- $\{\boldsymbol{x}_i\}_{i=1}^{\mathrm{m}} \subset \mathbb{R}^3$: 3D coordinates of feature points.
- m: the total number of feature points.
- $\{(\boldsymbol{\phi}_j, \boldsymbol{t}_j)\}_{j=1}^{\mathrm{n}} \subset \mathrm{SE}(3)$: camera pose parameters, rotation angle vector and translation vector.
- $\{\boldsymbol{y}_j\}_{j=1}^{\mathrm{n}}$: camera characteristic parameters.
- n: the total number of image frames.
- $\mathrm{s} = 9\mathrm{n} + 3\mathrm{m}$: the total number of parameters
- $(i, j) \in \mathcal{S} \subset \{1, ..., \mathrm{m}\} \times \{1, ..., \mathrm{n}\}$ if and only if the i-th feature point P_i is observed in the j-th frame. $\mathcal{S}$ is defined as the observation field of the dataset.
- $\mathrm{l} = \|\mathcal{S}\|$: the total number of observations.
- $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}} \subset \mathbb{R}^2$: 2D coordinates of the observations.

If $(i, j) \in \mathcal{S}$, a single reprojection error for one feature point P_i in j-th image frame is presented as the difference between its observed position and its reprojected position,

$$\boldsymbol{F}_{ij} = \boldsymbol{c}_{ij}^{'} - \mathbf{b}_{ij} = \begin{pmatrix} c_{ij,\mathrm{x}}^{'} \\ c_{ij,\mathrm{y}}^{'} \end{pmatrix} - \begin{pmatrix} \mathrm{b}_{ij,\mathrm{x}} \\ \mathrm{b}_{ij,\mathrm{y}} \end{pmatrix} \quad . \tag{2.15}$$

$$\boldsymbol{c}_{ij}^{'} = \Pi(\boldsymbol{x}_i, \boldsymbol{\phi}_j, \boldsymbol{t}_j, \boldsymbol{y}_j) \ . \tag{2.16}$$

$$\boldsymbol{R}_j = (\mathbf{I} - \boldsymbol{n}_{\boldsymbol{\phi}_j}\boldsymbol{n}_{\boldsymbol{\phi}_j}^T)\cos(\|\boldsymbol{\phi}_j\|) + \widehat{\boldsymbol{n}}_{\boldsymbol{\phi}_j}\sin(\|\boldsymbol{\phi}_j\|) + \boldsymbol{n}_{\boldsymbol{\phi}_j}\boldsymbol{n}_{\boldsymbol{\phi}_j}^T \ . \tag{2.17}$$

$${}_{C_j}\boldsymbol{x}_i = \begin{pmatrix} {}_{C_j}x_{i,\mathrm{x}} \\ {}_{C_j}x_{i,\mathrm{y}} \\ {}_{C_j}x_{i,\mathrm{z}} \end{pmatrix} = \boldsymbol{R}_j \begin{pmatrix} {}_{O}x_{i,\mathrm{x}} \\ {}_{O}x_{i,\mathrm{y}} \\ {}_{O}x_{i,\mathrm{z}} \end{pmatrix} + \begin{pmatrix} t_{j,\mathrm{x}} \\ t_{j,\mathrm{y}} \\ t_{j,\mathrm{z}} \end{pmatrix} \ . \tag{2.18}$$

$$\boldsymbol{c}_{ij} = \begin{pmatrix} c_{ij,\mathrm{x}} \\ c_{ij,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} -f_j \times \frac{{}_{C_j}x_{i,\mathrm{x}}}{{}_{C_j}x_{i,\mathrm{z}}} \\ -f_j \times \frac{{}_{C_j}x_{i,\mathrm{y}}}{{}_{C_j}x_{i,\mathrm{z}}} \end{pmatrix} \ . \tag{2.19}$$

$$\boldsymbol{c}_{ij}^{'} = \begin{pmatrix} c_{ij,\mathrm{x}}^{'} \\ c_{ij,\mathrm{y}}^{'} \end{pmatrix} = (1 + k_{1,j} \times \|\boldsymbol{c}_{ij}\|^2 + k_{2,j} \times \|\boldsymbol{c}_{ij}\|^4) \times \begin{pmatrix} c_{ij,\mathrm{x}} \\ c_{ij,\mathrm{y}} \end{pmatrix} \ . \tag{2.20}$$

$$\boldsymbol{y}_j = \begin{pmatrix} f_j \\ k_{1,j} \\ k_{2,j} \end{pmatrix} \ . \tag{2.21}$$

$\boldsymbol{c}_{ij}^{'}$ is the reprojected 2D coordinates, seeing (2.14). The reprojection function Π merges (2.17), (2.18), (2.19), and (2.20) together. $\boldsymbol{x}_i$ is the i-th feature point coordinates (3D) in original inertial system; $\boldsymbol{\phi}_j$, $\boldsymbol{t}_j$, and $\boldsymbol{y}_j$ are the camera parameters of the j-th image frame. Each image has different camera pose, $\boldsymbol{\phi}_j$ and $\boldsymbol{t}_j$. In this case, even if all images are captured with the same camera in collection, i.e. $\boldsymbol{y}_j$ should maintain the same for each image, these camera characteristic parameters are also optimized separately for each image.
Then, the optimization process in bundle adjustment is expressed as finding the minimum of an objective function (cost function), which sums the squares of a large number of nonlinear functions up. Thus, the minimization of the objective function is achieved using nonlinear least-squares algorithms [wika].
A simple objective function that quantifies the model fitting error is written as,

$$\psi_f = \sum_{(i,j)\in\mathcal{S}} \Psi(\|\boldsymbol{F}_{ij}\|) = \sum_{(i,j)\in\mathcal{S}} \frac{1}{2}\|\boldsymbol{F}_{ij}(\boldsymbol{x}_i, \boldsymbol{\phi}_j, \boldsymbol{t}_j, \boldsymbol{y}_j)\|^2 \ . \tag{2.22}$$

Furthermore, to simplify the following matrix manipulations, all camera parameters and feature coordinates parameters are jointed together to form a new parameter vector $\boldsymbol{u}$ with the dimension of s $\times$ 1.

$$\boldsymbol{u} = \{\{\boldsymbol{\phi}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{t}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{y}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{x}_i\}_{i=1}^{\mathrm{m}}\} \ , \tag{2.23}$$

more specifically,

$$\boldsymbol{u} = \begin{pmatrix} \boldsymbol{\phi}_1 \\ \boldsymbol{t}_1 \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{\phi}_j \\ \boldsymbol{t}_j \\ \boldsymbol{y}_j \\ \vdots \\ \boldsymbol{\phi}_{\mathrm{n}} \\ \boldsymbol{t}_{\mathrm{n}} \\ \boldsymbol{y}_{\mathrm{n}} \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{x}_{\mathrm{m}} \end{pmatrix} . \tag{2.24}$$

All single reprojection error vectors (dimension of 2×1), $\{\boldsymbol{F}_{ij}\}_{(i,j)\in\mathcal{S}}$, also form a new reprojection error vector $\boldsymbol{F}$ with dimension of $2\mathrm{l}\times1$, since each single reprojection error vector contains the errors in x and y.

$$\boldsymbol{F} = \begin{pmatrix} \boldsymbol{F}_{11} & \boldsymbol{F}_{12} & \dots & \boldsymbol{F}_{1\mathrm{n}} & \boldsymbol{F}_{21} & \dots & \boldsymbol{F}_{ij} & \dots & \boldsymbol{F}_{\mathrm{mn}} \end{pmatrix}^T_{(i,j)\in\mathcal{S}} . \tag{2.25}$$

Then, this optimization problem is expressed as,

$$\min_{\boldsymbol{u}} \psi_f(\boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{u})\|^2 . \tag{2.26}$$

2.2 Numerical Optimization Algorithm

The last fifty years have seen the development of a powerful collection of algorithms for unconstrained optimization of smooth functions, e.g. bundle adjustment.
In this section, we discuss these algorithms with the formulas in bundle adjustment. All algorithms for unconstrained minimization start from a point, $\boldsymbol{u}^0$, which is chosen according to some prior knowledge about the system and the dataset, or by algorithm. A proper starting point is a crucial factor to find optimal in the iterative process. Beginning at $\boldsymbol{u}^0$, optimization algorithms generate a sequence of iterates that is terminated when either no more progress can be made or when a solution point has been approximated with sufficient accuracy [NW06].
The algorithms use information about the function ψ_f at $\boldsymbol{u}^k$ and its derivatives, and possibly also information from earlier iterates, to determine how to move to the next iteration $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{\delta u}^k$. This increment step, $\boldsymbol{\delta u}^k$, is decided through a

model according to Taylor' Theorem. The criterion is the objective function, ψ_f, is decreased after some prescribed number of iterations, e.g. $\psi_f(\boldsymbol{u}^{k+5}) < \psi_f(\boldsymbol{u}^k)$. Most numerical optimization algorithms are built based on Taylor' Theorem.

Theorem 2.2.1 (Taylor's Theorem) *suppose that $\psi_f(\boldsymbol{u})$: is continuously differentiable. Then,*

$$\psi_f(\boldsymbol{u}+\boldsymbol{\delta u}) = \psi_f(\boldsymbol{u}) + \nabla\psi_f^T(\boldsymbol{u}+a\boldsymbol{\delta u})\boldsymbol{\delta u} \ , \tag{2.27}$$

for some $a \in (0,\ 1)$. Moreover, if $\psi_f(\boldsymbol{u})$ is twice continuously differentiable,

$$\nabla\psi_f(\boldsymbol{u}+\boldsymbol{\delta u}) = \nabla\psi_f(\boldsymbol{u}) + \int_0^1 \nabla^2\psi_f(\boldsymbol{u}+a\boldsymbol{\delta u})\boldsymbol{\delta u} da \tag{2.28}$$

and then,

$$\psi_f(\boldsymbol{u}+\boldsymbol{\delta u}) = \psi_f(\boldsymbol{u}) + \nabla\psi_f^T(\boldsymbol{u})\boldsymbol{\delta u} + \frac{1}{2}\boldsymbol{\delta u}^T\nabla^2\psi_f(\boldsymbol{u}+a\boldsymbol{\delta u})\boldsymbol{\delta u} \ , \tag{2.29}$$

for some $a \in (0,\ 1)$.

2.2.1 Line Search and Trust Region

To determine the increment step $\boldsymbol{\delta u}$, there are two fundamental strategies, line search and trust region. In the line search strategy, the algorithm firstly picks an initial value $\boldsymbol{u}^0$. Then, in the following iteration step before convergence, it finds a search direction $\boldsymbol{p}^k$ from $\boldsymbol{u}^k$ in each step. $\boldsymbol{p}^k$ must be ensured a descent direction, i.e. $\nabla\psi_f^{k^T}\boldsymbol{p}^k < 0$, so that for a small enough step along the direction $\boldsymbol{p}^k$, the objective function decreases [Hau07a]. Along this direction, it finds a suitable step length α^k, so that

$$\psi_f(\boldsymbol{u}^k + \alpha^k\boldsymbol{p}^k) < \psi_f^k = \psi_f(\boldsymbol{u}^k) \ , \tag{2.30}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \alpha^k\boldsymbol{p}^k \ . \tag{2.31}$$

In the trust region strategy, the objective function is also optimized iteratively after initialization. However, the objective function is approximated in each step with a quadratic model function which is trusted in a simple region, a ball or a ellipsoid of specified radius in a specified norm around the current point $\boldsymbol{u}^k$. The widely used model function is quadratic form, and the subproblem in each iteration step is expressed as,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + \boldsymbol{g}^{k^T}\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}^k\boldsymbol{p} \quad \text{s.t. } \|\boldsymbol{p}\| \le \Delta^k \ . \tag{2.32}$$

Here, $\boldsymbol{g}^k$ is the gradient vector at the current point, i.e. $\nabla\psi_f^T(\boldsymbol{u}^k)$; $\boldsymbol{H}^k$ is in general the Hessian matrix at the current point, or its approximation. Δ^k represents the radius of the trust region, which is updated according to how similar the model

function is as the real objective function. The details of this part is demonstrated in Section 3.3.3. The minimizer is either reached at,

$$\boldsymbol{p}^k = -(\boldsymbol{H}^k)^{-1}\boldsymbol{g}^k \ , \text{ if } \|\boldsymbol{p}^k\| \leq \Delta^k \text{ and } \boldsymbol{H}^k \text{ positive definite,} \tag{2.33}$$

or at the boundary of the trust region.
The minimizer of the model function in the trust region is chosen as the increment step $\boldsymbol{p}^k$ related to current point $\boldsymbol{u}^k$. $\boldsymbol{p}^k$ has a similar structure as $\boldsymbol{u}^k$,

$$\boldsymbol{p}^k = \{\{\boldsymbol{\delta\phi}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta t}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta y}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta x}_i^k\}_{i=1}^{\mathrm{m}}\} \ . \tag{2.34}$$

If the real objective function is reduced with this step, the increment step and the trust region radius are accepted, i.e. $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{p}^k$. If the candidate solution does not produce a sufficient decrease, which means the trust region is too large, it is shrunken and resolved [Hau07b].

2.2.2 Gradient Descent Algorithm

The gradient descent algorithm is one of the standard optimization algorithm of non-constraint problem. It is realized with the strategy of line search. In gradient descent algorithm, the picked search direction is against the current gradient direction, i.e. along $-\nabla\psi_f^{kT}$. The step length, α^k, is set to a small constant or searched with various methods, e.g. steepest descent algorithm, Wolfe condition. The more details please see [NW06].

2.2.3 Newton's Method

In Newton's Method, the search direction $\boldsymbol{p}^k$ is obtained through,

$$\boldsymbol{p}^k = -(\nabla^2\psi_f^k)^{-1}\nabla\psi_f^k \ . \tag{2.35}$$

Since Hessian matrix $\nabla^2\psi_f^k$ may not be positive definite, $\boldsymbol{p}^k$ may not be a descent direction. To solve this problem, either we can adjust the Hessian matrix to ensure it is positive definite, and use line search; or we can use the trust region to restrict the boundary. If the Hessian matrix is positive definite, the results of Newton's Method is converged quadratically [NW06].
For Newton's Method with trust region, if $\nabla^2\psi_f^k$ is not positive semidefinite, $\nabla^2\psi_f^k$ should be adjusted to be semidefinite, often added with a factorized diagonal matrix. This strategy is also used in the following Levenberg-Marquardt Algorithm, seeing Section 2.2.6. If $\nabla^2\psi_f^k$ is positive semidefinite, we solve (2.35). If $\boldsymbol{p}^k$ is located in the trust region boundary, i.e. $\|\boldsymbol{p}^k\| \leq \Delta^k$, the increment step is chosen as $\boldsymbol{p}^k$; otherwise, the optimal increment step is searched on the surface of the ball (trust region boundary), e.g. with dogleg-method.

2.2.4 Quasi-Newton's Method

Since the precise calculation of Hessian matrix often consumes lots of time, and is also unnecessary sometimes, in Quasi-Newton's Method, the Hessian matrix is replaced with its approximation which can be easily calculated. The search direction of Quasi-Newton's Method is expressed as,

$$\boldsymbol{p}^k = -(\boldsymbol{H'}^k)^{-1}\nabla\psi_f^k \ . \tag{2.36}$$

The symmetric and positive definite matrix $\boldsymbol{H'}^k$ is updated at every iteration by a quasi-Newton updating formula, such as, BFGS formula, SR1 Method [NW06]. The convergence rate of Quasi-Newton's Method is super-linear.

2.2.5 Gauss Newton Algorithm

For unconstrained least square optimization problem, such as bundle adjustment, Newton's Method is described in a much simpler form. Instead of solving the standard Newton equations (2.35), the Hessian matrix and the gradient vector are obtained in a rapid process through Jacobian matrix [Ryc14]. From (2.25) and (2.24), the reprojection error vector $\boldsymbol{F}$ can be considered as residual vector with a mapping: $\mathbb{R}^{\mathrm{s}} \to \mathbb{R}^{2\mathrm{l}}$. Jacobian matrix is formed,

$$\boldsymbol{F} = \begin{pmatrix} F_1 & F_2 & \dots & F_i & \dots & F_{2\mathrm{l}} \end{pmatrix}^T \ , \tag{2.37}$$

$$\boldsymbol{u} = \begin{pmatrix} u_1 & u_2 & \dots & u_j & \dots & u_{\mathrm{s}} \end{pmatrix}^T \ , \tag{2.38}$$

$$\psi_f(\boldsymbol{u}) = \sum_{i=1}^{2\mathrm{l}} \Psi_f(\|F_i(\boldsymbol{u})\|) = \sum_{i=1}^{2\mathrm{l}} \frac{1}{2}\|F_i(\boldsymbol{u})\|^2 \ . \tag{2.39}$$

$$\boldsymbol{J} = \nabla\boldsymbol{F} = \begin{pmatrix} \frac{\partial F_i}{\partial u_j} \end{pmatrix}_{i=1\dots 2\mathrm{l},\ j=1\dots \mathrm{s}} \ . \tag{2.40}$$

The gradient $\boldsymbol{g}$ and the Hessian matrix $\boldsymbol{H}$ is expressed as,

$$\boldsymbol{g} = \nabla\psi_f = \boldsymbol{J}^T\boldsymbol{F} \ , \tag{2.41}$$

$$\boldsymbol{H} = \nabla^2\psi_f = \boldsymbol{J}^T\boldsymbol{J} + \sum_{i=1}^{2\mathrm{l}} F_i\nabla^2 F_i \ . \tag{2.42}$$

In bundle adjustment, the first derivatives, $\boldsymbol{J}$, is relatively easily calculated with analytic formula. This availability of the first part in (2.42) of "for free" is the distinctive feature of least-squares problems. Moreover, this term $\boldsymbol{J}^T\boldsymbol{J}$ is often quite larger than the second term in (2.42), either because the residuals F_i are close to affine near the solution (that is, the $\nabla^2 F_i$ are relatively small) or because of small residuals (that is, the F_i are relatively small). In most practical situation, the second

term is neglected in comparison with the first term. [NW06] In the following chapter, we only use the first term in (2.42) as Hessian matrix in bundle adjustment.
Therefore, the search direction in (2.35) is replaced with solving equation in each iteration step,

$$\boldsymbol{J}^{k^T}\boldsymbol{J}^k\boldsymbol{p}^k = -\boldsymbol{J}^{k^T}\boldsymbol{F}^k \ . \tag{2.43}$$

In Gauss-Newton Algorithm $\boldsymbol{J}^T\boldsymbol{J}$ is a close approximation to Hessian matrix, and its convergence rate is similar to Newton's Method. Besides, if $\boldsymbol{J}$ has full rank and the gradient $\boldsymbol{g}$ is nonzero, the direction $\boldsymbol{p}$ is a descent direction for the objective function [NW06].

$$\boldsymbol{p}^T\boldsymbol{g} = \boldsymbol{p}^T\boldsymbol{J}^T\boldsymbol{F} = -\boldsymbol{p}^T\boldsymbol{J}^T\boldsymbol{J}\boldsymbol{p} = -\|\boldsymbol{J}\boldsymbol{p}\|^2 \leq 0 \ . \tag{2.44}$$

Moreover, the minimization process of the objective function is also an iteratively updated process of the error vector $\boldsymbol{F}$. At each iteration, (2.43) is solved to adjust $\boldsymbol{u}$ with a increment vector $\boldsymbol{p}$. If using linearization, the error vector is approximated with $\boldsymbol{F}(\boldsymbol{u}+\boldsymbol{p}) = \boldsymbol{F}(\boldsymbol{u}) + \boldsymbol{J}(\boldsymbol{u})\boldsymbol{p}$. The objective function is updated with,

$$\psi_f(\boldsymbol{u}+\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{u}) + \boldsymbol{J}(\boldsymbol{u})\boldsymbol{p}\|^2 \ . \tag{2.45}$$

Therefore, in each iteration step, the increment step of parameter vector $\boldsymbol{p}^k$ is obtained by solving,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} \frac{1}{2}\|\boldsymbol{F}^k + \boldsymbol{J}^k\boldsymbol{p}\|^2 \ . \tag{2.46}$$

This solved $\boldsymbol{p}^k$ is actually equivalent with the search direction $\boldsymbol{p}^k$ obtained from (2.43). In Gauss Newton Algorithm (GNA), either we can use line search based on the search direction, or use trust region strategies to set a boundary for (2.46).
For Gauss Newton Algorithm with trust region strategy, the optimization step is described as (2.46) with the constraint $\|\boldsymbol{p}\| \leq \Delta^k$. The corresponding model function is in quadratic form as (2.32), with $\boldsymbol{g}^k = \boldsymbol{J}^{k^T}\boldsymbol{F}^k$, and $\boldsymbol{H}^k = \boldsymbol{J}^{k^T}\boldsymbol{J}^k$.

2.2.6 Levenberg-Marquardt Algorithm

Lots of researchers implement Levenberg-Marquardt Algorithm in unconstrained non-linear least square optimization [Lou05] [Mor78] [Sha98]. Similar with the above algorithms, Levenberg-Marquardt Algorithm (LMA) also only provide a solution of the local minimum, which is not necessary a global minimum [wike]. Instead of explicit trust region boundary, LMA interpolates between the Gauss Newton Algorithm and Gradient Descent Algorithm, which yields a implicit boundary for increment step. In each iteration step, LMA solves,

$$\boldsymbol{H}_{\boldsymbol{\mu}}^k\boldsymbol{p}^k = -\boldsymbol{g}^k \ , \tag{2.47}$$

$$\boldsymbol{g}^k = \boldsymbol{J}^{k^T}\boldsymbol{F}^k \ , \tag{2.48}$$

$$\boldsymbol{H}_{\boldsymbol{\mu}}^{k} = \boldsymbol{J}^{k^T}\boldsymbol{J}^{k} + \mu\boldsymbol{D} \ , \tag{2.49}$$

μ is a nonnegative damping factor which controls the switching between Gauss Newton Algorithm and Gradient Descent Algorithm.
$\boldsymbol{D}$ is damping matrix. In the original LMA, $\boldsymbol{D} = \mathbf{I}$. If $\mu \to 0$, LMA changes to Gauss Newton Algorithm without trust region boundary, i.e. the radius of trust region is infinite large; if $\mu \to +\infty$, LMA changes to pure Gradient Descent Algorithm with a infinite small step length, i.e. the radius of trust region is infinite small. Obviously, in LMA, the trust region radius is coupled with interpolated direction. Besides, in [ASSS10], they also introduce a new damping matrix formed by the diagonal elements of Hessian matrix, seeing (2.50). About the comparison of different damping matrix, please see Section 3.3.4.

$$\boldsymbol{D} = diag(\boldsymbol{H}) \ . \tag{2.50}$$

In fact, LMA presents a minimization problem in k-th step of,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} \frac{1}{2}\|\boldsymbol{J}^{k}\boldsymbol{p} + \boldsymbol{F}^{k}\|^2 + \frac{1}{2}\mu\|\boldsymbol{D}'\boldsymbol{p}\|^2 \ , \tag{2.51}$$

with

$$\boldsymbol{D}'^{T}\boldsymbol{D}' = \boldsymbol{D}. \tag{2.52}$$

The above equation actually presents a varied quadratic model function around the current point without boundary restriction in each iteration step. Thus, the optimization in the k-th step is rewritten as (with $\boldsymbol{D} = \mathbf{I}$),

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + (\boldsymbol{J}^{k^T}\boldsymbol{F}^{k})^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T(\boldsymbol{J}^{k^T}\boldsymbol{J}^{k} + \mu\mathbf{I})\boldsymbol{p} \ . \tag{2.53}$$

The damping factor is updated in each iteration according to how well the model function approximates the real objective function. The details of damping factor updating please see Section 3.3.4. In general, LMA is more robust than GNA, which means it can find a solution even if its initialization is far off the final minimum [NW06].

2.3 Dataset

In the whole thesis, there are altogether two types of datasets used.
The first type dataset is real dataset provided by GRAIL Lab, University of Washington [ASSS]. They provide many datasets with different sizes collected by different devices. We use the first dataset in Ladybug dataset in Chapter 3 to test some basic features and performance of the algorithms. This dataset is a medium-scale dataset, containing 49 image frames, 7776 feature points, and 31843 observations. This dataset also provide an initial point for optimization. Bundle adjustment is highly non-convex, non-linear optimization problem, thus, there exist plenty of local

optimal points. Only when the initial point and the optimization process are carefully selected, the solution can approximate to the global optimal point. This thesis only focuses on finding a local optimal point with a given initial point (to ensure the found local optimal point is not far from global optimal point).
In order to find the general characteristics of the algorithms and compare the relative performance among algorithms, it is unnecessary to test them on a large-scale dataset. Thus, in the first step, we use this Ladybug dataset considering the convenience. Besides, in some sections, only part of this dataset is enough for us to clarify the problem. We randomly choose ten frames in the dataset, extract the corresponding feature points which appear at least in two frames, and confirm the related observations to form a sub-dataset.
The biggest disadvantage of these real datasets is that they do not contain the ground truth. Besides, the size and the sparsity of these real datasets are not controllable such that sometimes we can not select a dataset meeting our requirements arbitrarily. Therefore, in Section 5.1, we provide a method which can generate a synthetic bundle adjustment dataset with defined size and defined sparsity, i.e. a dataset with arbitrary number of frames, points and observations. Besides, the noise of observations, the outliers of observations and the initial point are also controllable. About the details of the synthetic dataset please see Chapter 5.

Chapter 3

Damped Inexact Newton's Method in Bundle Adjustment

In this chapter, we discuss some baseline implementations in bundle adjustment, some problems faced in the baseline implementation. Then, we introduce a novel damped inexact Newton's method combining the advantages from inexact Newton's method and truncated Newton's method, which yields a better performance related to baseline implementations.

3.1 Matrix Form in Bundle Adjustment

Firstly, the matrix form in bundle adjustment is described. Due to the specific feature, the matrix used in bundle adjustment can be calculated in a more efficient way.

3.1.1 Reprojection Error Vector

For l observations, there are l single reprojection error vectors with dimension of 2×1, which is computed with (2.15) based on its corresponding 3D feature positions and camera parameters. In bundle adjustment, there are lots of observations in one dataset, which can be calculated in parallel. Then, all single reprojection error vectors construct a whole reprojection error vector (2.25).

3.1.2 Partial Derivative

In order to get the Jacobian matrix, we should calculate the analytical partial derivatives of the single reprojection error (2.15).

$$\boldsymbol{F}_{ij} = \boldsymbol{c}_{ij}^{'} - \mathbf{b}_{ij} = \Pi(\boldsymbol{x}_i, \boldsymbol{\phi}_j, \boldsymbol{t}_j, \boldsymbol{y}_j) - \mathbf{b}_{ij} \ . \tag{3.1}$$

The single reprojection error $\boldsymbol{F}_{ij}$ is a function of corresponding feature point coordinates with the dimension of 3×1, i.e. $\boldsymbol{x}_i$, and corresponding camera parameters with

dimension of 9×1, built with $\boldsymbol{\phi}_j$, $\boldsymbol{t}_j$, and $\boldsymbol{y}_j$ respectively (seeing Section 2.1). Since there are several intermediate variables in (3.1), the partial derivatives are analytically calculated using chain rule. The computational process of single reprojection error, (2.6), (2.11), (2.13), and (2.14), is reformed as below,

$$\boldsymbol{R}_j = (\mathbf{I} - \boldsymbol{n}_{\boldsymbol{\phi}_j}\boldsymbol{n}_{\boldsymbol{\phi}_j}{}^T)\cos(\|\boldsymbol{\phi}_j\|) + \widehat{\boldsymbol{n}}_{\boldsymbol{\phi}_j}\sin(\|\boldsymbol{\phi}_j\|) + \boldsymbol{n}_{\boldsymbol{\phi}_j}\boldsymbol{n}_{\boldsymbol{\phi}_j}{}^T \ , \tag{3.2}$$

$$\boldsymbol{n}_{\boldsymbol{\phi}_j} = \frac{\boldsymbol{\phi}_j}{\|\boldsymbol{\phi}_j\|} \ , \tag{3.3}$$

$$\widehat{\boldsymbol{n}}_{\boldsymbol{\phi}_j} = \boldsymbol{n}_{\boldsymbol{\phi}_j}\times = \begin{pmatrix} 0 & -n_{\boldsymbol{\phi}_j,\mathrm{z}} & n_{\boldsymbol{\phi}_j,\mathrm{y}} \\ n_{\boldsymbol{\phi}_j,\mathrm{z}} & 0 & -n_{\boldsymbol{\phi}_j,\mathrm{x}} \\ -n_{\boldsymbol{\phi}_j,\mathrm{y}} & n_{\boldsymbol{\phi}_j,\mathrm{x}} & 0 \end{pmatrix} \ , \tag{3.4}$$

$$\boldsymbol{X}_{ij} = \begin{pmatrix} X_{ij,\mathrm{x}} \\ X_{ij,\mathrm{y}} \\ X_{ij,\mathrm{z}} \end{pmatrix} = \boldsymbol{R}_j\boldsymbol{x}_i + \boldsymbol{t}_j \ , \tag{3.5}$$

$$\boldsymbol{c}_{ij} = \begin{pmatrix} c_{ij,\mathrm{x}} \\ c_{ij,\mathrm{y}} \end{pmatrix} = \begin{pmatrix} -\frac{X_{ij,\mathrm{x}}}{X_{ij,\mathrm{z}}} \\ -\frac{X_{ij,\mathrm{y}}}{X_{ij,\mathrm{z}}} \end{pmatrix} \ , \tag{3.6}$$

$$\boldsymbol{c}_{ij}^{'} = \begin{pmatrix} c_{ij,\mathrm{x}}^{'} \\ c_{ij,\mathrm{y}}^{'} \end{pmatrix} = y_{j,1}(1 + y_{j,2} \times \|\boldsymbol{c}_{ij}\|^2 + y_{j,3} \times \|\boldsymbol{c}_{ij}\|^4) \times \begin{pmatrix} c_{ij,\mathrm{x}} \\ c_{ij,\mathrm{y}} \end{pmatrix} \ , \tag{3.7}$$

$$\boldsymbol{y}_j = \begin{pmatrix} y_{j,1} \\ y_{j,2} \\ y_{j,3} \end{pmatrix} = \begin{pmatrix} f_j \\ k_{1,j} \\ k_{2,j} \end{pmatrix} \ . \tag{3.8}$$

The partial derivatives are expressed using chain rule as,

$$\frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}_{ij}^{'}} = \mathbf{I}_{2\times 2} \ . \tag{3.9}$$

$$\frac{\partial \boldsymbol{c}_{ij}^{'}}{\partial \boldsymbol{y}_j} = \begin{pmatrix} dis \times c_{ij,\mathrm{x}} & f_j \times \|\boldsymbol{c}_{ij}\|^2 \times c_{ij,\mathrm{x}} & f_j \times \|\boldsymbol{c}_{ij}\|^4 \times c_{ij,\mathrm{x}} \\ dis \times c_{ij,\mathrm{y}} & f_j \times \|\boldsymbol{c}_{ij}\|^2 \times c_{ij,\mathrm{y}} & f_j \times \|\boldsymbol{c}_{ij}\|^4 \times c_{ij,\mathrm{y}} \end{pmatrix} \ , \tag{3.10}$$

$$dis = 1 + k_{1,j} \times \|\boldsymbol{c}_{ij}\|^2 + k_{2,j} \times \|\boldsymbol{c}_{ij}\|^4 \ . \tag{3.11}$$

$$\frac{\partial \boldsymbol{c}_{ij}^{'}}{\partial \boldsymbol{c}_{ij}} = \begin{pmatrix} \frac{\partial c_{ij,\mathrm{x}}^{'}}{\partial c_{ij,\mathrm{x}}} & \frac{\partial c_{ij,\mathrm{x}}^{'}}{\partial c_{ij,\mathrm{y}}} \\ \frac{\partial c_{ij,\mathrm{y}}^{'}}{\partial c_{ij,\mathrm{x}}} & \frac{\partial c_{ij,\mathrm{y}}^{'}}{\partial c_{ij,\mathrm{y}}} \end{pmatrix} \ . \tag{3.12}$$

$$\frac{\partial c_{ij,\mathrm{x}}^{'}}{\partial c_{ij,\mathrm{x}}} = f_j(1 + k_{1,j}(3{c_{ij,\mathrm{x}}}^2 + {c_{ij,\mathrm{y}}}^2) + k_{2,j}(5{c_{ij,\mathrm{x}}}^4 + {c_{ij,\mathrm{y}}}^4 + 6{c_{ij,\mathrm{x}}}^2{c_{ij,\mathrm{y}}}^2)) \ , \tag{3.13}$$

$$\frac{\partial c_{ij,\mathrm{x}}^{'}}{\partial c_{ij,\mathrm{y}}} = \frac{\partial c_{ij,\mathrm{y}}^{'}}{\partial c_{ij,\mathrm{x}}} = 2f_j c_{ij,\mathrm{x}}c_{ij,\mathrm{y}}(k_{1,j} + 2k_{2,j}({c_{ij,\mathrm{x}}}^2 + {c_{ij,\mathrm{y}}}^2)) \ , \tag{3.14}$$

$$\frac{\partial c'_{ij,\mathrm{y}}}{\partial c_{ij,\mathrm{y}}} = f_j(1 + k_{1,j}(3{c_{ij,\mathrm{y}}}^2 + {c_{ij,\mathrm{x}}}^2) + k_{2,j}(5{c_{ij,\mathrm{y}}}^4 + {c_{ij,\mathrm{x}}}^4 + 6{c_{ij,\mathrm{x}}}^2{c_{ij,\mathrm{y}}}^2)) \ . \tag{3.15}$$

$$\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}} = \begin{pmatrix} -\frac{1}{X_{ij,\mathrm{z}}} & 0 & \frac{X_{ij,\mathrm{x}}}{X_{ij,\mathrm{z}}^2} \\ 0 & -\frac{1}{X_{ij,\mathrm{z}}} & \frac{X_{ij,\mathrm{y}}}{X_{ij,\mathrm{z}}^2} \end{pmatrix} \ . \tag{3.16}$$

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{x}_i} = \boldsymbol{R}_j \ . \tag{3.17}$$

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{t}_j} = \mathbf{I}_{3\times 3} \ . \tag{3.18}$$

Considering there exists a small increment step of j-th camera parameters, rotation angle vector and translation vector, $\boldsymbol{\delta\phi}_j$ and $\boldsymbol{\delta t}_j$, around current points in se(3), $\{\boldsymbol{\phi}_j, \boldsymbol{t}_j\}$. The new rotation matrix and translation vector are updated as,

$$\boldsymbol{R_g}(\boldsymbol{\phi}_j + \boldsymbol{\delta\phi}_j) = \mathrm{Exp}(\boldsymbol{\phi}_j + \boldsymbol{\delta\phi}_j) \ , \tag{3.19}$$

$$\boldsymbol{t_g}(\boldsymbol{t}_j + \boldsymbol{\delta t}_j) = \boldsymbol{t}_j + \boldsymbol{\delta t}_j \ . \tag{3.20}$$

This process actually presents a global parameterization form, which is also the general parameterization form used in robotic kinematics. In the following Section 4.2, we also propose a local parameterization form in robotic kinematics.
In order to calculate the derivatives of $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$, we need to formulate the right Jacobian matrix for SO(3). The Jacobian matrix for SO(3) uses $\boxminus$ to define derivatives on the manifold [FCDS17] [Sol17] [Chi11].

$$\boldsymbol{R}_1 \boxminus \boldsymbol{R}_2 \triangleq \mathrm{Log}(\boldsymbol{R}_2^{-1}\boldsymbol{R}_1) \in \mathbb{R}^3 \ \text{ if } \boldsymbol{R}_1, \boldsymbol{R}_2 \in \mathrm{SO}(3) \ , \tag{3.21}$$

$$\frac{\partial \mathrm{R}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \triangleq \lim_{\boldsymbol{\delta\phi}\to 0} \frac{\mathrm{R}(\boldsymbol{\phi} + \boldsymbol{\delta\phi}) \boxminus \mathrm{R}(\boldsymbol{\phi})}{\boldsymbol{\delta\phi}} \ \text{ if } \mathrm{R}: \ \mathbb{R}^3 \to \mathrm{SO}(3) \ , \tag{3.22}$$

here, the function R maps the rotation angle vector $\boldsymbol{\phi}$ to the rotation matrix $\boldsymbol{R} \in$ SO(3), i.e. $\boldsymbol{R} = \mathrm{R}(\boldsymbol{\phi}) = \mathrm{Exp}(\boldsymbol{\phi})$.
Similar as (3.4), the formula,

$$\boldsymbol{\phi} \in \mathbb{R}^3 \to \widehat{\phi} \in \mathrm{so}(3) \ , \tag{3.23}$$

lifts from a twist to a Lie algebra, with,

$$\widehat{\boldsymbol{\phi}} = \begin{pmatrix} 0 & -\phi_{\mathrm{z}} & \phi_{\mathrm{y}} \\ \phi_{\mathrm{z}} & 0 & -\phi_{\mathrm{x}} \\ -\phi_{\mathrm{y}} & \phi_{\mathrm{x}} & 0 \end{pmatrix} \ . \tag{3.24}$$

This matrix $\widehat{\phi}$ also presents the cross product of $\boldsymbol{\phi}$ seeing (3.4).
The matrix exponential operation maps $\boldsymbol{\phi} \in$ so(3) to SO(3),

$$\mathrm{Exp}(\boldsymbol{\phi}) = \exp(\widehat{\boldsymbol{\phi}}) \in \mathrm{SO}(3) \ . \tag{3.25}$$

The right Jacobian matrix for SO(3) is presented as [Sol17],

$$\begin{aligned}\boldsymbol{J}_r(\boldsymbol{\phi}) &\triangleq \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\phi}} \\ &= \frac{\partial \mathrm{Exp}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \\ &= \lim_{\boldsymbol{\delta\phi}\to 0} \frac{\mathrm{Exp}(\boldsymbol{\phi}+\boldsymbol{\delta\phi}) \boxminus \mathrm{Exp}(\boldsymbol{\phi})}{\boldsymbol{\delta\phi}} \\ &= \lim_{\boldsymbol{\delta\phi}\to 0} \frac{\mathrm{Log}(\mathrm{Exp}(\boldsymbol{\phi})^{-1}\mathrm{Exp}(\boldsymbol{\phi}+\boldsymbol{\delta\phi}))}{\boldsymbol{\delta\phi}} \\ &= \mathbf{I}_{3\times 3} - \frac{1-\cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^2}\widehat{\boldsymbol{\phi}} + \frac{\|\boldsymbol{\phi}\| - \sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^3}\widehat{\boldsymbol{\phi}}^2.\end{aligned} \tag{3.26}$$

The formula of $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$ is then presented as,

$$\begin{aligned}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} &= \left.\frac{\partial \mathrm{Exp}(\boldsymbol{\phi}_j+\boldsymbol{\delta\phi}_j)\boldsymbol{x}_i}{\partial \boldsymbol{\delta\phi}_j}\right|_{\boldsymbol{\delta\phi}_j=\mathbf{0}} \\ &= -\boldsymbol{R}_j\widehat{\boldsymbol{x}}_i\boldsymbol{J}_r(\boldsymbol{\phi}_j),\end{aligned} \tag{3.27}$$

Since the concrete formula of $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$ is too complex, this partial derivative matrix is obtained through Matlab. Then this formula is simplified in the following form,

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = \begin{pmatrix} \frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{x}}} & \frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{y}}} & \frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{z}}} \\ \frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{x}}} & \frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{y}}} & \frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{z}}} \\ \frac{\partial X_{ij,\mathrm{z}}}{\partial \phi_{j,\mathrm{x}}} & \frac{\partial X_{ij,\mathrm{z}}}{\partial \phi_{j,\mathrm{y}}} & \frac{\partial X_{ij,\mathrm{z}}}{\partial \phi_{j,\mathrm{z}}} \end{pmatrix}, \tag{3.28}$$

$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$ is a function of $\boldsymbol{\phi}_j$ and $\boldsymbol{x}_i$. In order to have a shorter formula in presentation, the subscripts of feature point index i and camera index j are omitted here. $s\boldsymbol{\phi}$ and $c\boldsymbol{\phi}$ are used to replace $\sin(\|\boldsymbol{\phi}\|)$ and $\cos(\|\boldsymbol{\phi}\|)$. Besides, $\phi x_{\bullet\bullet}$ is used to present the product between $\phi_\bullet$ and $x_\bullet$ to , e.g. $\phi x_{\mathrm{xy}} := \phi_{\mathrm{x}} x_{\mathrm{y}}$, $\phi\phi x_{\mathrm{zxy}} := \phi_{\mathrm{z}}\phi_{\mathrm{x}} x_{\mathrm{y}}$.

$$\begin{aligned}\frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{x}}} =& -(\phi x_{\mathrm{xx}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\| + \\ & ((2\phi x_{\mathrm{xx}} + \phi x_{\mathrm{yy}} + \phi x_{\mathrm{zz}})(1-c\boldsymbol{\phi}) + (\phi\phi x_{\mathrm{xyz}} - \phi\phi x_{\mathrm{xzy}})c\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^2 + \\ & ((-\phi\phi x_{\mathrm{xyz}} + \phi\phi x_{\mathrm{xzy}} + \phi\phi_{\mathrm{xx}}(\phi x_{\mathrm{yy}} + \phi x_{\mathrm{zz}} + \phi x_{\mathrm{xx}}))s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^3 + \\ & (2\phi\phi_{\mathrm{xx}}(\phi x_{\mathrm{xx}} + \phi x_{\mathrm{yy}} + \phi x_{\mathrm{zz}})(c\boldsymbol{\phi}-1))/\|\boldsymbol{\phi}\|^4 ,\end{aligned} \tag{3.29}$$

$$\begin{aligned}\frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{y}}} =& (x_{\mathrm{z}} s\boldsymbol{\phi} - \phi x_{\mathrm{yx}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\| + \\ & (\phi x_{\mathrm{xy}}(1-c\boldsymbol{\phi}) + (\phi\phi x_{\mathrm{yyz}} - \phi\phi x_{\mathrm{yzy}})c\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^2 + \\ & ((\phi\phi x_{\mathrm{xxx}} + \phi\phi x_{\mathrm{xyy}} + \phi\phi x_{\mathrm{xzz}} + \phi x_{\mathrm{zy}} - \phi x_{\mathrm{yz}})\phi_{\mathrm{y}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^3 + \\ & (2\phi\phi_{\mathrm{xy}}(\phi x_{\mathrm{xx}} + \phi x_{\mathrm{yy}} + \phi x_{\mathrm{zz}})(c\boldsymbol{\phi}-1))/\|\boldsymbol{\phi}\|^4 ,\end{aligned} \tag{3.30}$$

$$\begin{aligned}\frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{z}}} =&(-x_{\mathrm{y}} s\boldsymbol{\phi} - \phi x_{\mathrm{zx}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|+\\ &(\phi x_{\mathrm{xz}}(1-c\boldsymbol{\phi}) + (\phi\phi x_{\mathrm{zyz}} - \phi\phi x_{\mathrm{zzy}})c\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^2+\\ &((\phi\phi x_{\mathrm{xxx}} + \phi\phi x_{\mathrm{xyy}} + \phi\phi x_{\mathrm{xzz}} + \phi x_{\mathrm{zy}} - \phi x_{\mathrm{yz}})\phi_{\mathrm{z}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^3+\\ &(2\phi\phi_{\mathrm{xz}}(\phi x_{\mathrm{xx}} + \phi x_{\mathrm{zz}} + \phi x_{\mathrm{yy}})(c\boldsymbol{\phi}-1))/\|\boldsymbol{\phi}\|^4 \ .\end{aligned} \tag{3.31}$$

Due to the symmetry of the form, the other two rows of $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$ can be obtained through the subscript replacement rule: x $\rightarrow$ y, y $\rightarrow$ z, and z $\rightarrow$ x. For example, $\frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{x}}}$ can be obtained from $\frac{\partial X_{ij,\mathrm{x}}}{\partial \phi_{j,\mathrm{z}}}$ with this rule, and $\frac{\partial X_{ij,\mathrm{z}}}{\partial \phi_{j,\mathrm{y}}}$ can be obtained from $\frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{x}}}$ with this rule.

$$\begin{aligned}\frac{\partial X_{ij,\mathrm{y}}}{\partial \phi_{j,\mathrm{x}}} =&(-x_{\mathrm{z}} s\boldsymbol{\phi} - \phi x_{\mathrm{xy}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|+\\ &(\phi x_{\mathrm{yx}}(1-c\boldsymbol{\phi}) + (\phi\phi x_{\mathrm{xzx}} - \phi\phi x_{\mathrm{xxz}})c\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^2+\\ &((\phi\phi x_{\mathrm{yyy}} + \phi\phi x_{\mathrm{yzz}} + \phi\phi x_{\mathrm{yxx}} + \phi x_{\mathrm{xz}} - \phi x_{\mathrm{zx}})\phi_{\mathrm{x}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^3+\\ &(2\phi\phi_{\mathrm{yx}}(\phi x_{\mathrm{yy}} + \phi x_{\mathrm{zz}} + \phi x_{\mathrm{xx}})(c\boldsymbol{\phi}-1))/\|\boldsymbol{\phi}\|^4 \ ,\end{aligned} \tag{3.32}$$

$$\begin{aligned}\frac{\partial X_{ij,\mathrm{z}}}{\partial \phi_{j,\mathrm{y}}} =&(-x_{\mathrm{x}} s\boldsymbol{\phi} - \phi x_{\mathrm{yz}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|+\\ &(\phi x_{\mathrm{zy}}(1-c\boldsymbol{\phi}) + (\phi\phi x_{\mathrm{yxy}} - \phi\phi x_{\mathrm{yyx}})c\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^2+\\ &((\phi\phi x_{\mathrm{zzz}} + \phi\phi x_{\mathrm{zxx}} + \phi\phi x_{\mathrm{zyy}} + \phi x_{\mathrm{yx}} - \phi x_{\mathrm{xy}})\phi_{\mathrm{y}} s\boldsymbol{\phi})/\|\boldsymbol{\phi}\|^3+\\ &(2\phi\phi_{\mathrm{zy}}(\phi x_{\mathrm{zz}} + \phi x_{\mathrm{xx}} + \phi x_{\mathrm{yy}})(c\boldsymbol{\phi}-1))/\|\boldsymbol{\phi}\|^4 \ .\end{aligned} \tag{3.33}$$

Therefore, with (3.9), (3.10), (3.11), (3.12), (3.13), (3.14), (3.15), (3.16), (3.17), (3.18), (3.28) the partial derivatives of single reprojection error are expressed as,

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{\phi}_j} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = \frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} \ . \tag{3.34}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{t}_j} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{t}_j} = \frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}} \ . \tag{3.35}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{y}_j} = \frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{y}_j} = \frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{y}_j} \ . \tag{3.36}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{x}_i} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{x}_i} = \frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\boldsymbol{R}_j \ . \tag{3.37}$$

3.1.3 Jacobian Matrix

Jacobian matrix in this bundle adjustment problem (seeing (2.37), (2.38), (2.40)), $\boldsymbol{J} = \nabla \boldsymbol{F} = \frac{d\boldsymbol{F}}{d\boldsymbol{u}}$, is constructed with the partial derivatives of each single reprojection error. The first 9n elements in $\boldsymbol{u}$ present the camera parameters of n image frames; and the left 3m elements present the feature position parameters of m feature points. The dimension of Jacobian matrix is $2\mathrm{l} \times \mathrm{s}$.

Since the single reprojection error has x and y two dimensions, each single reprojection error occupies two rows in Jacobian matrix, and the values in these two rows present the partial derivatives of this single reprojection error. The columns of Jacobian matrix present the partial derivatives on which arguments in parameter vector $\boldsymbol{u}$. It is clear that one single reprojection error is only dependent on one feature point and one set of camera parameters. Thus, there are only 12 non-zero derivatives in each row of Jacobian matrix, of which 9 derivatives correspond to camera parameters and the others correspond to feature point coordinates. The Jacobian matrix in bundle adjustment is highly sparse.

In order to save the memory space and the calculation time, the explicit Jacobian matrix is not necessary to build. Instead, only 2×12 derivatives of each single reprojection error vector obtained from (3.34), (3.35), (3.36), and (3.37), are stored. Later when used, the Jacobian matrix can be quickly rebuilt with these derivatives, their image frame indexes and their feature point indexes.

3.1.4 Verification of Jacobian Matrix

To verify the correctness of our analytical Jacobian matrix, we propose two methods.

In the first method, the Jacobian matrix is calculated with numerical method. Then, the results from numerical method is compared with the results from analytical method. In numerical partial derivatives calculation, we need to set an increment step to each argument. In single reprojection error formula, there are altogether 12 arguments (9 camera parameters and 3 feature point positions). In each partial derivative calculation, only one argument varies with a specific increment step, and the other 11 arguments maintain the same. The partial derivative on this argument is the ratio between the increment of the reprojection error and the specific increment step. This procedure is repeated for all single reprojection errors, i.e. l times. All single reprojection errors use the same set of increment steps. In numerical calculation, a proper increment step is the most crucial factor. We use a loop to search 12 optimal increment steps for 12 arguments based on the provided dataset.

The final chosen increment step set is,

$$\begin{pmatrix} \Delta\phi_{\mathrm{x}} \\ \Delta\phi_{\mathrm{y}} \\ \Delta\phi_{\mathrm{z}} \\ \Delta t_{\mathrm{x}} \\ \Delta t_{\mathrm{y}} \\ \Delta t_{\mathrm{z}} \\ \Delta f \\ \Delta k_1 \\ \Delta k_2 \\ \Delta x_{\mathrm{x}} \\ \Delta x_{\mathrm{y}} \\ \Delta x_{\mathrm{z}} \end{pmatrix} = \begin{pmatrix} 10^{-8} \\ 10^{-8} \\ 10^{-8} \\ 10^{-5} \\ 10^{-5} \\ 10^{-8} \\ 10^{0} \\ 10^{-3} \\ 10^{-3} \\ 10^{-7} \\ 10^{-7} \\ 10^{-8} \end{pmatrix} . \tag{3.38}$$

This optimal numerical step is also suitable for other bundle adjustment datasets, since this increment step reflects the essential characteristics in the geometric transformation of bundle adjustment, not the characteristics just for one specific dataset or one specific point in the argument space. Besides, this optimal step also indicates the relative sensitivity among the arguments. Therefore, this set of increment steps is not only used in numerical Jacobian matrix calculation, but also used in later scaling factor in solvers to reduce the numerical error.

In verification, we randomly choose a point in the argument space ($\mathbb{R}^{\mathrm{s}}$), and calculate the Jacobian matrix with the sub-dataset mentioned in Section 2.3 at this point with numerical method and analytical method. With this increment step set, the second norm of the difference matrix between numerical Jacobian matrix and analytical Jacobian matrix is 0.28. The dimension of the Jacobian matrix of this sub-dataset is 14670×6720. The average norm per element is very small, 10^{-8}.

In the second method, Taylor's Theorem (seeing Theorem 2.2.1) is used to verify the correctness. According to Taylor's Theorem,

$$\boldsymbol{F}(\boldsymbol{u} + \tau\boldsymbol{\delta u}) = \boldsymbol{F}(\boldsymbol{u}) + \tau\boldsymbol{J}(\boldsymbol{u})^T\boldsymbol{\delta u} + \boldsymbol{O}(\tau^2) \ . \tag{3.39}$$

$\boldsymbol{O}(\tau^2)$ presents the higher order remainder term of the approximation with the Jacobian matrix, which is in the second order of the magnitude of τ.

Firstly, an increment step, $\boldsymbol{\delta u}$, and an initial point, $\boldsymbol{u}$, are randomly chosen and maintain invariant in the following process. Secondly, a series of τ is selected, such as,

$$\tau = \{2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, ...\} \ . \tag{3.40}$$

If the analytical Jacobian matrix is correct, the norm of the reminder term will follow a geometric progression with a ratio of $\frac{1}{4}$, when using the above series in (3.39), i.e.

$$\{\|\boldsymbol{O}(\tau^2)\|\} = \{\|\boldsymbol{F}(\boldsymbol{u} + \tau\boldsymbol{\delta u}) - \boldsymbol{F}(\boldsymbol{u}) - \tau\boldsymbol{J}(\boldsymbol{u})^T\boldsymbol{\delta u}\|\} \propto \{\frac{1}{4^n}\} \ . \tag{3.41}$$

The result of $\{\|\boldsymbol{O}(\tau^2)\|\}$ is plotted in Figure 3.1.

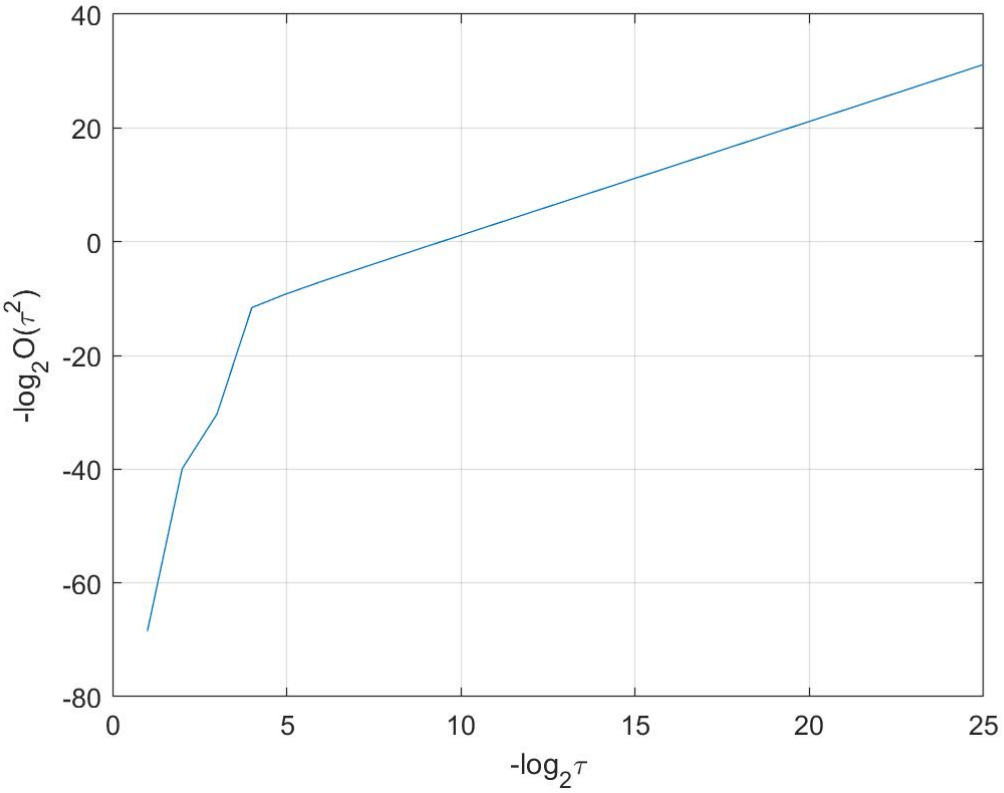

Figure 3.1: The reminder, $\{\|\boldsymbol{O}(\tau^2)\|\}$, varies with τ in (3.40). The slope factor is 2, thus, the analytical derivatives are correct.

3.1.5 Gradient Vector

Since bundle adjustment is a non-linear least square optimization, its gradient vector is expressed following (2.41) as,

$$\boldsymbol{g} = \boldsymbol{J}^T \boldsymbol{F} \ . \tag{3.42}$$

Due to the sparsity of Jacobian matrix $\boldsymbol{J}$, using fully explicit Jacobian matrix to multiply with the reprojection error vector $\boldsymbol{F}$ is inefficiency. In this thesis, we provide a method which rebuilds Jacobian matrix only through several sub-matrices with all non-zero elements. These sub-matrices record the non-zero elements in the original Jacobian matrix, according to each observation. Then, the multiplication between these sub-matrices and $\boldsymbol{F}$, i.e. the gradient vector, is computed. About the details of algorithm please see Appendix A.1.

3.1.6 Hessian Matrix

As talked about in Section 2.2.5, Hessian matrix in bundle adjustment is expressed as,

$$\boldsymbol{H} = \boldsymbol{J}^T \boldsymbol{J} \ . \tag{3.43}$$

Still because of the sparsity of Jacobian matrix, we do not need to explicitly form the Jacobian matrix and compute the Hessian matrix with (3.43). The Jacobian matrix is divided into two sub-matrices. One is camera parameter part $\boldsymbol{J_c}$ with the dimension of $2\mathrm{l} \times 9\mathrm{n}$; the other is feature position part $\boldsymbol{J_p}$ with the dimension of $2\mathrm{l} \times 3\mathrm{m}$. The Jacobian matrix and the Hessian matrix are presented with sub-matrix form as,

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{J_c} & \boldsymbol{J_p} \end{pmatrix} \ , \tag{3.44}$$

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^T & \boldsymbol{C} \end{pmatrix} \ , \tag{3.45}$$

$$\boldsymbol{B} = \boldsymbol{J_c}^T \boldsymbol{J_c} \ , \tag{3.46}$$

$$\boldsymbol{C} = \boldsymbol{J_p}^T \boldsymbol{J_p} \ , \tag{3.47}$$

$$\boldsymbol{E} = \boldsymbol{J_c}^T \boldsymbol{J_p} \ . \tag{3.48}$$

$\boldsymbol{J_c}$ is divided into n sub-matrices with the same dimension of $2\mathrm{l} \times 9$; similarly, $\boldsymbol{J_p}$ is divided into m sub-matrices with the same dimension of $2\mathrm{l} \times 3$.

$$\boldsymbol{J_c} = \begin{pmatrix} \boldsymbol{J_{c,1}} & \boldsymbol{J_{c,2}} & \cdots & \boldsymbol{J_{c,\mathrm{n}}} \end{pmatrix} \ , \tag{3.49}$$

$$\boldsymbol{J_p} = \begin{pmatrix} \boldsymbol{J_{p,1}} & \boldsymbol{J_{p,2}} & \cdots & \boldsymbol{J_{p,\mathrm{m}}} \end{pmatrix} \ . \tag{3.50}$$

Since there are only 9 non-zero elements in each row of $\boldsymbol{J_c}$ and 3 non-zero elements in each row of $\boldsymbol{J_p}$, we have,

$$\boldsymbol{J_{c,i}}^T \boldsymbol{J_{c,j}} \neq \boldsymbol{0}_{9\times 9}, \text{ if and only if } i = j \ ; \tag{3.51}$$

$$\boldsymbol{J_{p,i}}^T \boldsymbol{J_{p,j}} \neq \boldsymbol{0}_{3\times 3}, \text{ if and only if } i = j \ . \tag{3.52}$$

Therefore, $\boldsymbol{B}$ is a block diagonal matrix with the block size of 9×9, and $\boldsymbol{C}$ is a block diagonal matrix with the block size of 3×3.
$\boldsymbol{E}$ is a sparse matrix with the dimension of $9\mathrm{n} \times 3\mathrm{m}$. Here, we divide $\boldsymbol{E}^T$ into $\mathrm{m} \times \mathrm{n}$ sub-matrices with the same dimension of 3×9.

$$\boldsymbol{E}_{ij}^T = \boldsymbol{J}_{\boldsymbol{p},i}^T \boldsymbol{J}_{\boldsymbol{c},j}, \tag{3.53}$$

with $i \in \{1, ..., \mathrm{m}\}$ and $j \in \{1, ..., \mathrm{n}\}$. Only when $\boldsymbol{J_{p},i}$ and $\boldsymbol{J_{c},j}$ have the non-zero elements in at least one same row, $\boldsymbol{E}_{ij}^T \neq \boldsymbol{0}_{3\times 9}$, i.e.

$$\boldsymbol{E}_{ij}^T \neq \boldsymbol{0}_{3\times 9}, \text{ if and only if } (i,j) \in \mathcal{S}. \tag{3.54}$$

The details of the relationship between Jacobian matrix and Hessian matrix are plotted in Figure 3.2 and Figure 3.3. To simplify the plot, we only demonstrate a very small bundle adjustment problem with 4 image frames and 6 feature points.

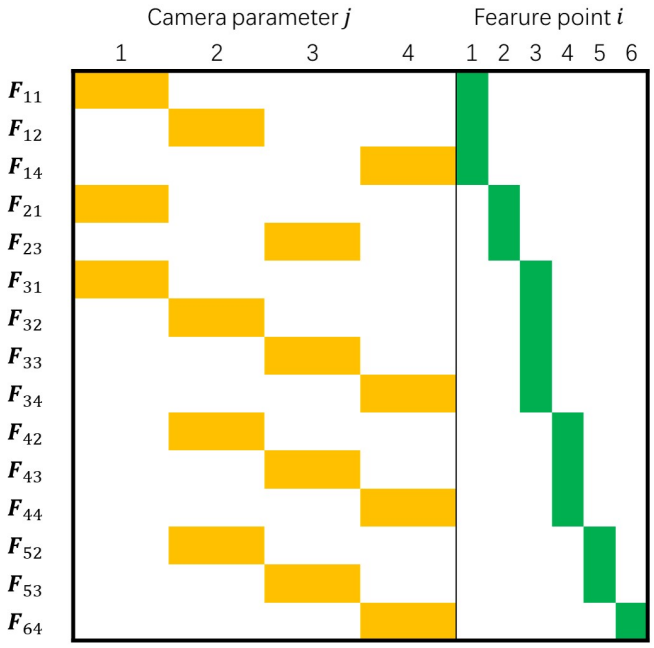

Figure 3.2: Jacobian matrix demonstration of a small bundle adjustment problem. There are 4 image frames and 6 feature points. The observation field is $(ij) \in \{(11), (12), (14), (21), (23), (31), (32), (33), (34), (42), (43), (44), (52), (53), (64)\}$. The yellow block in camera corresponding Jacobian part is of size of 2×9; the green block in feature point corresponding Jacobian part is of size of $\bullet \times 3$.

With the above statement, the Hessian matrix is computed in a more efficient way by avoiding lots of unnecessary computing with "0". About the details of this algorithm, please see Appendix A.1.

In order to demonstrate the efficiency of our algorithm, we also implement the original computation process of Hessian matrix without optimization. The memory consumption and the time consumption are compared among different implementations. All implementations are tested in Matlab based on the sub-dataset of Ladybug dataset provided in Section 2.3. The memory consumption consists of the Jacobian matrix and the Hessian matrix; the computing time counts up from the original sub-dataset to the final Hessian matrix. The implementations used in comparison are original explicit structure without any optimization (ES), explicit structure with the *sparse* command in Matlab (ES-sparse), implicit structure with *sparse* command (IS-sparse), and implicit structure with cell form in Matlab (IS-cell). The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM.

The average processing time and memory consumption of four implementations is demonstrated in Table 3.1.

Obviously, IS-cell implementation consumes less memory and less processing time. Even if IS-cell implementation needs to reconstruct the matrix from the cell form when necessary in the following process, such as $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{E}$, $\boldsymbol{J_c}$, $\boldsymbol{J_p}$, etc. But the

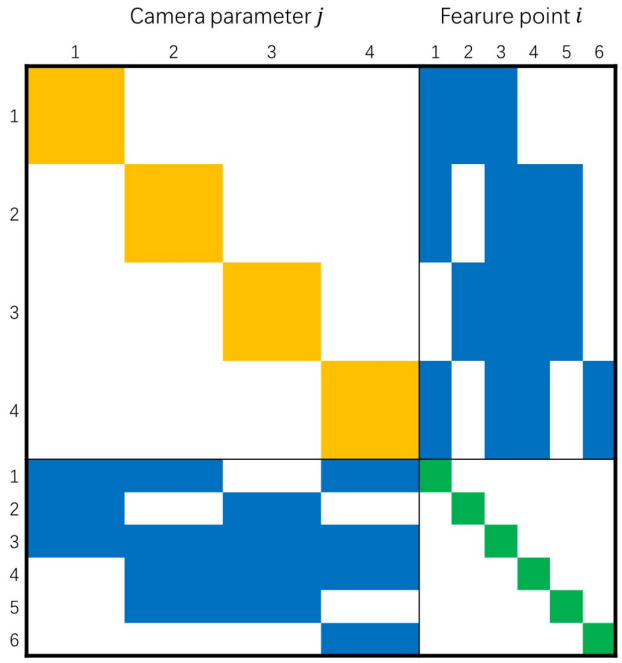

Figure 3.3: Hessian matrix demonstration of a small bundle adjustment problem (same as in Figure 3.2). The yellow block in $\boldsymbol{B}$ is of size of 9×9; the green block in $\boldsymbol{C}$ is of size of 3×3; the blue block in $\boldsymbol{E}^T$ is of size of 3×9. If the blue blocks in $\boldsymbol{E}^T$ is labeled with index (i, j), (i, j)-block is a non-zero matrix, if and only if when $(i, j) \in \mathcal{S}$.

reconstructing of all needed matrices in each iteration step costs only around 0.02s, which is negligible in comparison with other processing step.

3.2 Baseline Implementations of Bundle Adjustment

In this section, some baseline implementations of bundle adjustment are presented. For large-scale least square optimization problem, researchers have developed lots of algorithms, such as, Gauss Newton Algorithm (GNA), Levenberg-Marquardt Algorithm (LMA), Quasi Newton Algorithm, Truncated Newton's Method (TNM), Inexact Newton's Method (TNM), etc. [NW06] Some of these algorithms have already briefly discussed in Section 2.2. Among them, Gauss Newton Algorithm with (scaled) trust region and Levenberg-Marquardt Algorithm build the basic framework in the most bundle adjustment in recent years.
In this section, we select Levenberg-Marquardt Algorithm solved by Cholesky decomposition and Truncated Newton's Method proposed in [ASSS10] as the baseline implementations, which are used to compare the performance of our algorithms.

Table 3.1: Processing time and memory comparison between different matrix form implementations. IS-cell matrix form is used in this thesis.

Implementation	ES	ES-sparse	IS-sparse	IS-cell
Time/s	201	194	1.32	0.37
Memory/MB	7.31	3.02	2.84	2.82

Both LMA and GNA use the square model function to approximate the objective function around the current point, (2.32) and (2.53) respectively. The difference of both algorithms happens mainly in the explicit and implicit trust region formulation. In LMA, like we have talked about in Section 2.2.6, by adding damping factor in the Hessian matrix, LMA owns an implicit trust region. On the other hand, in GNA, the optimal increment step in each iteration is always restricted by an explicit trust region boundary in (2.32), seeing Section 2.2.5.
In LMA, damping factor μ is iteratively updated in each step. In each step, $\boldsymbol{p}^k$ is obtained by solving,

$$(\boldsymbol{J}^{k^T}\boldsymbol{J}^k + \mu\boldsymbol{D})\boldsymbol{p}^k = -\boldsymbol{g}^k = -\boldsymbol{J}^{k^T}\boldsymbol{F}^k \ . \tag{3.55}$$

In GNA, Δ^k is iteratively updated in each step. In each step, $\boldsymbol{p}^k$ is obtained by solving,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + \boldsymbol{g}^{k^T}\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}^k\boldsymbol{p} \quad \text{s.t. } \|\boldsymbol{p}\| \leq \Delta^k \ . \tag{3.56}$$

The minimizer is either reached with,

$$(\boldsymbol{J}^{k^T}\boldsymbol{J}^k)\boldsymbol{p}^k = -\boldsymbol{J}^{k^T}\boldsymbol{F}^k \ , \text{ if } \|\boldsymbol{p}^k\| \leq \Delta^k \text{ and } \boldsymbol{J}^{k^T}\boldsymbol{J}^k \text{ positive definite}, \tag{3.57}$$

or on the boundary of the trust region.
In LMA, a damping factor μ is added in the diagonal of the original Hessian matrix. Therefore, a new redefined Hessian matrix with damping factor, $\boldsymbol{H}_{\boldsymbol{\mu}}$, is used,

$$\boldsymbol{H}_{\boldsymbol{\mu}} = \boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{D} \ . \tag{3.58}$$

Still, $\boldsymbol{H}_{\boldsymbol{\mu}}$ can also be presented in the sub-matrix form similar as (3.45),

$$\boldsymbol{H}_{\boldsymbol{\mu}} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^T & \boldsymbol{C} \end{pmatrix} \ . \tag{3.59}$$

In GNA, the Hessian matrix $\boldsymbol{H}$ has the same value and structure as damped Hessian matrix $\boldsymbol{H}_{\boldsymbol{\mu}}$ in LMA, except for diagonal elements, seeing (3.45). Thus, some GNA versions can be directly obtained by replacing $\boldsymbol{H}_{\boldsymbol{\mu}}$ with $\boldsymbol{H}$. Only $\boldsymbol{B}$ and $\boldsymbol{C}$ are adjusted from the original form with damping factor.

3.2.1 Schur Complement

In bundle adjustment, the equation system is always exceed 10000-dimension. Direct solving (3.55) is a pretty computation demanding task. In [Hay68], they propose a trick in solving large linear system of equations (LSE), i.e. Schur complement (SC). About more details of Schur complement, please see [Zha06]. Here, we only discuss the implementation of Schur complement trick in bundle adjustment.
Like we have talked before, both LMA and GNA have a same sub-matrix form of (damped) Hessian matrix according to the camera parameter and the feature position parameters,

$$\boldsymbol{H_\mu} \text{ or } \boldsymbol{H} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^T & \boldsymbol{C} \end{pmatrix} \ . \tag{3.60}$$

The gradient vector $\boldsymbol{g}$ and the increment step vector (solution) $\boldsymbol{p}$ in (3.55) and in (3.57) can also split up to the camera related part and the feature point related part as,

$$\begin{pmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^T & \boldsymbol{C} \end{pmatrix} \begin{pmatrix} \boldsymbol{p_c} \\ \boldsymbol{p_p} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{g_c} \\ \boldsymbol{g_p} \end{pmatrix} \ , \tag{3.61}$$

$$\boldsymbol{p} = \begin{pmatrix} \boldsymbol{p_c} \\ \boldsymbol{p_p} \end{pmatrix} \ , \tag{3.62}$$

$$\boldsymbol{g} = \begin{pmatrix} \boldsymbol{g_c} \\ \boldsymbol{g_p} \end{pmatrix} \ , \tag{3.63}$$

where $\boldsymbol{p_c}$ and $\boldsymbol{g_c}$ are of the dimension of 9n $\times$ 1; $\boldsymbol{p_p}$ and $\boldsymbol{g_p}$ are of the dimension of 3m $\times$ 1.
If $\boldsymbol{C}$ is full rank, instead of solving (3.55), the equation system is rewritten as,

$$(\boldsymbol{B} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^T)\boldsymbol{p_c} = -\boldsymbol{g_c} + \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{g_p} \ , \tag{3.64}$$

$$\boldsymbol{S_C} = \boldsymbol{B} - \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^T \ , \tag{3.65}$$

$$\boldsymbol{v} = -(-\boldsymbol{g_c} + \boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{g_p}) \ , \tag{3.66}$$

which is the Schur Complement of $\boldsymbol{C}$, also known as the reduced camera matrix [ASSS10]. $\boldsymbol{S_C}$ is a block structured symmetric positive definite matrix, with block size of 9 $\times$ 9. The block $\boldsymbol{S_{C,ij}}$ corresponding to the pair of images frame i and j, is non-zero if and only if the two images observe at least one common point [ASSS10].
After solving (3.64), $\boldsymbol{p_p}$ is obtained through,

$$\boldsymbol{p_p} = \boldsymbol{C}^{-1}(-\boldsymbol{g_p} - \boldsymbol{E}^T\boldsymbol{p_c}) \ . \tag{3.67}$$

The original solving s $\times$ s linear equation system problem is converted into inversion of block diagonal matrix $\boldsymbol{C}$, some matrix-vector multiplications, and solving 9n $\times$ 9n linear equation system (3.64).

$\boldsymbol{C}$ is a block diagonal matrix with block size of 3×3,

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{C_1} & & & \\ & \boldsymbol{C_2} & & \\ & & \ddots & \\ & & & \boldsymbol{C_m} \end{pmatrix} . \tag{3.68}$$

The inversion of $\boldsymbol{C}$ is relatively a cheap process, $O(\mathrm{m})$ algorithm, with

$$\boldsymbol{C}^{-1} = \begin{pmatrix} \boldsymbol{C_1}^{-1} & & & \\ & \boldsymbol{C_2}^{-1} & & \\ & & \ddots & \\ & & & \boldsymbol{C_m}^{-1} \end{pmatrix} . \tag{3.69}$$

On the other hand, if $\boldsymbol{B}$ is full rank, instead of solving (3.55) or (3.57), the equation system can be also rewritten as,

$$(\boldsymbol{C} - \boldsymbol{E}^T\boldsymbol{B}^{-1}\boldsymbol{E})\boldsymbol{p_p} = -\boldsymbol{g_p} + \boldsymbol{E}^T\boldsymbol{B}^{-1}\boldsymbol{g_c} , \tag{3.70}$$

$$\boldsymbol{S_B} = \boldsymbol{C} - \boldsymbol{E}^T\boldsymbol{B}^{-1}\boldsymbol{E} , \tag{3.71}$$

$$\boldsymbol{w} = -(-\boldsymbol{g_p} + \boldsymbol{E}^T\boldsymbol{B}^{-1}\boldsymbol{g_c}) , \tag{3.72}$$

which is the Schur complement of $\boldsymbol{B}$, i.e. the reduced feature points matrix, similar as Schur complement of $\boldsymbol{C}$.
After solving (3.70), $\boldsymbol{p_p}$ is obtained through,

$$\boldsymbol{p_c} = \boldsymbol{B}^{-1}(-\boldsymbol{g_c} - \boldsymbol{E}\boldsymbol{p_p}) . \tag{3.73}$$

The original solving s × s linear equation system problem is converted into inversion of block diagonal matrix $\boldsymbol{B}$, some matrix-vector multiplications, and solving 3m×3m linear equation system (3.70).
In general bundle adjustment problem, the number of image frames (cameras) is much smaller than the number of feature points, so that $9\mathrm{n} \lll 3\mathrm{m}$. Therefore, the complexity of solving (3.64) is far cheaper than solving (3.70).
Even if in SC of $\boldsymbol{B}$, the matrix to invert has a smaller dimension than in SC of $\boldsymbol{C}$, the inversion of matrix is cubic complexity computation, i.e. in Schur complement of $\boldsymbol{B}$ it is 9^3n and in Schur complement of $\boldsymbol{C}$ 3^3m. Anyway, generally, the inversion of $\boldsymbol{B}$ is cheaper than the inversion of $\boldsymbol{C}$. However, the complexity in the part of solving linear equation system occupies the major computation in comparison with inversion part, no matter which solvers are used. The most widely used solvers in bundle adjustment are decomposition and conjugates gradient, which are discussed in the following sections.

3.2.2 Cholesky Decomposition

Decomposition, such as LU decomposition, singular value decomposition, Cholesky decomposition, is the basic idea for solving linear system of equations. For good conditioned, positive definite and relatively small matrix, decomposition is still the most effective method to get an exact solution [NW06]. Since the system matrix of LMA in bundle adjustment, i.e. $\boldsymbol{H_\mu}$, $\boldsymbol{S_B}$, $\boldsymbol{S_C}$, are Hermitian, positive-definite matrix, Cholesky decomposition is most widely used to solve the equation system of LMA in small-scaled bundle adjustment.

Theorem 3.2.1 (Cholesky Decomposition) $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^*$, *where* $\boldsymbol{L}$ *is a lower triangular matrix with real and positive diagonal entries, and* $\boldsymbol{L}^*$ *denotes the conjugate transpose of* $\boldsymbol{L}$. *Every Hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition [wikb].*

In GNA, due to without damping factor, the Hessian matrix $\boldsymbol{H}$ can not be guaranteed strictly positive definite (may close to singular), so that using Cholesky decomposition in GNA not only costs extremely large computation time, but also leads to extremely inaccurate solution. For example, we implement Cholesky decomposition in GNA coupled with Schur complement of $\boldsymbol{C}$ in Matlab, and test the algorithm on Ladybug dataset proposed in Section 2.3. After around 20 iteration steps, the condition number of $\boldsymbol{S_C}$ is over 10^{16}.
Therefore, we only test Cholesky decomposition in LMA. We implement three variants in LMA, with SC of $\boldsymbol{C}$, with SC of $\boldsymbol{B}$, and without SC. All linear equation system solvers is realized with Cholesky decomposition (CD) in Matlab. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. For each implementation, we run 50 iteration steps. The computation time in each CD, and the time costing in each iteration step are recorded, since in each iteration step there may exist more than one CD to get a successful reduction in the objective function (more details in Section 3.3.3). The time spent in each CD is counted from forming the corresponding equation system to obtaining $\boldsymbol{p}^k$. Besides, the objective function and the gradient norm after each iteration step are also collected.
The objective function and the gradient norm vary along the time are demonstrated in Figure 3.4 and Figure 3.5.
The processing time of three CD implementations is presented in Table 3.2.

Table 3.2: Processing time comparison of CD with different SC implementations. SC of $\boldsymbol{C}$ has the highest efficiency.

Implementation	no SC ($\boldsymbol{H_\mu}$)	SC of $\boldsymbol{B}$	SC of $\boldsymbol{C}$
Time per CD/s	5.88	34.25	4.73
Time in 50 iteration/s	709	> 5000	365

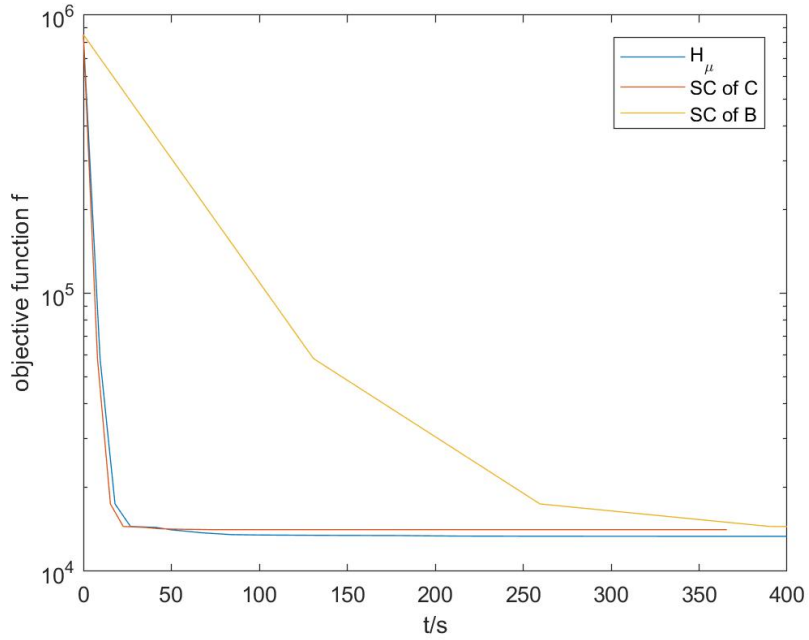

Figure 3.4: Comparison of the objective function curve between different Schur complement in LMA solved by Cholesky decomposition.

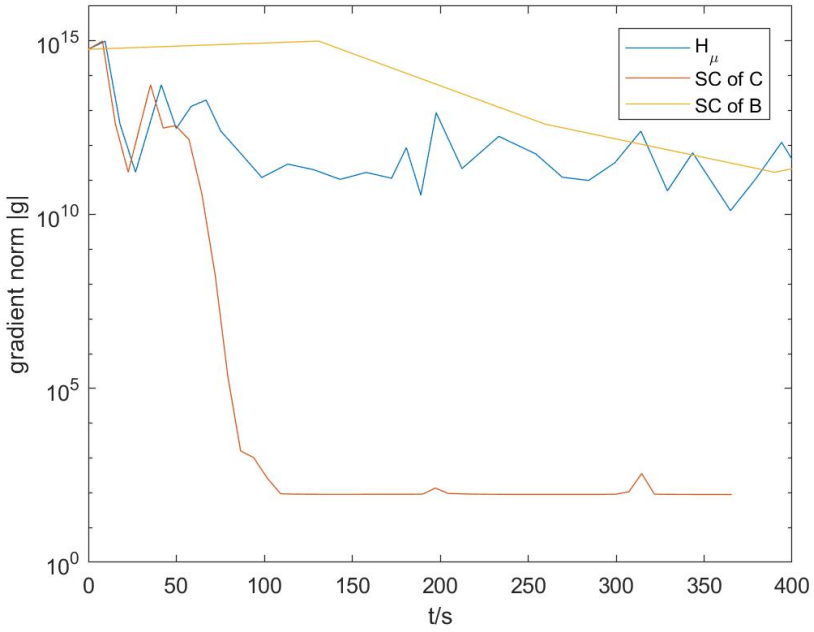

Figure 3.5: Comparison of the gradient norm curve between different Schur complement in LMA solved by Cholesky decomposition. The implementation with SC of $\boldsymbol{C}$ has the best performance, and stands for "LMA-CD" in the following.

With SC of $\boldsymbol{C}$, a promotion in processing time is obtained in solving LSE in bundle adjustment. However, the processing time with SC of $\boldsymbol{B}$ is even much larger than without SC. The reason is that the feature points arguments are far more (almost 50 times) than the camera arguments in our dataset, so that $\boldsymbol{S_B}$ has a similar dimension with $\boldsymbol{H_\mu}$, besides, explicit constructing $\boldsymbol{S_B}$ also costs lots of time.

In optimization, the curve of the objective function demonstrates how fast and how deep the objective function reduces along the optimization process; the curve of the gradient norm reveal how fast and how close the solution approaches a local optimal point. Since bundle adjustment is a highly non-convex, non-linear optimization problem, even if the objective function maintains almost stable after several iteration steps, it is still not enough to prove it is a local optimal. In bundle adjustment, lots of points may have the similar value of the objective function, although these points locate far apart from each other. Only when the gradient norm is small enough, this resolved point is considered as a local optimal.

From Figure 3.4, it is summarized that both SC of $\boldsymbol{C}$ and without SC have a similar and fast reduction curve of the objective function related to SC of $\boldsymbol{B}$. From Figure 3.5, it is obvious that SC of $\boldsymbol{C}$ gains a significant decreasing of the gradient norm after several iteration steps. Therefore, LMA coupled with SC of $\boldsymbol{C}$ solved by CD is more efficiency than other two CD implementations.

LMA coupled with SC of $\boldsymbol{C}$ solved by CD is still the most efficient algorithm in small-scale bundle adjustment [ASSS10]. However, CD solver also faces challenges even if in small-scale bundle adjustment. CD solver is not robust when the matrix of equation system is bad conditioned or near singular, which leads to an extreme large computation time and huge error. Thus, even if using CD in LMA, the damping factor (damping matrix) needs to be carefully selected to ensure the matrix is good conditioned and far from singular. For example, if the damping matrix strategy (3.74) proposed in [ASSS10] is implemented in LMA coupled with SC, CD solver generates NaN and/or Inf element after several iterations. Same situation also happens sometimes when the updating process of damping factor is changed.

$$\boldsymbol{H_\mu} = \boldsymbol{J}^{k^T}\boldsymbol{J}^k + \mu \mathrm{diag}(\boldsymbol{J}^{k^T}\boldsymbol{J}^k) \ . \tag{3.74}$$

For solving large-scale dense bundle adjustment, CD is not efficient anymore. Decomposition of large matrix often costs much more time than using other strategy to solving the equation system, e.g. preconditioned conjugate gradient, seeing more details in Section 5.3. Moreover, in the later implementation with conjugate gradient, the explicit expression of $\boldsymbol{S_C}$ and $\boldsymbol{S_B}$ can be avoided through matrix-vector multiplication, which also save much processing time in solving equation system [ASSS10]. Besides, the avoiding of explicit expression of $\boldsymbol{S_C}$ and $\boldsymbol{S_B}$ also saves lots of memory in comparison with the solver without Schur complement.

3.2.3 Baseline Implementations

Like we have talked in Section 1.2, we select LMA coupled with SC of $\boldsymbol{C}$ solved by CD and the algorithm proposed in [ASSS10] as the baseline implementations in the thesis. Moreover, in the following thesis, these both baselines are called as "LMA-CD" and "BAL" for short.
A comparison of the objective function and the gradient norm between both baseline implementations is plotted in Figure 3.6 and Figure 3.7.

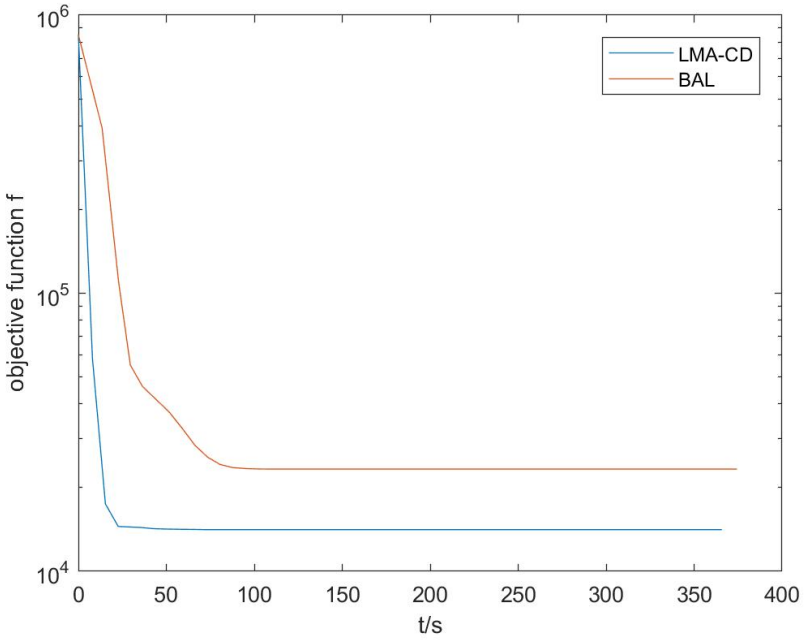

Figure 3.6: Comparison of the objective function curves between both baseline implementations. "LMA-CD" performs better in the objective function reduction on the Ladybug dataset.

3.3 Damped Inexact Newton's Method

The common method of solving linear equation system $\boldsymbol{Ax} = \boldsymbol{b}$ is Cholesky decomposition, if $\boldsymbol{A}$ is a Hermitian, positive-definite matrix. In bundle adjustment, since $\boldsymbol{S_C}$, $\boldsymbol{S_B}$, and $\boldsymbol{H_\mu}$ have a sparse structure. Except for direct using Cholesky decomposition [TMHF99], Chen et al use row and column re-ordering algorithms to maximize the sparsity of the Cholesky decomposition, and focus compute effort on the non-zero part of the factorization [CDHR08], [ASSS10].
However, $\boldsymbol{S_C}$ and $\boldsymbol{S_B}$ have a large dimension in the large-scale dataset, or $\boldsymbol{S_C}$ and $\boldsymbol{S_B}$ are not sparse enough, such as, community photo collections, so that explicit construction of both matrices is already pretty computation and storage demanding.

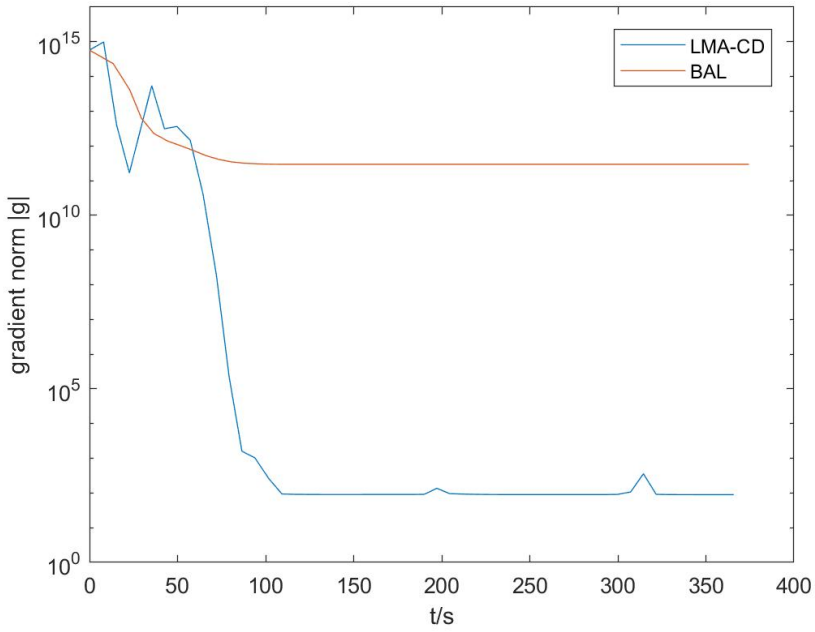

Figure 3.7: Comparison of the gradient norm curves between both baseline implementations. "LMA-CD" performs better in the gradient reduction on the Ladybug dataset.

Besides, Cholesky decomposition is not robust when facing bad conditioned or near singular matrix.

We propose to use conjugate gradient (CG) to solve the linear equation system. There are two widely used fundamental optimization algorithm with CG. Using CG to solve the equation system in Gauss Newton Algorithm with trust region, is called inexact Newton's method; using CG to solve the equation system in Levenberg-Marquadt Algorithm, is called truncated Newton's method [ASSS10].

In this section, we firstly introduce these two methods, and present our important improvements in both methods. Some of these developed strategies are also supposed to be used in the following algorithm as well, such as, trust region radius updating, PCG solver with scaled trust region, damping factor updating, etc. Due to the unsatisfied performance in both methods, then, we presented an inexact Newton's method with damping factor in Hessian matrix, where Levenberg-Marquadt Algorithm is bounded by a explicit scaled trust region in bundle adjustment. This algorithm combines the advantages from both truncated Newton's method and inexact Newton's method, which yields a significant advancement.

3.3.1 Conjugate Gradient

Conjugate gradient is a useful technique for solving large linear system of equations [NW06], e.g.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \ . \tag{3.75}$$

In creating a new vector $\boldsymbol{a}_j$ in the conjugate gradient set, it only needs the previous vector $\boldsymbol{a}_{j-1}$. The new vector is automatically conjugate to all vectors in the set. This strategy requires little storage and computation, which is particularly suitable for solving large equation system. Every conjugate direction $\boldsymbol{a}_j$ is chosen to be a linear combination of the current negative residual $-\boldsymbol{r}_j$ and previous direction $\boldsymbol{a}_{j-1}$.

$$\boldsymbol{r} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \ , \tag{3.76}$$

which is the steepest descent direction of the objective function,

$$\psi_{\boldsymbol{A}} = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} \ . \tag{3.77}$$

The conjugate gradient process actually iteratively minimize the objective function (3.77), until finding a satisfied solution $\boldsymbol{x}_j$.

$$\boldsymbol{a}_j = -\boldsymbol{r}_j + \beta_j\boldsymbol{a}_{j-1} \ . \tag{3.78}$$

β is to ensure that $\boldsymbol{a}_j$ and $\boldsymbol{a}_{j-1}$ are conjugate with respect to $\boldsymbol{A}$, i.e.

$$\boldsymbol{a}_{j-1}^T\boldsymbol{A}\boldsymbol{a}_j = 0 \ , \tag{3.79}$$

so that,

$$\beta_j = \frac{\boldsymbol{r}_j^T\boldsymbol{A}\boldsymbol{a}_{j-1}}{\boldsymbol{a}_{j-1}^T\boldsymbol{A}\boldsymbol{a}_{j-1}} \ . \tag{3.80}$$

$\boldsymbol{x}$ is updated with the direction of conjugate vector in each iteration as,

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j\boldsymbol{a}_j \ , \tag{3.81}$$

where the step length α_j is obtained with,

$$\alpha_j = \frac{\boldsymbol{r}_j^T\boldsymbol{a}_j}{\boldsymbol{a}_j^T\boldsymbol{A}\boldsymbol{a}_j} \ . \tag{3.82}$$

The original conjugate gradient algorithm please see [NW06]. The convergence rate of conjugate gradient (CG) is so slower than solving the equation system with Newton's Method, since in general we need to update $\boldsymbol{x}$ with each CG vector in the set. The number of CG vectors equals to the dimension of $\boldsymbol{A}$.
However, using CG to solve a precise solution of step is inefficient and also unnecessary in bundle adjustment. A termination condition is set for the iteration loop of CG, i.e. if the norm of residual vector $\boldsymbol{r}_j$ is smaller than a threshold, the CG loop

stops and $\boldsymbol{x}$ is set as current value in j-th iteration. In this thesis, the terminate condition of CG loop is chosen as $\|\boldsymbol{r}_j\| < 0.1\|\boldsymbol{b}\|$.
Firstly, we implement the CG algorithm for Schur complement of $\boldsymbol{C}$, and comparing the processing time of CG with Cholesky decomposition and LU decomposition for SC of $\boldsymbol{C}$. All three implementations solve (3.64) based on sub-dataset, and then form the complete argument space $\boldsymbol{p}$ with $\boldsymbol{p_c}$ and $\boldsymbol{p_p}$ from (3.67). The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM.
These three implementations are respectively used to solve 50 LSEs of Schur complement. The average processing time for solving these LSEs is demonstrated in Table 3.3.

Table 3.3: Processing time comparison between solving LSEs (SC of $\boldsymbol{C}$) with CG and decompositions. CG solver is the fastest.

Implementation	CG	Cholesky Decomposition	LU Decomposition
Time per LSE/s	0.322	0.338	0.348

In general, Cholesky decomposition is much faster than LU decomposition for Hermitian, positive-definite matrix. However, in our implementations, their processing time in Table 3.3 is almost the same. Because of round-off error problem in numerical calculation, $\boldsymbol{S_C}$ is not a strict Hermitian, positive-definite matrix. Before using Cholesky decomposition, $\boldsymbol{S_C}$ is set as $(\boldsymbol{S_C} + \boldsymbol{S_C^T})/2$ to compensate the error.
The norm of the difference between the solution of CG and the solution from decompositions is computed to demonstrate the accuracy of CG, e.g. for Cholesky decomposition (CD),

$$\frac{\|\boldsymbol{p}_{CG} - \boldsymbol{p}_{CD}\|^2}{\mathrm{s}} \,. \tag{3.83}$$

The results about the accuracy of PCG show below in Table 3.4.

Table 3.4: Accuracy comparison between different solvers (with SC of $\boldsymbol{C}$)

Comparison pair	CG and CD	CG and LUD	CD and LUD
Difference Norm/1	0.010	0.010	0.000

From the results in Table 3.3 and Table 3.4, we summarize that for a small-scale LSE, CG does not reduce the processing time significantly, on the contrary, CG does not solve LSE exactly as Cholesky decomposition and LU decomposition.
However, a significant advantage of CG in comparison with decomposition is that CG is much more robust in solving LSE even if the system matrix $\boldsymbol{A}$ is bad conditioned or near singular. The strategy of CG is approximating the solution iteratively through the conjugate gradient vector of $\boldsymbol{A}$, which is not affected so much by the intrinsic property of $\boldsymbol{A}$ as decomposition. When $\boldsymbol{A}$ is bad conditioned, it only takes more CG steps in approximating; when $\boldsymbol{A}$ is near singular, CG solver does not crash as decomposition.

3.3.2 Preconditioned Conjugate Gradient

Since the system matrix in bundle adjustment, $\boldsymbol{H}$, $\boldsymbol{S_C}$, or $\boldsymbol{S_B}$, is often bad conditioned, the CG loop can be further accelerated through improving the eigenvalue distribution of system matrix. When the condition number is reduced, the termination condition is reached with less CG steps, i.e. the CG loop terminates faster.
Preconditioned Conjugate Gradient (PCG) reforms the equation system with a preconditioner matrix $\boldsymbol{P}$. $\boldsymbol{P}$ can be stable or varied in each iteration steps. With preconditioner, the condition number of the system matrix is reduced. The selection of preconditioner is discussed later in Section 3.3.5.
With $\boldsymbol{P}$, instead of solving $\boldsymbol{x}$ in $\boldsymbol{Ax} = \boldsymbol{b}$, we solve new vector $\breve{\boldsymbol{x}}$,

$$\breve{\boldsymbol{x}} = \boldsymbol{P}\boldsymbol{x} \ , \tag{3.84}$$

in the equation system,

$$(\boldsymbol{P}^{-T}\boldsymbol{A}\boldsymbol{P}^{-1})\breve{\boldsymbol{x}} = \boldsymbol{P}^{-T}\boldsymbol{b} \ . \tag{3.85}$$

After obtaining $\breve{\boldsymbol{x}}$, the original vector $\boldsymbol{x}$ is acquired with $\boldsymbol{x} = \boldsymbol{P}^{-1}\breve{\boldsymbol{x}}$.
Similar with previous CG, PCG finds a point $\breve{\boldsymbol{x}}$ which minimizes the objective function,

$$\breve{\psi}_{\boldsymbol{A}} = \frac{1}{2}\breve{\boldsymbol{x}}^T\breve{\boldsymbol{A}}\breve{\boldsymbol{x}} - \breve{\boldsymbol{b}}^T\breve{\boldsymbol{x}} \ , \tag{3.86}$$

with

$$\breve{\boldsymbol{A}} = \boldsymbol{P}^{-T}\boldsymbol{A}\boldsymbol{P}^{-1} \ , \tag{3.87}$$

$$\breve{\boldsymbol{b}} = \boldsymbol{P}^{-T}\boldsymbol{b} \ . \tag{3.88}$$

After simplifying (3.86), obviously, $\breve{\psi}_{\boldsymbol{A}}$ equals to $\psi_{\boldsymbol{A}}$ in (3.77), which means the objective function does not change with preconditioner.
In PCG, the preconditioner matrix $\boldsymbol{M}$ is usually used, which is defined as $\boldsymbol{M} = \boldsymbol{P}^T\boldsymbol{P}$. In the following thesis, when we mention preconditioner matrix, it represents $\boldsymbol{M}$. $\boldsymbol{M}$ is chosen to be a highly sparse, diagonal or block diagonal square matrix with the same size as $\boldsymbol{A}$. Here, the general PCG form used for solving $\boldsymbol{Ax} = \boldsymbol{b}$ with the preconditioner $\boldsymbol{M}$ is briefly described in Algorithm 1.
Using CG to solving an inexact increment step in LMA (3.55) or in GNA 3.57, indicates a optimization of the objective function based on a quadratic model function around the current point. The preconditioner in CG and the coupled Schur complement trick only having a acceleration impact on solving LSE.

3.3.3 Inexact Newton's Method

Here, we introduce our improved inexact Newton's method (INM) systematically. In inexact Newton's method, a (scaled) trust region is combined with CG solver to restrict that the step size of each increment, such that the quadratic model function in GNA better approximates the behavior of the objective function in a proper area. according to how well this approximation is, the trust region radius for the

Algorithm 1 PCG

Require: Equation system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$; Preconditioner $\boldsymbol{M}$; Maximum iterative steps in PCG loop, $\#_{PCG}$; Initial $\boldsymbol{x}_0$;

Ensure: Final result $\boldsymbol{x}_j$;

Initialize residual $\boldsymbol{r}_0 \leftarrow \boldsymbol{b}$;
Set termination condition $\epsilon \leftarrow min(0.1, \|\boldsymbol{b}\|) * \|\boldsymbol{b}\|$;
Solve $\boldsymbol{M}\boldsymbol{q}_0 = \boldsymbol{r}_0$ for $\boldsymbol{q}_0$;
Initialize conjugate gradient $\boldsymbol{a}_0 \leftarrow -\boldsymbol{q}_0$;
$j \leftarrow 0$;
while $j < \#_{PCG}$ **and** $\|\boldsymbol{r}_j\| > \epsilon$ **do**
    $\alpha_j \leftarrow \frac{\boldsymbol{r}_j^T \boldsymbol{q}_j}{\boldsymbol{a}_j^T \boldsymbol{A} \boldsymbol{a}_j}$;
    $\boldsymbol{x}_{j+1} \leftarrow \boldsymbol{x}_j + \alpha_j \boldsymbol{a}_j$;
    $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j + \alpha_j \boldsymbol{A} \boldsymbol{a}_j$;
    Solve $\boldsymbol{M}\boldsymbol{q}_{j+1} = \boldsymbol{r}_{j+1}$ for $\boldsymbol{q}_{j+1}$;
    $\beta_{j+1} \leftarrow \frac{\boldsymbol{r}_{j+1}^T \boldsymbol{q}_{j+1}}{\boldsymbol{r}_j^T \boldsymbol{q}_j}$;
    $\boldsymbol{a}_{j+1} \leftarrow -\boldsymbol{q}_{j+1} + \beta_{j+1} \boldsymbol{a}_j$;
    $j \leftarrow j + 1$;
end while
return $\boldsymbol{x}_j$;

next iteration is updated. This explicit region boundary strategy robustly define a restriction area in each iteration related to the implicit step size in LMA such that the increment step is large enough, on the other hand, the approximation is precise enough.

In inexact Newton's method, an optimal increment step in each iteration step is obtained by solving (3.89),

$$\boldsymbol{p}^k = \min_{\boldsymbol{p} \in \mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + {\boldsymbol{g}^k}^T \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^T \boldsymbol{H}^k \boldsymbol{p} \quad \text{s.t. } \|\boldsymbol{p}\| \leq \Delta^k \ . \tag{3.89}$$

Trust Region Radius Updating

Firstly, the updating rule of the trust region radius, Δ^k, must be confirmed. The most commonly used updating rule is built according to a ratio,

$$\rho^k = \frac{\psi_f(\boldsymbol{u}^k) - \psi_f(\boldsymbol{u}^k + \boldsymbol{p}^k)}{m_f^k(\boldsymbol{0}) - m_f^k(\boldsymbol{p}^k)} \ . \tag{3.90}$$

For the definition of ψ_f and m_f^k please see (2.26) and (3.89). This ratio presents how well the model function m_f^k approximates the real objective function ψ_f around current point $\boldsymbol{u}^k$. If it is near 1, which means the approximation is well, the trust region radius can be increased; if it is near 0, which means the approximation is bad, the trust region radius should be decreased. If it is smaller than 0, which means the

real function is increased with this increment step, this step is rejected. In bundle adjustment, $\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$, i.e. Hessian matrix is always assumed to be semi-positive definite such that the denominator of ρ^k is greater equal 0 in analytical. Then we can choose a smaller trust region, seeing Section 2.2.1. However, due to some numerical error in bad conditioned situation, the denominator of ρ^k may also be smaller than zero, which causes catastrophic problem if the algorithm is not robust enough.
In this thesis, we develop a novel trust region radius updating process calibrated from [NW06], seeing Algorithm 2.

Algorithm 2 Trust Region Radius Updating

Require: Increment step $\boldsymbol{p}^k$; Ratio ρ^k; Current radius Δ^k;
Ensure: Next radius Δ^{k+1};
1: **if** $\rho^k < 0.25$ **then**
2: $\quad\Delta^{k+1} \leftarrow 0.25\Delta^k$;
3: **else**
4: $\quad$**if** $\rho^k > 0.75$ **and** $\|\boldsymbol{p}^k\| = \Delta^k$ **then**
5: $\quad\quad\Delta^{k+1} \leftarrow 2\Delta^k$;
6: $\quad$**else**
7: $\quad\quad\Delta^{k+1} \leftarrow \Delta^k$;
8: $\quad$**end if**
9: **end if**
10: **return** Δ^{k+1};

Trust Region Newton-PCG Method

The arguments vector $\boldsymbol{u}^k$ moves a small step $\boldsymbol{p}^k$ which is the minimum point of the model function (3.89). This quadratic model function is considered as a new local objective function, which models the behavior of the system (the real objective function) around the current point $\boldsymbol{u}^k$, and the optimal step $\boldsymbol{p}^k$ minimize this local objective function in the neighbor area. With this procedure, the real objective function (2.26) is iteratively reduced to a minimum. Obviously, the generated step $\boldsymbol{p}^k$ must be small, since (3.89) only simulates the system in a small neighbor region. But if it is too small, the reduction of the objective funtion is also too small, so that the optimization needs too many steps. The optimal increment step $\boldsymbol{p}^k$ is constrained within a trust region Δ^k, which is actually a hyper ball in argument space. $\boldsymbol{p}^k$ can be approximated by Trust Region Newton-PCG Method, which is also called PCG-Steihaug [NW06].
Here, we present a novel Trust Region Newton-PCG Method in Algorithm 3, which further improves the original Trust Region Newton-PCG Method. Firstly, the termination condition for PCG loop is changed to a more robust condition. Secondly, a more efficient and more precise strategy in searching an approximation solution

on the trust region boundary is proposed, when the solution exceeds the boundary in loop.
Trust Region Newton-PCG Method aims to solve an optimal $\boldsymbol{x}$ in (3.91).

$$\min_{\boldsymbol{x}} \psi_{\boldsymbol{Ab}}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} \quad \text{s.t. } \|\boldsymbol{x}\| \leq \Delta \ , \tag{3.91}$$

which is actually the same presentation of (3.89). Thus, this solver builds a quadratic model function around current point. When needed, the algorithm can also return the reduction of the model function with the resolved point $\boldsymbol{x}$,

$$\psi_{\boldsymbol{Ab}}(\boldsymbol{0}) - \psi_{\boldsymbol{Ab}}(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} \ . \tag{3.92}$$

Algorithm 3 Trust Region Newton-PCG Method

Require: Matrix $\boldsymbol{A}$, vector $\boldsymbol{b}$; Preconditioner $\boldsymbol{M}$; Trust region radius Δ; Maximum iterative steps in PCG loop, $\#_{PCG}$; Termination factor $t_\epsilon \leftarrow 0.1$

Ensure: Final result $\boldsymbol{x}$;

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{0}$;
Initialize residual $\boldsymbol{r}_0 \leftarrow \boldsymbol{b}$;
Set termination condition $\epsilon \leftarrow min(t_\epsilon, \|\boldsymbol{b}\|) * \|\boldsymbol{b}\|$;
Solve $\boldsymbol{M}\boldsymbol{q}_0 = \boldsymbol{r}_0$ for $\boldsymbol{q}_0$;
Initialize conjugate gradient $\boldsymbol{a}_0 \leftarrow -\boldsymbol{q}_0$;
for $j = 0$ to $\#_{PCG}$ **do**
    $\alpha_j \leftarrow \frac{\boldsymbol{r}_j^T\boldsymbol{q}_j}{\boldsymbol{a}_j^T\boldsymbol{A}\boldsymbol{a}_j}$;
    $\boldsymbol{x}_{j+1} \leftarrow \boldsymbol{x}_j + \alpha_j\boldsymbol{a}_j$;
    if $\|\boldsymbol{x}_{j+1}\| \geq \Delta$ **then**
        Find $\tau \geq 0$, such that $\boldsymbol{x} \leftarrow \boldsymbol{x}_j + \tau\boldsymbol{a}_j$ with $\|\boldsymbol{x}\| = \Delta$;
        return $\boldsymbol{x}$;
    end if
    $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{r}_j + \alpha_j\boldsymbol{A}\boldsymbol{a}_j$;
    if $\|\boldsymbol{r}_{j+1}\| \leq \epsilon$ **then**
        $\boldsymbol{x} \leftarrow \boldsymbol{x}_{j+1}$;
        return $\boldsymbol{x}$;
    end if
    Solve $\boldsymbol{M}\boldsymbol{q}_{j+1} = \boldsymbol{r}_{j+1}$ for $\boldsymbol{q}_{j+1}$;
    $\beta_{j+1} \leftarrow \frac{\boldsymbol{r}_{j+1}^T\boldsymbol{q}_{j+1}}{\boldsymbol{r}_j^T\boldsymbol{q}_j}$;
    $\boldsymbol{a}_{j+1} \leftarrow -\boldsymbol{q}_{j+1} + \beta_{j+1}\boldsymbol{a}_j$;
    $j \leftarrow j + 1$;
end for
$\boldsymbol{x} \leftarrow \boldsymbol{x}_{j+1}$;
return $\boldsymbol{x}$;

The calculation process mentioned in Algorithm 3 line 10 is realized with The Law of Cosines as,

$$\|\boldsymbol{x}_j\|^2 + \tau^2\|\boldsymbol{a}_j\|^2 + 2\tau\boldsymbol{x}_j \cdot \boldsymbol{a}_j = \Delta^2 \ . \tag{3.93}$$

Then, τ is computed from a quadratic equation.

Scaling

However, each argument has a different definition about "small" step due to the different sensitivity and different original magnitude order. This phenomenon is essentially coincident with the crucial numerical step chosen in the numerical Jacobian matrix calculation, seeing Section 3.1.4. It leads to that the change in the argument space is much more sensitive in some certain directions than others. An optimization process is poorly scaled if the changes to the arguments $\boldsymbol{x}$ in a certain direction generate much larger variations in the value of the objective function than another directions [NW06].
Bundle adjustment is always poorly scaled, seeing (3.38), e.g. the rotation angles are highly sensitive related to other arguments, and the magnitude of focus length is much larger than others. If the original argument space is used, the obtained increment step vector can not update the current point robustly. Some directions may not even change in several steps. To make the solver more balanced in each direction, we define and solve the optimization problem in a scaled arguments space. A new scaled arguments vector $\widetilde{\boldsymbol{x}}$ is used to replace the original vector $\boldsymbol{x}$. The widely used scaling strategy is diagonal scaling with positive diagonal matrix $\mathbf{D}$.

$$\widetilde{\boldsymbol{x}} = \mathbf{D}\boldsymbol{x} \ . \tag{3.94}$$

Since the shape of trust region should be such that our confidence in the model is more or less the same at all points in the region, the scaled trust region is consider to be an ellipsoid in which the axes are short in the sensitive directions and longer in the less sensitive directions [NW06]. Another advantage of scaling is that when current increment step in PCG loop exceeds the scaled trust region, the loop is also terminated. Then, a proper increment step on the boundary is selected, and the further iterations in PCG are avoided.
In bundle adjustment, this procedure is represented as,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + (\boldsymbol{g}^k)^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}^k\boldsymbol{p} \quad \text{s.t. } \|\mathbf{D}\boldsymbol{p}\| \leq \Delta^k \ . \tag{3.95}$$

The model function can be also written in scaling form,

$$\min_{\widetilde{\boldsymbol{p}}\in\mathbb{R}^s} \widetilde{m}_f^k(\widetilde{\boldsymbol{p}}) = \psi_f^k + (\widetilde{\boldsymbol{g}}^k)^T\widetilde{\boldsymbol{p}} + \frac{1}{2}\widetilde{\boldsymbol{p}}^T\widetilde{\boldsymbol{H}}^k\widetilde{\boldsymbol{p}} \quad \text{s.t. } \|\widetilde{\boldsymbol{p}}\| \leq \Delta^k \ , \tag{3.96}$$

with

$$\widetilde{\boldsymbol{g}}^k = \mathbf{D}^{-1}\boldsymbol{g}^k \ , \tag{3.97}$$

$$\widetilde{\boldsymbol{H}}^k = \mathbf{D}^{-1}\boldsymbol{H}^k\mathbf{D}^{-1} \ , \tag{3.98}$$

$$\widetilde{\boldsymbol{p}} = \mathbf{D}\boldsymbol{p} \ . \tag{3.99}$$

When simplifying (3.96) with (3.97), (3.98), (3.99), (3.96) is proved to be the same as (3.95). Therefore, in the bundle adjustment, we can simple replace (3.95) with (3.96), replace $\widetilde{\boldsymbol{p}}$ with $\boldsymbol{p}$, and also for $\widetilde{\boldsymbol{g}}^k$ and $\widetilde{\boldsymbol{H}}^k$, which yields the coherent optimization process as previous arguments space. Only when it needs the arguments demonstration in the original space sometimes, such as, $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{p}^k$, $\widetilde{\boldsymbol{p}}$ needs to be transformed to $\boldsymbol{p}$.

Coupled with Schur Complement

In this section, Scaled Trust Region Newton-PCG Method is coupled with Schur complement of $\boldsymbol{C}$. The scaling matrix and the trust region radius are constructed according to the camera corresponding increment step $\boldsymbol{p_c}$. When coupled with Schur complement, the optimal increment step directly solved by PCG is not the complete arguments space. When using SC of $\boldsymbol{C}$, the direct resolved increment step is $\boldsymbol{p}_{\boldsymbol{c}}^k$ in k-th iteration,

$$\boldsymbol{S}_{\boldsymbol{C}}^k\boldsymbol{p}_{\boldsymbol{c}}^k = -\boldsymbol{v}^k \ . \tag{3.100}$$

The other variant form, Schur complement of $\boldsymbol{B}$, has the similar representation. Then, $\boldsymbol{p_c}$ is converted to full argument space $\boldsymbol{p}^k$. More details please see Section 3.2.1.
If we consider this real solving process in camera arguments sub-space (Schur complement of $\boldsymbol{C}$), it also contains a implicit quadratic model function in each iteration,

$$\min_{\boldsymbol{p_c}\in\mathbb{R}^{9n}} m_{\boldsymbol{c},f}^k(\boldsymbol{p_c}) = (\boldsymbol{v}^k)^T\boldsymbol{p_c} + \frac{1}{2}\boldsymbol{p_c}^T\boldsymbol{S}_{\boldsymbol{C}}^k\boldsymbol{p_c} \quad \text{s.t. } \|\mathbf{D_c}\boldsymbol{p_c}\| \le \Delta_c^k \ , \tag{3.101}$$

where $\mathbf{D_c}$ is the diagonal scaling matrix for camera arguments; Δ_c^k is the trust region radius used for camera arguments sub-space; $\boldsymbol{v}^k$ and $\boldsymbol{S}_{\boldsymbol{C}}^k$ are presented in (3.65) and (3.66).
Due to the computation process in PCG solver, the value of (3.101) can be directly acquired without further computation. If the decrement of the model function (3.101) can take the place of the decrement of the original model function (3.95), the computational cost in (3.90) reduces further. After our careful computation and verification,

$$m_f^k(\mathbf{0}) - m_f^k(\boldsymbol{p}^k) = -m_f^k(\boldsymbol{p}^k) = -m_{\boldsymbol{c},f}^k(\boldsymbol{p}_{\boldsymbol{c}}^k) + \frac{1}{2}{\boldsymbol{g}_{\boldsymbol{p}}^k}^T{\boldsymbol{C}^{k}}^{-1}\boldsymbol{g}_{\boldsymbol{p}}^k \ , \tag{3.102}$$

with

$$-m_{\boldsymbol{c},f}^k(\boldsymbol{p}_{\boldsymbol{c}}^k) = m_{\boldsymbol{c},f}^k(\mathbf{0}) - m_{\boldsymbol{c},f}^k(\boldsymbol{p}_{\boldsymbol{c}}^k) \ . \tag{3.103}$$

$\boldsymbol{C}^k$ is a sub-matrix in Hessian matrix $\boldsymbol{H}^k$, seeing (3.45). $\boldsymbol{g}_{\boldsymbol{p}}^k$ is the sub-gradient vector corresponding to feature points parameters. Obviously, the second term on

the right side of (3.102) is not dependent on the resolved increment step $\boldsymbol{p}_{\boldsymbol{c}}^k$, i.e. it is a constant in one iteration. Thus, when $\boldsymbol{p}_{\boldsymbol{c}}^k$ is the optimal point in Schur complement, its converted full increment step vector, $\boldsymbol{p}^k$, is still the optimal in original full equation system, which also makes sense that the Schur complement trick can be used instead of solving the full equation system.
Since the inversion of $\boldsymbol{C}^k$ already exists in SC process, (3.102) only needs some extra matrix-vector multiplications, whose computation complexity is far simpler than computing the reduction in the complete arguments space. Therefore,

$$\rho^k = \frac{\psi_f(\boldsymbol{u}^k) - \psi_f(\boldsymbol{u}^k + \boldsymbol{p}^k)}{-m_{\boldsymbol{c},f}^k(\boldsymbol{p}_{\boldsymbol{c}}^k) + \frac{1}{2}\boldsymbol{g}_{\boldsymbol{p}}^{k\,T}\boldsymbol{C}^{k-1}\boldsymbol{g}_{\boldsymbol{p}}^k} \ . \tag{3.104}$$

The Scaled Trust Region Newton-PCG Method coupled with SC of $\boldsymbol{C}$ is presented in Algorithm 4.

Algorithm 4 Scaled Trust Region Newton-PCG with Schur Complement of $\boldsymbol{C}$

Require: Hessian matrix $\boldsymbol{H}$ in form (3.45); Gradient vector $\boldsymbol{g}$ in form (3.63); Scaling matrix $\mathbf{D}_{\mathbf{c}}$;
Ensure: Increment step $\boldsymbol{p}$; Reduction of model function $-m_f(\boldsymbol{p})$;
1: Compute LSE $\boldsymbol{S}_{\boldsymbol{C}}$ and $\boldsymbol{v}$ with (3.65) and (3.66);
2: Scaling $\widetilde{\boldsymbol{S}}_{\boldsymbol{C}} \leftarrow \mathbf{D}_{\mathbf{c}}^{-1}\boldsymbol{S}_{\boldsymbol{C}}\mathbf{D}_{\mathbf{c}}^{-1}$, $\widetilde{\boldsymbol{v}} \leftarrow \mathbf{D}_{\mathbf{c}}^{-1}\boldsymbol{v}$;
3: Construct scaled Schur complement's LSE: $\widetilde{\boldsymbol{S}}_{\boldsymbol{C}}\widetilde{\boldsymbol{p}}_{\boldsymbol{c}} = -\widetilde{\boldsymbol{v}}$
4: Compute preconditioner $\boldsymbol{M}$ with the strategy from Section 3.3.2;
5: Solve $\widetilde{\boldsymbol{S}}_{\boldsymbol{C}}\widetilde{\boldsymbol{p}}_{\boldsymbol{c}} = -\widetilde{\boldsymbol{v}}$ for $\widetilde{\boldsymbol{p}}_{\boldsymbol{c}}$ with Algorithm 3;
6: Reverse to unscaled solution, $\boldsymbol{p}_{\boldsymbol{c}} \leftarrow \mathbf{D}_{\mathbf{c}}^{-1}\widetilde{\boldsymbol{p}}_{\boldsymbol{c}}$
7: Compute the full increment step vector $\boldsymbol{p}$ with (3.67);
8: Compute $-m_f(\boldsymbol{p})$ with (3.102);
9: **return** $\boldsymbol{p}$ and $-m_f(\boldsymbol{p})$;

In order to terminate the iterations when the solution approximates the optimal solution well enough, two termination conditions are added into the process. One is for the norm of the current gradient; the other is for the objective function. Either of both value is reduced down to a specific threshold (ξ_1 and ξ_2), which means the optimal arguments are found.
The complete algorithm of Inexact Newton's Method is described in Algorithm 5. When needed, the INM algorithms for Schur complement of $\boldsymbol{B}$ or without Schur complement can be obtained easily from Algorithm 5 with slight revision.
The matrix $\boldsymbol{S}_{\boldsymbol{C}}$ in LSE of each iteration step does not need to be explicitly computed, since all matrix-matrix multiplication appeared in Algorithm 4 line 5 can be replaced with cascaded matrix-vector multiplication. This strategy not only saves memory space but also saves calculation time, about more details seeing [ASSS10].
Here, We implement altogether three variants of scaling matrix in INM, each with three version of Schur complement (as in CD test). The selected scaling matrices

are identity matrix (no scaling), diagonal matrix formed by the optimal increment step in numerical calculation (3.38), and the diagonal matrix of Hessian matrix (damping matrix proposed in [ASSS10]). Each scaling matrix is regulated according to different versions of SC. The preconditioner is selected as identity matrix (no preconditioner). The maximum iterative steps in PCG loop is chosen to be 1000. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. For each implementation, we run 50 iteration steps. The computation time in each Newton-PCG, and the time costing in each iteration step are recorded, since in each iteration step there may exist more than one Newton-PCG to get a successful reduction in the objective function. One iteration step is terminated until the "update" flag is set to *True*, which indicates it finds a reduction step in this iteration, seeing Algorithm 5 line 13; otherwise, it continue to update the trust region radius, and run another Newton-PCG loop to find a new increment step.

A comparison of the objective function and the gradient norm among different implementations is plotted in Figure 3.8 and Figure 3.9.

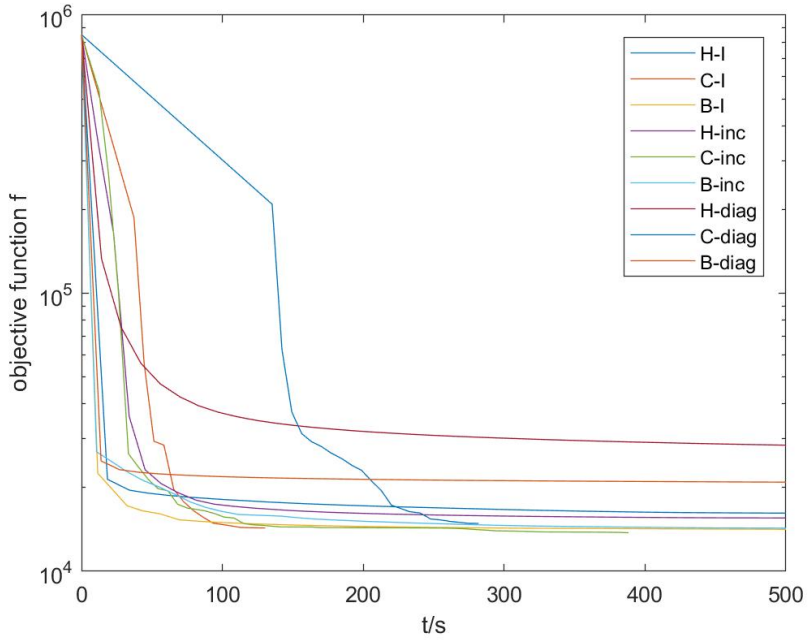

Figure 3.8: Comparison of the objective function curves between different INM implementations. "H" represents without Schur complement; "C" represents Schur complement of $\boldsymbol{C}$; "B" represents Schur complement of $\boldsymbol{B}$. "inc" represents scaling matrix formed from (3.38); "diag" represents scaling matrix formed from the diagonal of Hessian matrix; "I" represents without scaling, i.e. scaling matrix is identity matrix. "C-inc" has the smallest value of the objective function.

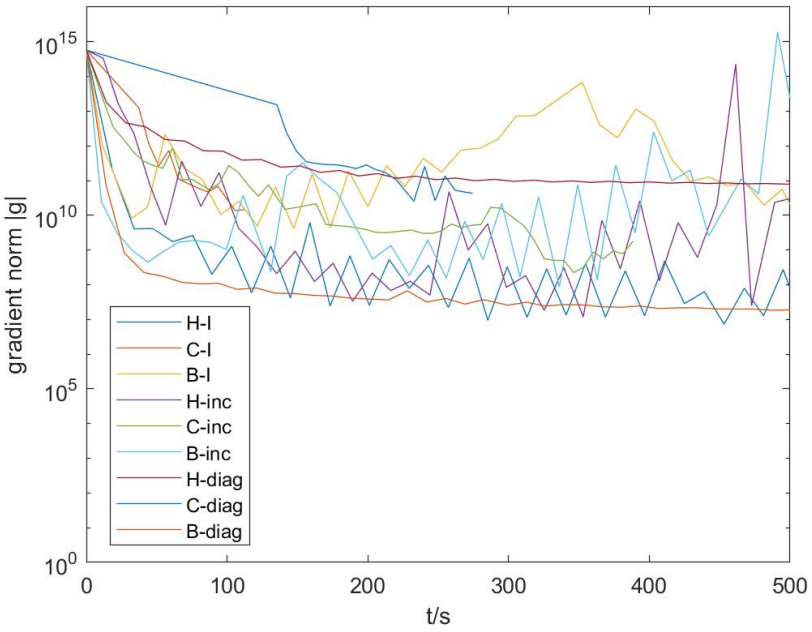

Figure 3.9: Comparison of the gradient norm curve between different INM implementations. "B-diag" performs the best decline curve of the gradient norm.

From the objective function curve figure, almost every implementations obtain a satisfied and similar reduction except for $\boldsymbol{C}$-diag and $\boldsymbol{H}$-diag; in gradient norm curves, $\boldsymbol{B}$-diag implementation has the fastest decreasing. We choose three implementations which yield both good objective function curves and good gradient curves, and present their processing time in Table 3.5. It is no surprise that SC of $\boldsymbol{C}$ has a significant improvement in processing time than others.

Table 3.5: Processing time comparison between INM implementations. "C-inc" is the fastest.

Implementation	$\boldsymbol{H}$-I	$\boldsymbol{C}$-inc	$\boldsymbol{B}$-diag
Time per Newton-PCG/s	11.48	4.37	10.91
Time in 50 iteration/s	692	388	659

3.3.4 Truncated Newton's Method

We also introduce the truncated Newton's method (TNM) briefly. In truncated Newton's method, Levenberg-Marquadt Algorithm is combined with PCG solver. The damping factor in LMA has a similar effectiveness as trust region boundary in INM to restrict the step size of each increment, but in an implicit way. Besides, the damping matrix in LMA also has a similar role as scaling matrix in INM to avoid

poor scaled argument space, but also in an implicit way. According to how well the reduction in quadratic model function (2.53) approximates the reduction of the real objective function, the damping factor (or damping matrix) for the next iteration is updated.
The damping factor updating procedure also varies in different situation. In our algorithm, we use the basic idea provided by [Lou05], and make some small revisions. The damping factor is also updated based on the ratio ρ^k in (3.104) with an auxiliary coefficient ν, seeing Algorithm 6.
Another variant in damping factor part is damping matrix, the original damping matrix is $\mathbf{I}_{s\times s}$, so that when $\mu \to +\infty$, LMA becomes a pure Gradient Descent Algorithm with small step length. In [ASSS10], Agarwal et al. provide a new choice of damping matrix, i.e. diagonal matrix of $\boldsymbol{H}$.
LMA coupled with Schur complement of $\boldsymbol{C}$ solved by PCG is introduced in Algorithm 7. The whole truncated Newton's method is introduced in Algorithm 8.
Here, We implement altogether three variants of damping matrix (similar as scaling matrix in INM) in TNM, each with three version of Schur complement (as in INM test). The preconditioner is selected as identity matrix (no preconditioner). The selected damping matrices are identity matrix, diagonal matrix formed by the optimal increment step in numerical calculation (3.38), and the diagonal matrix of Hessian matrix (as in [ASSS10]). The maximum iterative steps in PCG loop is chosen to be 1000. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. For each implementation, we run 50 iteration steps. The computation time in each LMA-PCG, and the time costing in each iteration step are recorded. Still in each iteration step there may exist more than one LMA-PCG to get a successful reduction in the objective function.
A comparison of the objective function and the gradient norm among different implementations is plotted in Figure 3.10 and Figure 3.11.
From the objective function curve figure, almost every implementation obtains a satisfied and similar reduction; in gradient norm curves, almost every implementation has a huge oscillation. We choose three typical implementations, and present their processing time in Table 3.6. It is no surprise that SC of $\boldsymbol{C}$ has a significant improvement in processing time than others.

Table 3.6: Processing time comparison between TNM implementations. "C-inc" is the fastest.

Implementation	$\boldsymbol{H}$-I	$\boldsymbol{C}$-inc	$\boldsymbol{B}$-inc
Time per LMA-PCG/s	10.78	6.23	10.44
Time in 50 iteration/s	653	678	734

Here, we also compare the performance between INM and TNM. We choose three typical implementations in both methods, $\boldsymbol{H}$-I, $\boldsymbol{C}$-inc, $\boldsymbol{B}$-diag, and plot their objective function curves and gradient norm curves in Figure 3.12 and Figure 3.13.

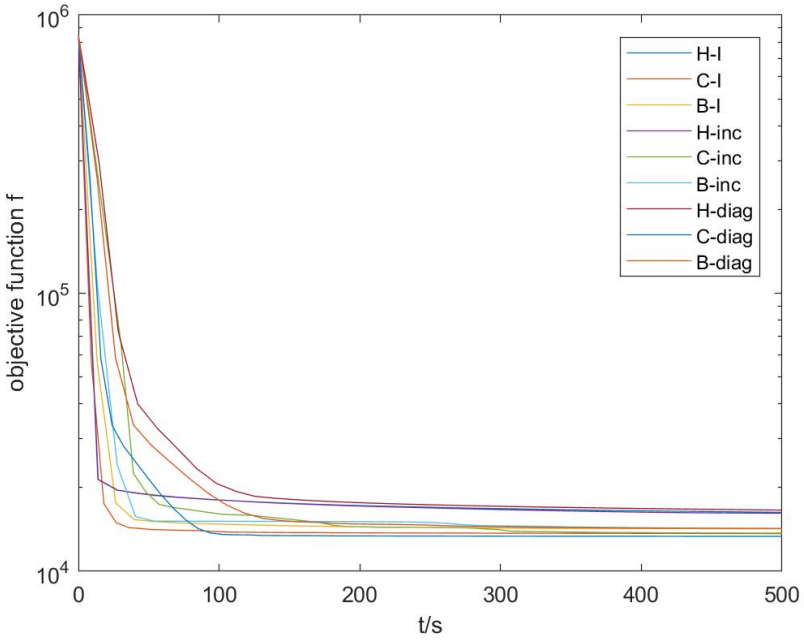

Figure 3.10: Comparison of the objective function curves between different TNM implementations. "H" represents without Schur complement; "C" represents Schur complement of $\boldsymbol{C}$; "B" represents Schur complement of $\boldsymbol{B}$. "inc" represents damping matrix formed from (3.38); "diag" represents damping matrix formed from the diagonal of Hessian matrix; "I" represents with identity damping matrix. "C-diag" has the smallest value of the objective function.

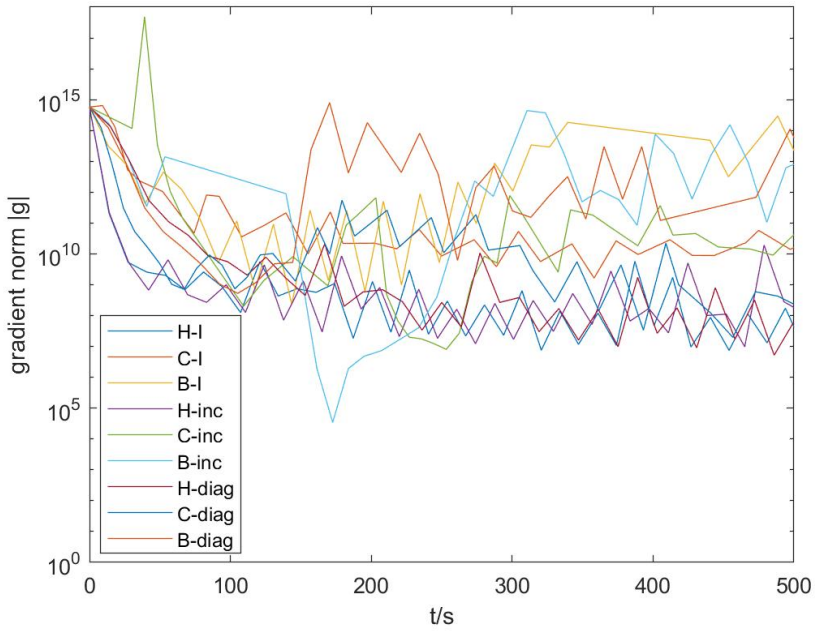

Figure 3.11: Comparison of the gradient norm curves between different TNM implementations. "H-I" has the smallest fluctuation in the gradient reduction.

On the whole, INM implementations have a better gradient norm curve than TNM; however, TNM yields a little faster reduction of the objective function. The reasons come from, that the explicit (scaled) trust region boundary ensure a more robust increment step, so that the optimization process is towards a minimal in near quadratic form; on the other hand, the diagonal element calibration by the damping factor in LMA improves the condition number of Hessian matrix and also avoid the singularity problem, so that PCG solver works better.

In addition, in some implementations of INM (GNA), the Hessian matrix is not strict positive definite anymore and with a large condition number after several iteration steps due to the lack of damping factor, so that the process cannot be executed further, e.g. $\boldsymbol{C}$-diag in INM.

3.3.5 Inexact Newton's Method with Damping

Due to the remarkable advantages in both Newton's methods, we are desired to take the best of them. We combine them together and propose a damped inexact Newton's method (DINM), which adds the damping calibration in Hessian matrix in INM to improve the Hessian matrix. In other words, an explicit (scaled) trust region is implemented in TNM to increase the robustness of step size.

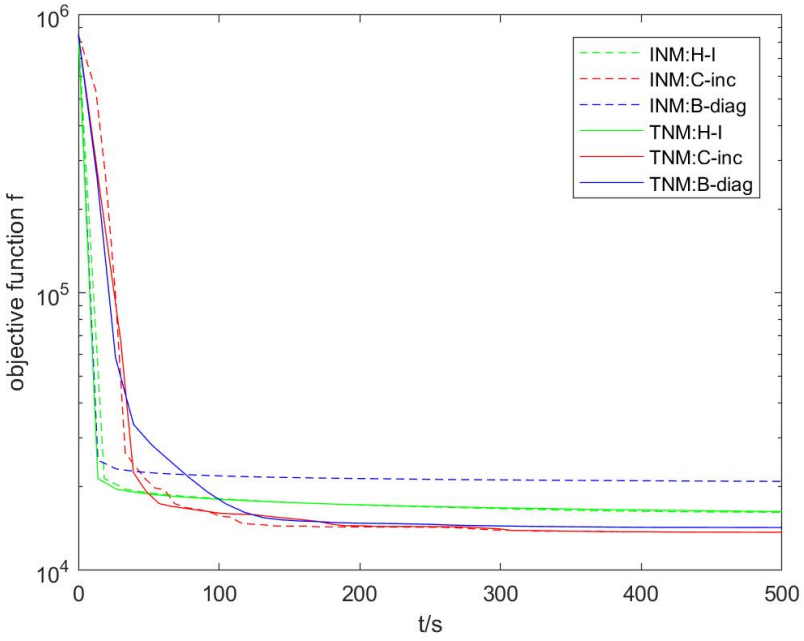

Figure 3.12: Comparison of the objective function curves between INM implementations and TNM implementations. All INM implementations are plotted in dash lines; and their corresponding TNM implementations are plotted in solid lines with the same colors. In general, TNM implementations have a faster and deeper reduction of the objective function related to INMs.

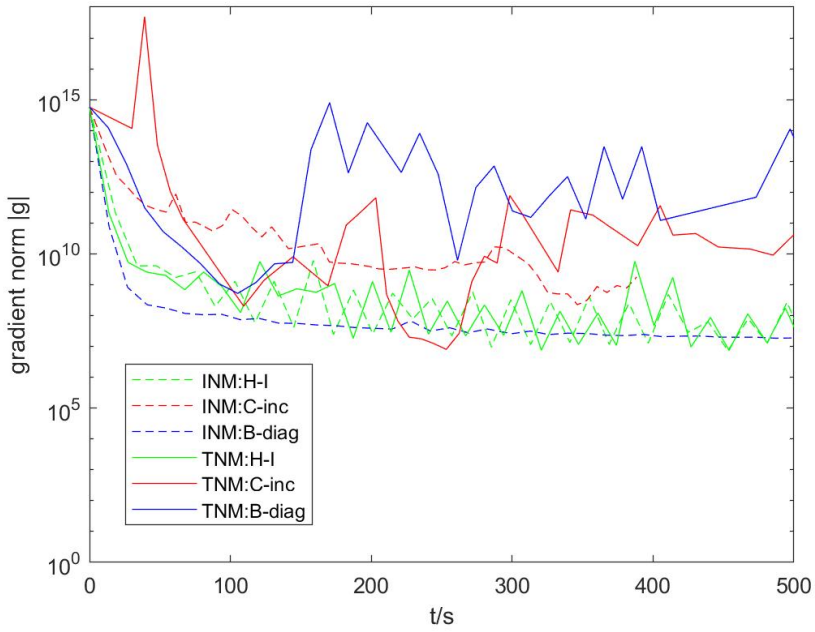

Figure 3.13: Comparison of the gradient norm curves between INM implementations and TNM implementations. In general, INM implementations have a deeper and more stable reduction of the gradient norm related to TNMs.

The optimization process in k-th iteration step in DINM is,

$$\min_{\boldsymbol{p}\in\mathbb{R}^s} m_f^k(\boldsymbol{p}) = \psi_f^k + (\boldsymbol{g}^k)^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}_{\boldsymbol{\mu}}\boldsymbol{p} \quad \text{s.t. } \|\boldsymbol{p}\| \leq \Delta^k \ . \tag{3.105}$$

Since in the previous implementations, Schur complement of $\boldsymbol{C}$ has a significant improvement in processing time and also yields a similar reduction curve related to without SC and SC of $\boldsymbol{B}$. Thus, we only implement our damped inexact Newton's method couple with SC of $\boldsymbol{C}$ in this section. The whole damped inexact Newton's method is presented in Algorithm 9.
Here, We implement altogether four variants of DINM with different scaling matrices and damping matrices (similar as before). The preconditioner is still selected as identity matrix (no preconditioner). The maximum iterative steps in PCG loop is chosen to be 1000. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. For each implementation, we run 50 iteration steps. The computation time in each Newton-PCG, and the time costing in each iteration step are recorded. A comparison of the objective function and the gradient norm among different implementations is plotted in Figure 3.14 and Figure 3.15.
From the objective function curve figure, "inc-I" obtains a more satisfied reduction than others; "inc-I" and "inc-diag" have a extremely better gradient norm curve

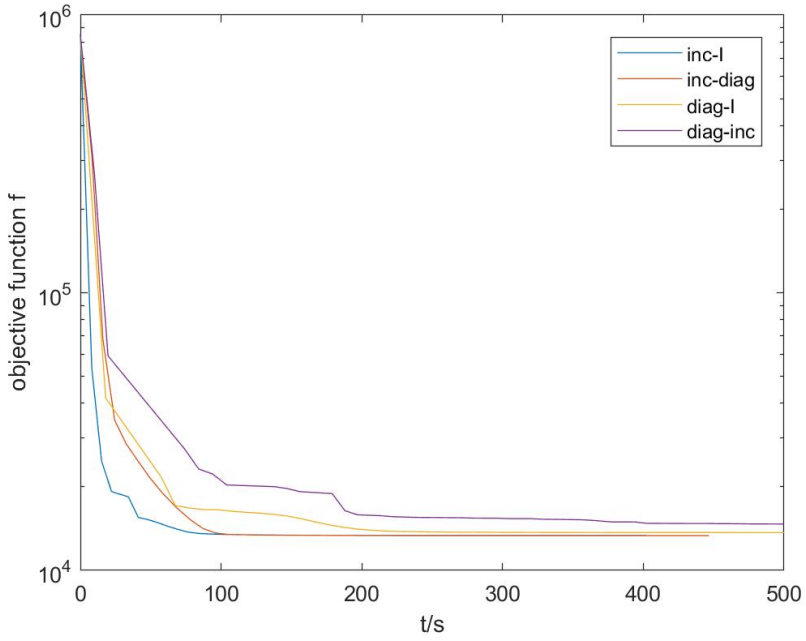

Figure 3.14: Comparison of the objective function curves between different DINM implementations (with different scaling matrix and different damping matrix). For example, "inc-diag" represents, scaling matrix is formed from (3.38), and damping matrix is formed from the diagonal of Hessian matrix. "inc-I" has the fastest and deepest decreasing of the objective function.

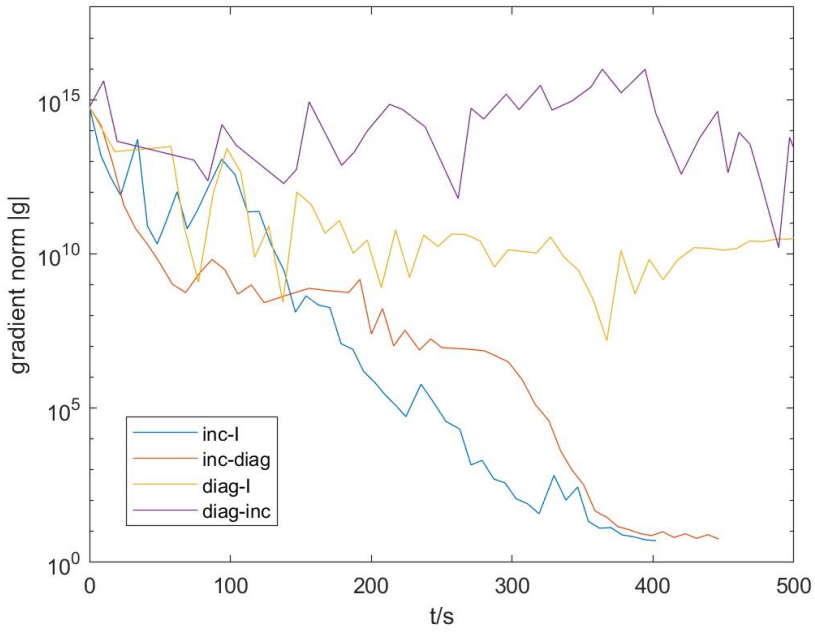

Figure 3.15: Comparison of the gradient norm curves between different DINM implementations. "inc-I" and "inc-diag" perform better in the gradient decline than others.

than others. These four typical implementations' processing time is presented in Table 3.7.

Table 3.7: Processing time comparison between DINM implementations. "inc-I" is the fastest implementation.

Implementation	inc-I	inc-diag	diag-I	diag-inc
Time per Newton-PCG/s	5.37	5.60	7.13	5.69
Time in 50 iteration/s	402	446	524	622

Then, we make a comparison of DINM with INM, TNM and baseline implementations in Figure 3.16, Figure 3.17, and Table 3.8. Both baseline implementations are in SC of $\boldsymbol{C}$ mode.
From gradient norm figure, our DINM algorithms have already gained a huge improvement in the gradient norm reduction over all other implementations. The number of all arguments, s = 23769, which means the dimension of the gradient is also 23769. After 50 steps, the gradient norm of DINM has been down to less than 10, i.e. the average value of each single gradient is below 10^{-3}. Even if considering the maximum value in the gradient vector, it has reduced to around 0.5. The results indicate that DINM finds a local optimal argument after 50 steps for this Ladybug dataset. CD implementation also yields a significant reduction in the gradient norm

curve. However, after several steps, the gradient norm of CD implementation maintains around 100, and does not decrease anymore. Besides, like we have discussed in Section 3.2.2, CD solver is not as robust as PCG solver, when facing a near singular or bad conditioned equation system.
From objective function figure, both DINM algorithms also generate a fast and large drop along the optimization. Furthermore, "DINM:inc-I" gets the lowest objective function after 50 steps among all implementations.
From processing time table, our DINM algorithms accelerates almost 1.5 times than INM and TNM. Even if they are still a little slower than two baseline implementations, their processing time is in an acceptable range considering their promotion in accuracy. Besides, since the explicit construction of

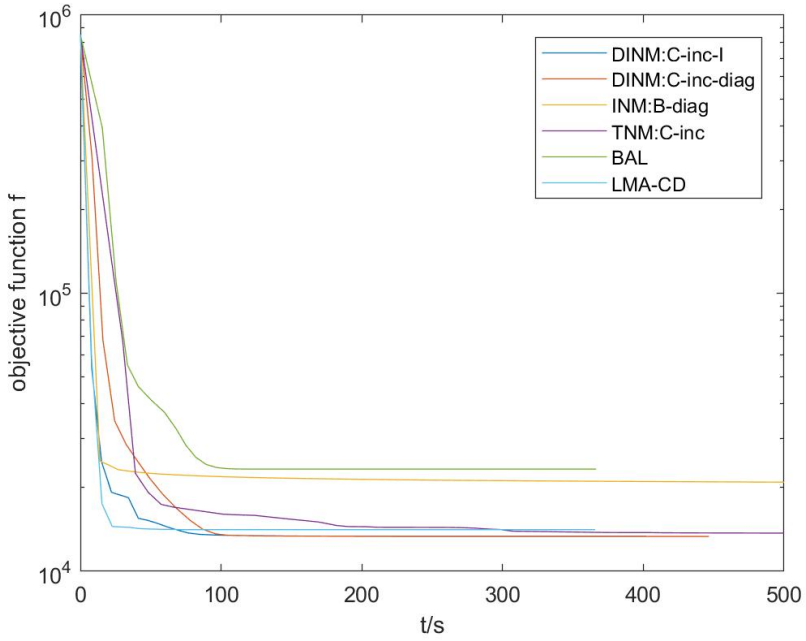

Figure 3.16: Comparison of the objective function curves between DINM implementations and INM, TNM and baseline implementations. The preconditioner in all implementations is selected as identity matrix (no preconditioner). "DINM:inc-I" presents DINM with "inc" scaling matrix and identity damping matrix; "DINM:inc-diag" presents DINM with "inc" scaling matrix and "diag" damping matrix, more details in caption of Figure 3.14. "INM:B-diag" presents INM with SC of $\boldsymbol{B}$ and "diag" scaling matrix, more details in caption of Figure 3.8. "TNM:C-inc" presents TNM with SC of $\boldsymbol{C}$ and "inc" damping matrix, more details in caption of Figure 3.10."BAL" presents the baseline implementation provided by [ASSS10]. "LMA-CD" presents the baseline implementation of Cholesky decomposition coupled with SC of $\boldsymbol{C}$, more details in Section 3.2.2. "inc-I" has the fastest and deepest decreasing of the objective function.

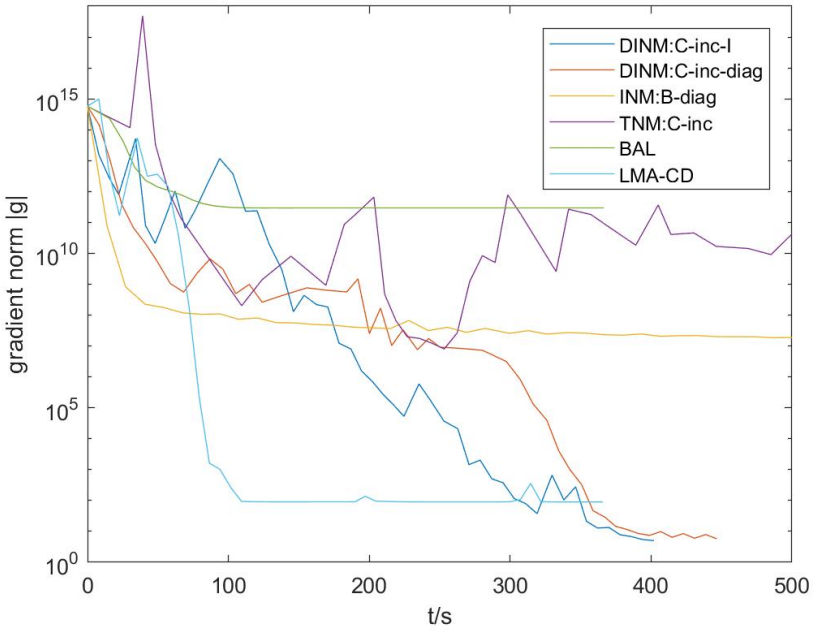

Figure 3.17: Comparison of the gradient norm curves between DINM implementations and INM, TNM and baseline implementations. "inc-I" and "inc-diag" perform better in the gradient decline than others.

Table 3.8: Processing time comparison between DINM implementations and INM, TNM and baseline implementations. "LMA-CD" is the fastest.

Implementation	DINM:inc-I	DINM:inc-diag	INM:$\boldsymbol{B}$-diag	TNM:$\boldsymbol{C}$-inc	BAL	LMA-CD
Time per PCG/s	5.37	5.60	10.91	6.32	4.62	4.73
Time in 50 steps/s	402	446	659	678	374	365

Selection of Preconditioner

In this section, the selection of preconditioner $\boldsymbol{M}$ in PCG solver is discussed. About more details of $\boldsymbol{M}$ please see Section 3.3.2. Since the purpose of the preconditioner is to reduce the condition number of system matrix of LSE, $\boldsymbol{A}$, the best preconditioner should has an inverse of $\boldsymbol{A}^{-1}$. In general, $\boldsymbol{A}^{-1}$ is impossible to compute, thus, a preconditioner should be easily to compute and closer to $\boldsymbol{A}^{-1}$ [NW06]. Since the preconditioner is also used as a system matrix of LSE in PCG loop, seeing Algorithm 1, another important requirement of preconditioner is sparse enough, at best, diagonal or block diagonal to ensure the inverse of $\boldsymbol{M}$ is cheap to compute.
However, it is hard to select a good preconditioner selection. The preconditioner in bundle adjustment is really sensitive, such that there is not a clear mechanism to determine a proper preconditioner in bundle adjustment [ASSS10]. Here, we select five different preconditioners $\boldsymbol{M}$ for DINM algorithm with scaling matrix formed from (3.38) and identity damping matrix (in SC of $\boldsymbol{C}$ mode). Since the LSE to solve is (3.100) in PCG of DINM, a good preconditioner $\boldsymbol{M}$ should be closed to $\boldsymbol{S_C}$ and diagonal. The proposed preconditioner is listed below.

- $\mathbf{I}_{9n\times 9n}$: identity matrix, i.e. without preconditioner.
- Diagonal blocks of $\boldsymbol{S_C}$: the diagonal blocks of $\boldsymbol{S_C}$ with size of 9×9 [ASSS10].
- $\boldsymbol{B}$: sub-matrix $\boldsymbol{B}$ in damped Hessian matrix $\boldsymbol{H_\mu}$ [ASSS10].
- Diagonal elements of $\boldsymbol{S_C}$: diagonal matrix formed by the diagonal elements of $\boldsymbol{S_C}$.
- Diagonal elements of $\boldsymbol{B}$: diagonal matrix formed by the diagonal elements of $\boldsymbol{B}$.

We implement the above five preconditioners in DINM. Since the termination condition of PCG is unchanged, the accuracy between different selection of preconditioner is the same. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. For each implementation, we run 50 iteration DINM steps. We compare the average iteration steps in PCG loop until the PCG loop is terminated, the average processing time in each PCG loop, and the time costing in each DINM steps. Still in each DINM iteration step there may exist more than one PCG loop to get a successful reduction in the objective function. The maximum iterative steps in PCG loop is chosen to be 1000.
The comparison results of different preconditioners are demonstrated in Figure 3.18, Figure 3.19, and Table 3.9.
Obviously, for this Ludybug dataset, without preconditioner is the best implementation. In other words, the system matrix of equation system (3.100), which is directly

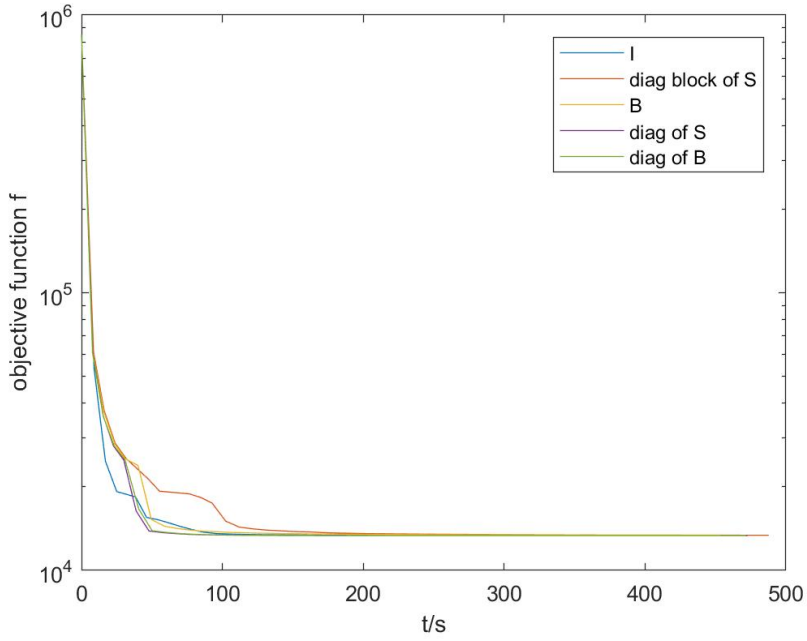

Figure 3.18: Comparison of the objective function curves between different preconditioners in DINM implementations. All implementations are based on the algorithm "DINM:inc-I" (seeing the previous section). "I" presents DINM with identity preconditioner (no preconditioner). "diag block of S" presents DINM with the preconditioner formed by the diagonal blocks of $\boldsymbol{S_C}$. "B" presents DINM with the preconditioner of $\boldsymbol{B}$. "diag of S" presents DINM with the preconditioner of diagonal elements of $\boldsymbol{S_C}$. "diag of B" presents DINM with the preconditioner of diagonal elements of $\boldsymbol{B}$. All implementations have a similar reduction of the objective function.

Table 3.9: Processing time comparison between different preconditioners in DINM implementations. Without preconditioner computes even faster.

Preconditioner	I	diag block of $\boldsymbol{S_C}$	$\boldsymbol{B}$	diag of $\boldsymbol{S_C}$	diag of $\boldsymbol{B}$
Time per PCG/s	5.37	6.79	6.55	6.85	6.84
Steps per PCG/s	256	835	920	913	920
Time in 50 steps/s	402	487	453	472	471

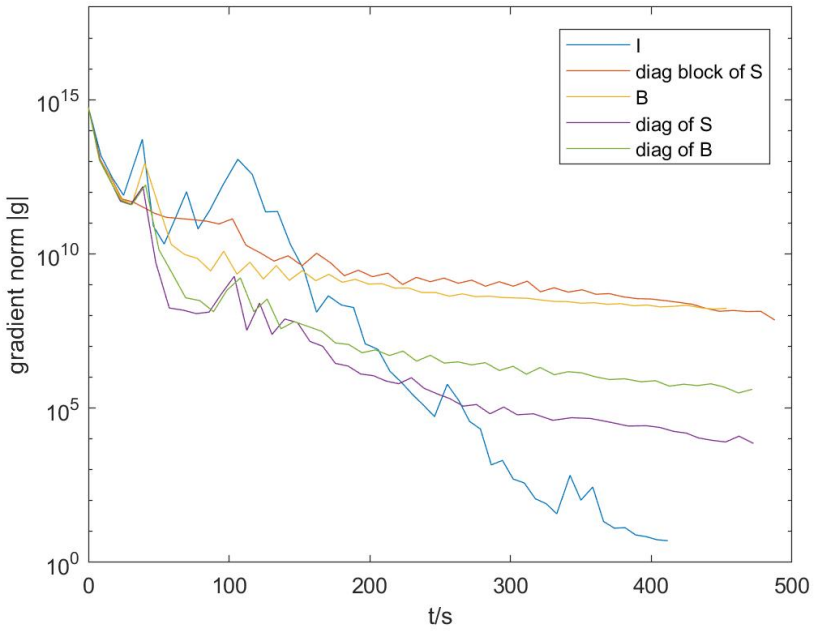

Figure 3.19: Comparison of the gradient norm curves between different preconditioners in DINM implementations. Implementation without preconditioner ("I") yields a faster and deeper decline curve of the gradient norm than others.

built from DINM, is already good conditioned without preconditioner. But this situation does not always happen in DINM. If with preconditioner, PCG also needs several extra calculation for preconditioner matrix, e.g. construction, inversion.

To be more intuitive about the influence from iteration steps in PCG loop, we also implement DINM with different maximum iteration steps in PCG loop. Here, the maximum iterative steps in PCG loop is chosen to be 100, 300, 500, 700, 1000, 2000, respectively, while still using "DINM:inc-I" without preconditioner. But the termination condition of PCG loop is the same. Because PCG solver only computed the equation system in an approximated solution, the more iteration steps there exist in one loop, the more exact the solution is.

The comparison results of different iteration steps in PCG solver is demonstrated in Figure 3.20, Figure 3.21, and Table 3.10.

From the objective function curve and the gradient norm curve, the implementations with more iteration steps in PCG loop gain a faster and a steeper descent in both curves, which means a more precise solution in each PCG loop contributes to a better optimization process. In addition, from Table 3.10, a larger threshold in iteration steps even gets fewer average iteration steps (till termination) in PCG loop, and less computation time. The reason is that a more precise solution leads to less computation in the following DINM steps, even if it may cost more PCG steps in this DINM steps. Thus, an enough large maximum iteration steps in PCG loop

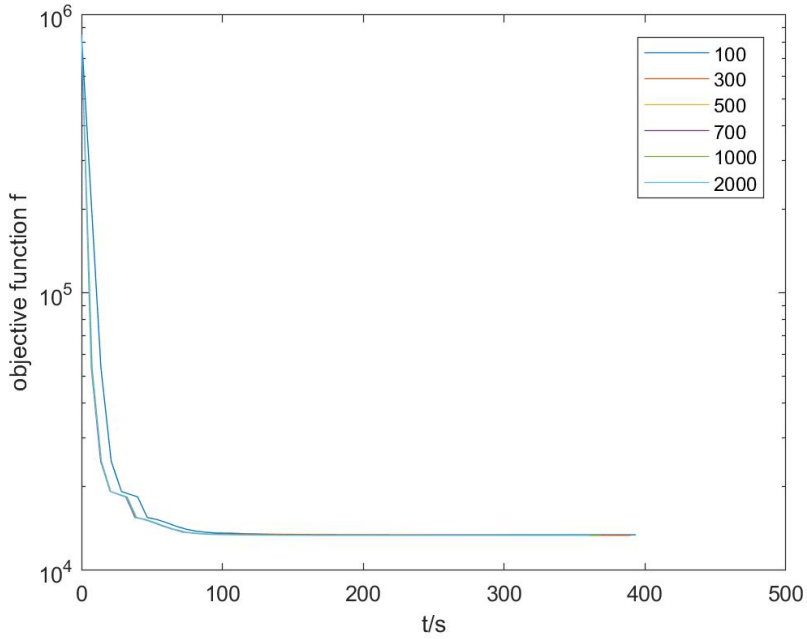

Figure 3.20: Comparison of the objective function curves between different iteration steps in PCG loop in DINM implementations. All implementations are based on the algorithm "DINM:inc-I" (seeing the previous section). The number presents the maximum iteration steps in PCG loop. All implementations have a similar objective curve.

Table 3.10: Processing time comparison between different iteration steps in PCG loop in DINM implementations. A larger number of maximum iterative steps in PCG loop even costs less time averagely.

Iteration Steps	100	300	500	700	1000	2000
Time per PCG/s	4.14	4.10	4.62	4.58	4.50	4.41
Steps per PCG/s	54	123	269	274	256	240
Time in 50 steps/s	393	390	363	368	365	360

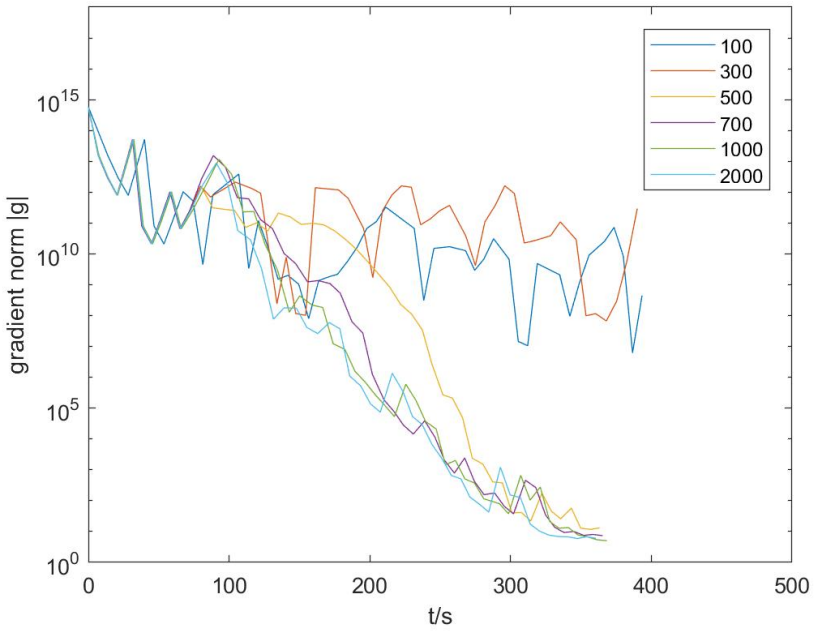

Figure 3.21: Comparison of the gradient norm curves between different iteration steps in PCG loop in DINM implementations. The gradient norm has a faster decreasing with a larger number of maximum iterative steps in PCG loop, and this threshold should be set to a value greater than 700 for this dataset.

must be confirmed in DINM to ensure an enough precise solution in PCG solver.
In addition, we still want to test the influence of termination condition in Algorithm 3. The termination factor t_ϵ controls the stop of PCG solver, and also controls the accuracy and the efficiency of PCG solver. Until here, the termination factor t_ϵ is set to the default value 0.1, which is also a popular value suggested by many bundle adjustment implementations. Here, the termination factor is chosen to be 0.5, 0.2, 0.1, 0.05, 0.01, respectively, while still using "DINM:inc-I" without preconditioner. But there is no constraint of the maximum iteration steps in PCG loop, i.e. PCG loop can only be stopped when the termination condition of t_ϵ is reached. In general, the smaller the termination factor is, the more exact the solution is, and also with more computation time.
The comparison results of different termination factors in PCG solver is demonstrated in Figure 3.22, Figure 3.23, and Table 3.11.

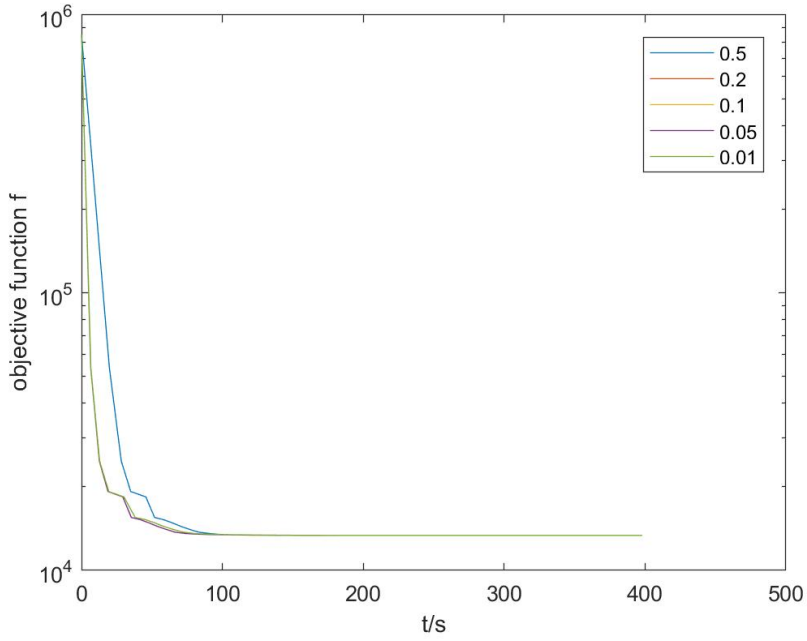

Figure 3.22: Comparison of the objective function curves between different termination factors in PCG loop in DINM implementations. All implementations are based on the algorithm "DINM:inc-I" (seeing the previous section). The number presents the termination factor in PCG loop. All implementations have a similar objective curve.

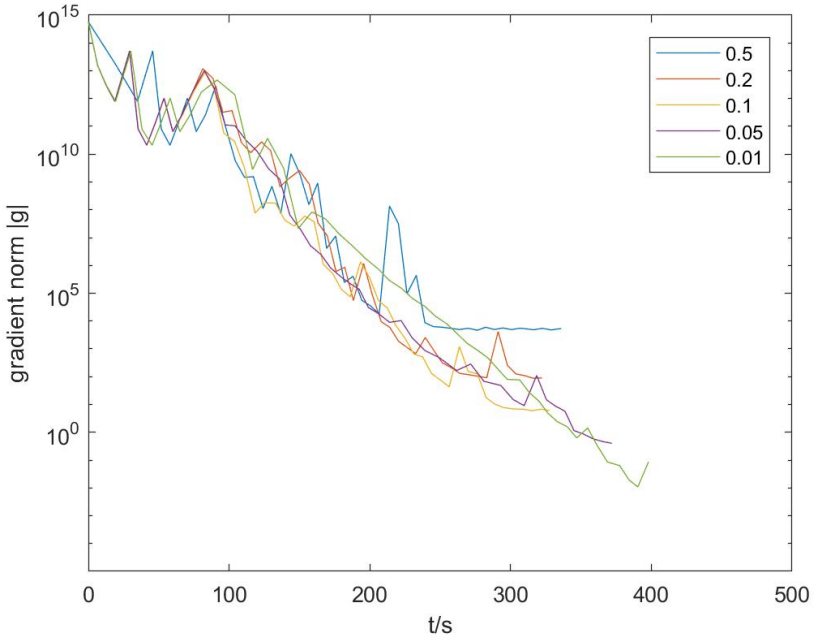

Figure 3.23: Comparison of the gradient norm curves between different termination factors in PCG loop in DINM implementations. The gradient norm has a more stable reduction with a smaller number of the termination factor, but a too small number also causes a long computation time. A proper termination factor can be set to a value around $0.05 \sim 0.1$.

Table 3.11: Processing time comparison between different termination factors in PCG loop in DINM implementations. A smaller number costs more processing time. A proper termination factor can be set to a value around 0.1.

Iteration Steps	0.5	0.2	0.1	0.05	0.01
Time per PCG/s	4.14	3.96	4.05	4.63	5.30
Time in 50 steps/s	335	322	327	371	397

Algorithm 5 Inexact Newton's Method

Require: Initial arguments vector $\boldsymbol{u}^0$ with (2.38), which contains $\{\boldsymbol{x}_i^0\}_{i=1}^{\mathrm{m}}$, $\{(\boldsymbol{\phi}_j^0, \boldsymbol{t}_j^0)\}_{j=1}^{\mathrm{n}}$, $\{\boldsymbol{y}_j^0\}_{j=1}^{\mathrm{n}}$; Observation field $(i,j) \in \mathcal{S}$; Observation dataset $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}}$; Scaling matrix $\mathbf{D_c}$; Maximum iterative steps in PCG loop, $\#_{PCG}$; Maximum iterative steps in INM, $\#_{INM}$; Termination condition of INM, ξ_1, ξ_2; Initial trust region radius Δ_c^0;

Ensure: Optimal arguments vector $\boldsymbol{u}^k$; Optimal objective function ψ_f^k;

Initialize the objective function $\psi_f^0 \leftarrow \psi_f(\boldsymbol{u}^0)$;
Initialize step number $k \leftarrow 0$
while $k < \#_{INM}$ **do**
    Compute current reprojection error vector $\boldsymbol{F}^k$ with (2.37);
    Compute current objective function $\psi_f^k \leftarrow \psi_f(\boldsymbol{u}^k)$ with (2.26);
    Compute current Jacobian matrix $\boldsymbol{J}^k$ (2.40);
    Compute current gradient vector $\boldsymbol{g}^k$ with (3.42);
    if $\|\boldsymbol{g}^k\| < \xi_1$ **or** $\psi_f^k < \xi_2$ **then**
        return $\boldsymbol{u}^k$ and ψ_f^k;
    end if
    Compute current Hessian matrix $\boldsymbol{H}^k$ with (3.43);
    Set flag $update \leftarrow$ "$False$";
    while $update =$ "$False$" **do**
        Compute the increment step $\boldsymbol{p}^k$, and reduction of model function $-m_f^k(\boldsymbol{p}^k)$ with Algorithm 4;
        Compute new objective function $\psi_f^{k+1} \leftarrow \psi_f(\boldsymbol{u}^{k+1})$ with (2.26);
        Compute ratio ρ^k with (3.104);
        Update trust region radius Δ_c^k with Algorithm 2;
        if $\rho^k > 0$ **then**
            Update arguments vector $\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{u}^k + \boldsymbol{p}^k$;
            $\Delta_c^{k+1} \leftarrow \Delta_c^k$;
            $k \leftarrow k + 1$;
            Set flag $update \leftarrow$ "$True$";
        end if
    end while
end while
return $\boldsymbol{u}^k$ and ψ_f^k;

Algorithm 6 Damping Factor Updating

Require: Current damping factor μ^k; Current auxiliary coefficient ν^k; Ratio ρ^k;
Ensure: Next damping factor μ^{k+1}; Next auxiliary coefficient ν^{k+1};

```
if ρ^k > 0 then
    μ^{k+1} ← μ * max(1/3, 1 − (2ρ^k − 1)^3);
    ν^{k+1} ← 2;
else
    μ^{k+1} ← μ * ν;
    ν^{k+1} ← 2ν^k;
end if
return μ^{k+1}, ν^{k+1};
```

Algorithm 7 LMA-PCG with Schur Complement of $\boldsymbol{C}$

Require: Hessian matrix $\boldsymbol{H}$ in form (3.45); Gradient vector $\boldsymbol{g}$ in form (3.63); Damping matrix $\boldsymbol{D}$; Damping factor μ
Ensure: Increment step $\boldsymbol{p}$; Reduction of model function $-m_f(\boldsymbol{p})$;

```
Compute the damped Hessian matrix H_μ in form (3.59)
Compute LSE S_C and v with (3.65) and (3.66);
Scaling S̃_C ← D^{-1} S_C D^{-1}, ṽ ← D^{-1} v;
Construct scaled Schur complement's LSE: S̃_C p̃_c = −ṽ
Compute preconditioner M with the strategy from Section 3.3.2;
Solve S̃_C p̃_c = −ṽ for p̃_c with Algorithm 1;
Reverse to unscaled solution, p_c ← D^{-1} p̃_c
Compute the full increment step vector p with (3.67);
Compute −m_f(p) with (3.102);
return p and −m_f(p);
```

Algorithm 8 Truncated Newton's Method

Require: Initial arguments vector $\boldsymbol{u}^0$ with (2.38), which contains $\{\boldsymbol{x}_i^0\}_{i=1}^{\mathrm{m}}$, $\{(\boldsymbol{\phi}_j^0, \boldsymbol{t}_j^0)\}_{j=1}^{\mathrm{n}}$, $\{\boldsymbol{y}_j^0\}_{j=1}^{\mathrm{n}}$; Observation field $(i,j) \in \mathcal{S}$; Observation dataset $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}}$; Damping matrix $\boldsymbol{D}$; Maximum iterative steps in PCG loop, $\#_{PCG}$; Maximum iterative steps in TNM, $\#_{TNM}$; Termination condition of TNM, ξ_1, ξ_2; Initial damping factor μ^0;

Ensure: Optimal arguments vector $\boldsymbol{u}^k$; Optimal objective function ψ_f^k;

Initialize the objective function $\psi_f^0 \leftarrow \psi_f(\boldsymbol{u}^0)$;
Initialize step number $k \leftarrow 0$
while $k < \#_{TNM}$ **do**
    Compute current reprojection error vector $\boldsymbol{F}^k$ with (2.37);
    Compute current objective function $\psi_f^k \leftarrow \psi_f(\boldsymbol{u}^k)$ with (2.26);
    Compute current Jacobian matrix $\boldsymbol{J}^k$ (2.40);
    Compute current gradient vector $\boldsymbol{g}^k$ with (3.42);
    if $\|\boldsymbol{g}^k\| < \xi_1$ **or** $\psi_f^k < \xi_2$ **then**
        return $\boldsymbol{u}^k$ and ψ_f^k;
    end if
    Compute current Hessian matrix $\boldsymbol{H}^k$ with (3.43);
    Set flag $update \leftarrow$ "$False$";
    while $update =$ "$False$" **do**
        Compute the increment step $\boldsymbol{p}^k$, and reduction of model function $-m_f^k(\boldsymbol{p}^k)$ with Algorithm 7;
        Compute new objective function $\psi_f^{k+1} \leftarrow \psi_f(\boldsymbol{u}^{k+1})$ with (2.26);
        Compute ratio ρ^k with (3.104);
        Update damping factor μ^k with Algorithm 6;
        if $\rho^k > 0$ **then**
            Update arguments vector $\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{u}^k + \boldsymbol{p}^k$;
            $\mu^{k+1} \leftarrow \mu^k$;
            $k \leftarrow k + 1$;
            Set flag $update \leftarrow$ "$True$";
        end if
    end while
end while
return $\boldsymbol{u}^k$ and ψ_f^k;

Algorithm 9 Damped Inexact Newton's Method

Require: Initial arguments vector $\boldsymbol{u}^0$ with (2.38), which contains $\{\boldsymbol{x}_i^0\}_{i=1}^{\mathrm{m}}$, $\{(\boldsymbol{\phi}_j^0, \boldsymbol{t}_j^0)\}_{j=1}^{\mathrm{n}}$, $\{\boldsymbol{y}_j^0\}_{j=1}^{\mathrm{n}}$; Observation field $(i,j) \in \mathcal{S}$; Observation dataset $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}}$; Scaling matrix $\mathbf{D_c}$; Damping matrix $\boldsymbol{D}$; Maximum iterative steps in PCG loop, $\#_{PCG}$; Maximum iterative steps in DINM loop, $\#_{DINM}$; Termination condition of DINM, ξ_1, ξ_2; Initial trust region radius Δ_c^0; Initial damping factor μ^0;

Ensure: Optimal arguments vector $\boldsymbol{u}^k$; Optimal objective function ψ_f^k;

Initialize the objective function $\psi_f^0 \leftarrow \psi_f(\boldsymbol{u}^0)$;
Initialize step number $k \leftarrow 0$;
while $k < \#_{DINM}$ **do**
    Compute current reprojection error vector $\boldsymbol{F}^k$ with (2.37);
    Compute current objective function $\psi_f^k \leftarrow \psi_f(\boldsymbol{u}^k)$ with (2.26);
    Compute current Jacobian matrix $\boldsymbol{J}^k$ (2.40);
    Compute current gradient vector $\boldsymbol{g}^k$ with (3.42);
    if $\|\boldsymbol{g}^k\| < \xi_1$ **or** $\psi_f^k < \xi_2$ **then**
        return $\boldsymbol{u}^k$ and ψ_f^k;
    end if
    Compute current Hessian matrix $\boldsymbol{H}^k$ with (3.43);
    Set flag $update \leftarrow$ "$False$";
    while $update =$ "$False$" **do**
        $\boldsymbol{H}_{\boldsymbol{\mu}}^k \leftarrow \boldsymbol{H}^k + \mu^k \boldsymbol{D}$
        Compute the increment step $\boldsymbol{p}^k$, and reduction of model function $-m_f^k(\boldsymbol{p}^k)$ with Algorithm 4;
        Compute new objective function $\psi_f^{k+1} \leftarrow \psi_f(\boldsymbol{u}^{k+1})$ with (2.26);
        Compute ratio ρ^k with (3.104);
        Update trust region radius Δ_c^k with Algorithm 2;
        Update damping factor μ^k with Algorithm 6;
        if $\rho^k > 0$ **then**
            Update arguments vector $\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{u}^k + \boldsymbol{p}^k$;
            $\mu^{k+1} \leftarrow \mu^k$;
            $\Delta_c^{k+1} \leftarrow \Delta_c^k$;
            $k \leftarrow k + 1$;
            Set flag $update \leftarrow$ "$True$";
        end if
    end while
end while
return $\boldsymbol{u}^k$ and ψ_f^k;

Chapter 4

Further Improvements

In this chapter, we further improve our DINM algorithm with three strategies. Firstly, we introduce an adaptively switched Schur complement mode in DINM (Adaptive DINM); secondly, we replace the global parameterization with local parameterization, and provide a local parameterized DINM (local DINM); finally, to solve the outliers problem, we use reweighted potential function in DINM, and propose an iteratively reweighted least squares (IRLS) algorithm.

4.1 Adaptive DINM

In some of the previous DINMs, the gradient norm has a stark oscillation and only a slight reduction after several steps, seeing Figure 3.15, Figure 3.19 and Figure 3.21. Even if each increment step can be still approximated with PCG solver, however, this approximation is already far from the real solution. This deterioration is caused by the bad conditioned matrix $\boldsymbol{C}$, and the inverse of $\boldsymbol{C}$ is hard to be accurately computed. At this time, the matrix $\boldsymbol{C}$ has a large condition number over 10^7. Even if with a larger number of iterative steps in PCG loop, i.e. larger $\#_{PCG}$, the solution does not improve distinctly. The PCG loop is not terminated until the maximum iteration step reaches. Nevertheless, the condition number of matrix $\boldsymbol{B}$ is still relatively small, around 10^4.
Therefore, we develop an adaptive DINM algorithm, which switches between the mode of Schur complement of $\boldsymbol{C}$ and the mode of Schur complement of $\boldsymbol{B}$ in each iteration step according to the condition numbers of both matrices. The adaptive DINM algorithm ensures a more robust and more precise solution from PCG in each iteration.
The Adaptive DINM (ADINM) algorithm is presented in Algorithm 10.
The scaling matrices for $\boldsymbol{p_c}$ and $\boldsymbol{p_p}$ come from the same increment step (3.38), so that the norms of every argument after scaling are at the same order of magnitude. Thus, when the mode is switched to SC of $\boldsymbol{B}$, the radius of trust region for $\widetilde{\boldsymbol{p}}_{\boldsymbol{p}}$ is scaled according to the ratio between the dimensions of both vectors, as in Algorithm 10

line 18. In the whole algorithm, only Δ_c is needed to be explicitly constructed and updated.
We choose three typical DINM implementations in Section 3.3.5, and implement their corresponding adaptive DINM version to compare the improvement. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. The maximum iterative steps in PCG loop is chosen to be 3000.

- "inc-I" : "inc" scaling matrix, identity damping matrix, without preconditioner.
- "inc-diag" : "inc" scaling matrix, damping matrix formed from diagonal elements of Hessian matrix, without preconditioner.
- "inc-I-diag" : "inc" scaling matrix, identity damping matrix, with preconditioner formed from diagonal elements of $\boldsymbol{B}$ in SC of $\boldsymbol{C}$ (diagonal elements of $\boldsymbol{C}$ in SC of $\boldsymbol{B}$).

The curves of the objective function along 50 successive (Adapt) DINM iteration steps are plotted in Figure 4.1, and the curves of the gradient norm are plotted in Figure 4.2.
Although for solving good conditioned LSE, the mode of Schur complement of $\boldsymbol{B}$ costs more time than the mode of Schur complement of $\boldsymbol{C}$, solving the LSE built with relatively good conditioned $\boldsymbol{B}$ even saves much computational effort than with bad conditioned $\boldsymbol{C}$. In comparison of time efficiency, the whole processing time is collected, shown in Table 4.1.

Table 4.1: Processing time comparison between DINMs and ADINMs. Adaptive DINMs cost more time than original DINMs, but with less iteration steps in PCG loop.

Implementation	DINM:inc-I	ADINM:inc-I	DINM:inc-I-diag	ADINM:inc-I-diag
Time per Newton-PCG/s	4.16	5.20	5.61	6.44
Steps per PCG/s	240	224	1132	969
Time in 50 iteration/s	340	394	398	441

From the objective function curve figure, DINM and ADINM almost generate the same curve. From the gradient norm curve figure, ADINM and DINM are the same in the first several iterations. The reason is because the condition number of $\boldsymbol{C}$ is smaller than $\boldsymbol{B}$, so that ADINM uses the mode of SC of $\boldsymbol{C}$, i.e. the same as DINM. In the last several iterations, ADINM gains a significant reduction of gradient norm related to DINM, because the condition number of $\boldsymbol{B}$ is smaller at this moment, and ADINM switches to the mode of SC of $\boldsymbol{B}$. In Table 4.1, the average iteration steps in PCG loop of ADINM is fewer than DINM, since the adaptive switching avoids an inverse operation of bad conditioned matrix, and guarantees a relative precise

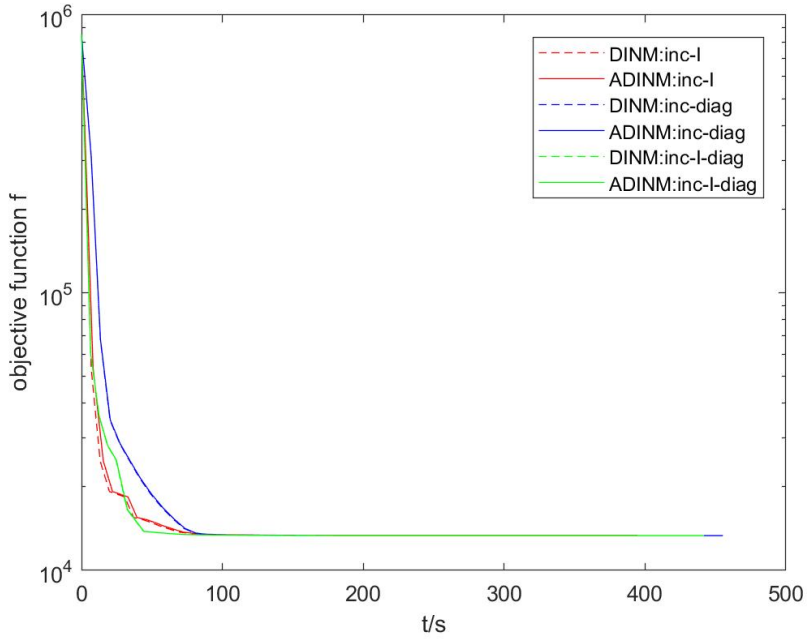

Figure 4.1: Comparison of the objective function curve between DINM implementations and corresponding ADINM implementations. Every DINM implementation and its corresponding ADINM implementation are grouped together with the same color of curves. DINMs are plotted in dash lines; ADINMs are plotted in solid lines. Both ADINM and DINM have a similar reduction curve of the objective function.

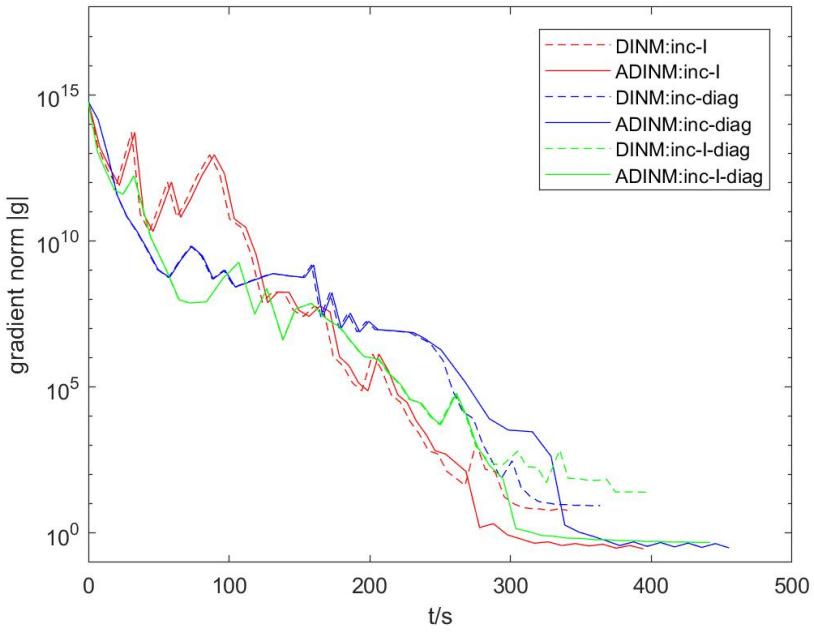

Figure 4.2: Comparison of the gradient norm curve between DINM implementations and corresponding ADINM implementations. ADINM yields a deeper decline curve of the gradient norm than DINM.

solution from PCG. However, the computation time spent in SC of $\boldsymbol{B}$ is larger than SC of $\boldsymbol{C}$, seeing Section 3.2.1. Therefore, even if with a fewer iteration steps in PCG loop, ADINM still costs more time than DINM.
In addition, when comparing the results from "DINM:inc-I-diag" with 3000 iteration steps (green dash line in Figure 4.2) with the results from "DINM:inc-I-diag" with 1000 iteration steps (olive green solid line in Figure 3.19), we conclude that increasing the maximum iteration steps in PCG loop promotes the optimization process hugely, and even with less computation time.

4.2 Local Parameterization

The above robotic kinematics used in the previous sections is based on the global parameterization, seeing Section 3.1.2. The camera pose parameters for a specific frame j contain two parts, rotation angle vector $\boldsymbol{\phi}_j$, and translation vector $\boldsymbol{t}_j$.
Now, we propose a new parameterization process, local parameterization, where the rotation matrix and the translation vector are updated locally according to the current camera pose, i.e. a retraction to SE(3) at $\{\boldsymbol{R}_j, \boldsymbol{t}_j\}$. With local parameterization, the linearization around current points fits better than global parameterization, i.e. the mapping from se(3) to SE(3) is more accurate.

$$\boldsymbol{R_l}(\boldsymbol{\phi}_j + \boldsymbol{\delta\phi}_j) = \boldsymbol{R}(\boldsymbol{\phi}_j)\mathrm{Exp}(\boldsymbol{\delta\phi}_j) \ , \tag{4.1}$$

$$\boldsymbol{t_l}(\boldsymbol{t}_j + \boldsymbol{\delta t}_j) = \boldsymbol{t}_j + \boldsymbol{R}(\boldsymbol{\phi}_j)\boldsymbol{\delta t}_j \ , \tag{4.2}$$

$$\boldsymbol{R}_j = \boldsymbol{R}(\boldsymbol{\phi}_j) \ . \tag{4.3}$$

The 3D transformation of $\boldsymbol{x}_i$ in j-th camera system, (3.5), related to local increments $\boldsymbol{\delta\phi}_j$ and $\boldsymbol{\delta t}_j$, are redefined as,

$$\boldsymbol{\delta R}_j = \mathrm{Exp}(\boldsymbol{\delta\phi}_j) \ , \tag{4.4}$$

$$\boldsymbol{X}_{ij} + \boldsymbol{\delta X}_{ij} = \boldsymbol{R}_j(\boldsymbol{\delta R}_j\boldsymbol{x}_i + \boldsymbol{\delta t}_j) + \boldsymbol{t}_j \ . \tag{4.5}$$

In local parameterization, a new set of arguments containing all rotation matrices, all translation vector, all camera characteristic parameters, and all 3D feature points coordinates is used,

$$\boldsymbol{w} = \{\{\boldsymbol{R}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{t}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{y}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{x}_i\}_{i=1}^{\mathrm{m}}\} \ . \tag{4.6}$$

The reprojection error vector is then presented as a function of the new arguments $\boldsymbol{w}$, but its structure maintains the same as before,

$$\boldsymbol{F}(\boldsymbol{w}) \ . \tag{4.7}$$

In each iteration step, we still obtain the increment step $\boldsymbol{p}$ corresponding to the previous parameter vector $\boldsymbol{u}$ in (2.24) as in the previous section, seeing (2.33),

$$\boldsymbol{p} = \{\{\boldsymbol{\delta\phi}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta t}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta y}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta x}_i\}_{i=1}^{\mathrm{m}}\} \ . \tag{4.8}$$

However, the updating of the arguments are not the simple "add" operation anymore. The arguments set $\boldsymbol{w}$ is updated according to (4.1) and (4.2), as,

$$\boldsymbol{w} \boxplus \boldsymbol{p} = \{\{\boldsymbol{R}_j\mathrm{Exp}(\boldsymbol{\delta\phi}_j)\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{t}_j + \boldsymbol{R}_j\boldsymbol{\delta t}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{y}_j + \boldsymbol{\delta y}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{x}_i + \boldsymbol{\delta x}_i\}_{i=1}^{\mathrm{m}}\} \ . \tag{4.9}$$

In local parameterization, the value of the Jacobian matrix is also changed. The Jacobian matrix still presents the partial derivatives from the reprojection error vector to the arguments vector $\boldsymbol{u}$. Thus, the structure of Jacobian matrix is the same as before, i.e. $2\mathrm{l} \times \mathrm{s}$. The Jacobian matrix in local parameterization is computed as,

$$\boldsymbol{J} = \left.\frac{\partial \boldsymbol{F}(\boldsymbol{w} \boxplus \boldsymbol{p})}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{0}} \ . \tag{4.10}$$

The partial derivatives of single reprojection error $\boldsymbol{F}_{ij}$ to its corresponding camera parameters and feature position parameters are presented as,

$$\left.\frac{\partial \boldsymbol{F}_{ij}(\boldsymbol{w}_{ij} \boxplus \boldsymbol{\delta\phi}_j)}{\partial \boldsymbol{\delta\phi}_j}\right|_{\boldsymbol{\delta\phi}_j=\boldsymbol{0}} , \tag{4.11}$$

$$\left.\frac{\partial \boldsymbol{F}_{ij}(\boldsymbol{w}_{ij} \boxplus \boldsymbol{\delta t}_j)}{\partial \boldsymbol{\delta t}_j}\right|_{\boldsymbol{\delta t}_j=\mathbf{0}}, \tag{4.12}$$

$$\left.\frac{\partial \boldsymbol{F}_{ij}(\boldsymbol{w}_{ij} \boxplus \boldsymbol{\delta y}_j)}{\partial \boldsymbol{\delta y}_j}\right|_{\boldsymbol{\delta y}_j=\mathbf{0}}, \tag{4.13}$$

$$\left.\frac{\partial \boldsymbol{F}_{ij}(\boldsymbol{w}_{ij} \boxplus \boldsymbol{\delta x}_i)}{\partial \boldsymbol{\delta x}_i}\right|_{\boldsymbol{\delta x}_i=\mathbf{0}}, \tag{4.14}$$

with $\boldsymbol{w}_{ij}$, which contains all parameters in $\boldsymbol{w}$ corresponding to i-th feature points and j-th image frame.

The Jacobian matrix is still calculated through chain rules as in Section 3.1.2. For local parameterization, if we consider f is a function of a, the partial derivative from f to a in local parameterization is then expressed as,

$$\left.\frac{\partial f(a+\delta a)}{\partial \delta a}\right|_{\delta a=0} = \frac{\partial (f+\delta f)}{\partial (a+\delta a)} \left.\frac{\partial (a+\delta a)}{\partial \delta a}\right|_{\delta a=0} = \left.\frac{\partial f(a+\delta a)}{\partial (a+\delta a)}\right|_{\delta a=0} = \frac{\partial f(a)}{\partial a} . \tag{4.15}$$

Since the local increments of $\boldsymbol{\delta x}_i$ and $\boldsymbol{\delta y}_j$ are basic addition operation as (4.15), $\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{y}_j}$ and $\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{x}_i}$ are the same as in (3.36) and (3.37). Besides, every formula in chain rule demonstrated in Section 3.1.2 maintains the same, except $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{t}_j}$ and $\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}$.

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{t}_j} = \left.\frac{\partial \boldsymbol{X}_{ij}(\boldsymbol{t}_j \boxplus \boldsymbol{\delta t}_j)}{\partial \boldsymbol{\delta t}_j}\right|_{\boldsymbol{\delta t}_j=\mathbf{0}} = \boldsymbol{R}_j , \tag{4.16}$$

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = \left.\frac{\partial \boldsymbol{X}_{ij}(\boldsymbol{R}_j \boxplus \boldsymbol{\delta \phi}_j)}{\partial \boldsymbol{\delta \phi}_j}\right|_{\boldsymbol{\delta \phi}_j=\mathbf{0}} = \boldsymbol{R}_j \left.\frac{\partial \boldsymbol{\delta R}_j \boldsymbol{x}_i}{\partial \boldsymbol{\delta \phi}_j}\right|_{\boldsymbol{\delta \phi}_j=\mathbf{0}} . \tag{4.17}$$

$$\left.\frac{\partial \boldsymbol{\delta R}_j \boldsymbol{x}_i}{\partial \boldsymbol{\delta \phi}_j}\right|_{\boldsymbol{\delta \phi}_j=\mathbf{0}} = \left.\frac{\partial \mathrm{Exp}(\boldsymbol{\delta \phi}_j)\boldsymbol{x}_i}{\partial \boldsymbol{\delta \phi}_j}\right|_{\boldsymbol{\delta \phi}_j=\mathbf{0}} = -\mathrm{Exp}(\mathbf{0})\widehat{\boldsymbol{x}}_i \boldsymbol{J}_r(\mathbf{0}) = -\widehat{\boldsymbol{x}}_i . \tag{4.18}$$

Here $\left.\frac{\partial \boldsymbol{\delta R}_j \boldsymbol{x}_i}{\partial \boldsymbol{\delta \phi}_j}\right|_{\boldsymbol{\delta \phi}_j=\mathbf{0}}$ can also be obtained from (3.28) by substituting $\boldsymbol{\phi}_j = \mathbf{0}$.

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = -\boldsymbol{R}_j \begin{pmatrix} 0 & -x_{i,\mathrm{z}} & x_{i,\mathrm{y}} \\ x_{i,\mathrm{z}} & 0 & -x_{i,\mathrm{x}} \\ -x_{i,\mathrm{y}} & x_{i,\mathrm{x}} & 0 \end{pmatrix} . \tag{4.19}$$

The partial derivatives of the local parameterization are much simpler than the global parameterization. In local parameterization, derivatives needs fewer multiplications, and does not need trigonometric functions. In each iteration step, trigonometric functions only appear in rotation matrix updating process (4.4). Besides, since the increment step is always small, these trigonometric functions can also be avoided with the approximation of small angles, which also saves the calculation time. The disadvantage of the local parameterization is that it costs little more memory, since it needs to store all rotation matrices (3×3) of each image frame rather than rotation angle vector (3×3) in optimization.

Here, We implement the local parameterization in "DINM:inc-I" instead of the global parameterization. The maximum iterative steps in PCG loop is chosen to be 1000. The testing computer is Surface Book with i7-6600U CPU @ 2.60GHz, 8.00GB RAM. The test dataset is the full Ladybug dataset provided in Section 2.3. We still run 50 iteration steps.
A comparison of the objective function and the gradient norm between local and global parameterization in "DINM:inc-I" is plotted in Figure 4.3 and Figure 4.4.

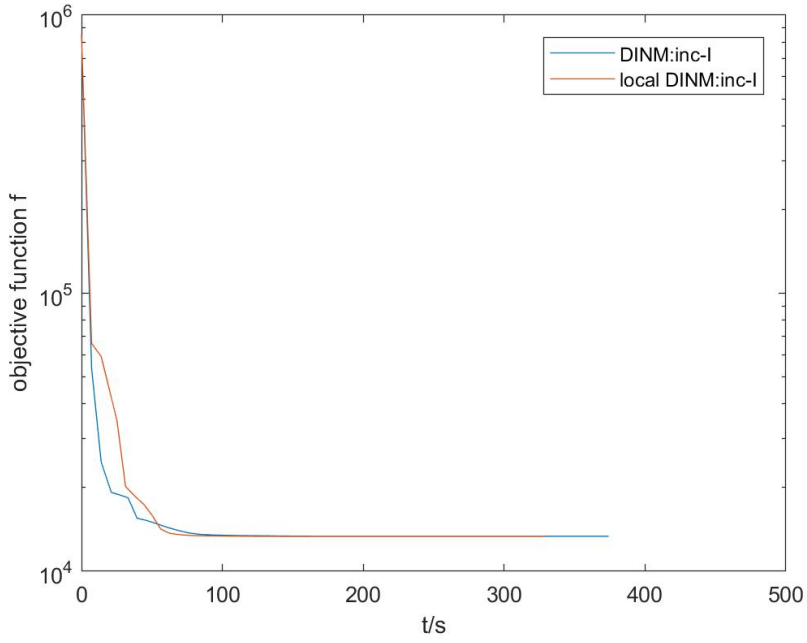

Figure 4.3: Comparison of the objective function curve between local and global parameterization. Local parameterization has a similar objective function curve as global parameterization.

From the objective function curve figure, both parameterizations obtain a similar reduction; in gradient norm curves, local parameterization has a faster decline.
Their processing time is presented in Table 4.2. Because the local parameterization needs less computation in Jacobian matrix, even if it spends more iterations in PCG loop, it still yields a faster optimization of 50 iterations.

Table 4.2: Processing time comparison between local and global parameterization. Local parameterization computes faster than global parameterization.

Implementation	local	global
Time per PCG/s	4.72	4.94
Steps per PCG/s	292	256
Time in 50 iteration/s	329	374

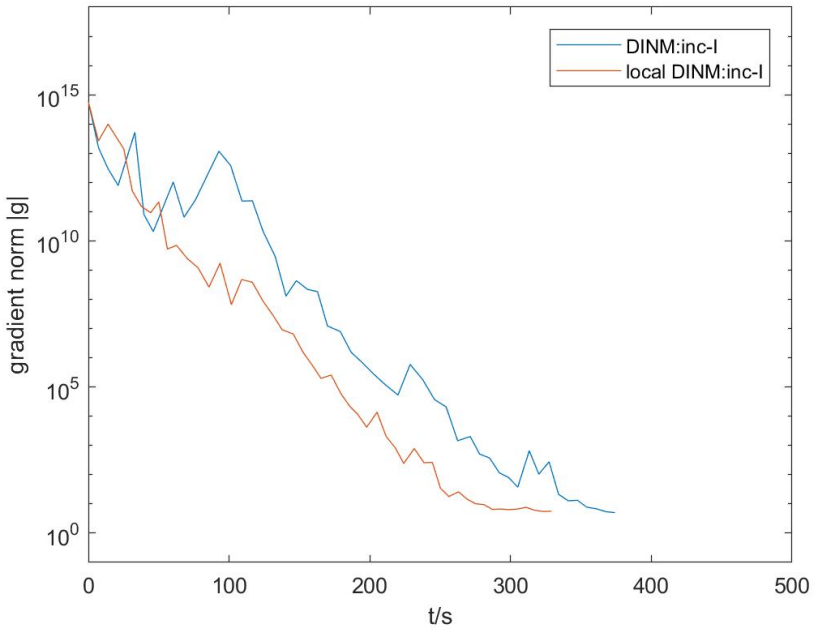

Figure 4.4: Comparison of the gradient norm curve between local and global parameterization. Local parameterization has a little faster reduction curve of the gradient norm.

4.3 Iteratively Reweighted Least Squares (IRLS)

In this section, we use a reweighted potential function of reprojection errors instead of the square formed objective function, which resolves the outliers problem. Besides, we also introduce an iteratively reweighted least squares algorithm (IRLS) with local parameterization.

4.3.1 Reweighted Potential Function

Even if the gradient norm continues to reduce to a smaller value with ADINM or DINM, the declined value of the objective function is still not satisfied. In other words, the found local optimal value of the objective function is in a larger order than expected.
Besides, from the results plots of reprojection errors in Chapter 5 (Figure 5.5), there are only a little observations on the boundary of images which are different with the their reprojected points. Other observations fit the reprojected points pretty well. Thus, the most effort of the optimization is used to balance the difference of these outliers. However, the fitness with the outliers generates an unexpected optimization solution in general, whose arguments values are far from the real optimal arguments values.

The distributions of the value of gradient vector and the value of reprojection error vector after 50 iteration with Adaptive DINM are plotted in Figure 4.6 and Figure 4.5.

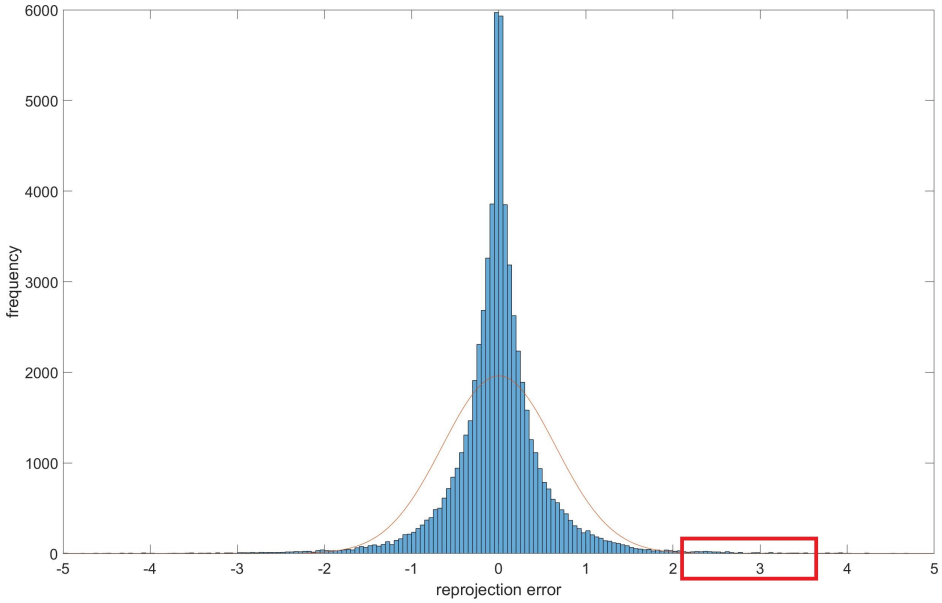

Figure 4.5: The frequency histogram of reprojection errors after 50 iteration steps of ADINM. The orange curve presents the Gaussian probability distribution function with the same mean and variance. The example red rectangle locates the outliers, where the frequency of reprojection errors is higher than corresponding Gaussian distribution.

From the figures, obviously, only a small part of elements in the gradient vector and reprojection vector still has a large value in comparison with others after 50 iteration steps. The percentages whose absolute values are greater than 2 in reprojection error vector is 1.60%, and the corresponding probability is 0.2% in the Gaussian distribution with the same mean and variance. The percentages whose absolute values are greater than 0.01 in gradients vector is 0.45%, and the corresponding probability is 0.05% in the Gaussian distribution with the same mean and variance. In order to recede the influence from outliers, a new potential function, $\psi_{f,\bullet(\gamma)}(\boldsymbol{u})$, is selected as the objective function. $\psi_{f,\bullet(\gamma)}(\boldsymbol{u})$ is a objective function related to a parameter γ, which controls the slope of the potential function $\Psi_{\bullet(\gamma)}(\|F_i\|)$ in different values. In this thesis, Huber loss function and truncated quadratic function are used as potential function [HW77].

$$\psi_{f,\bullet(\gamma)}(\boldsymbol{u}) = \sum_{i=1}^{2l} \Psi_{\bullet(\gamma)}(\|F_i\|) \ . \tag{4.20}$$

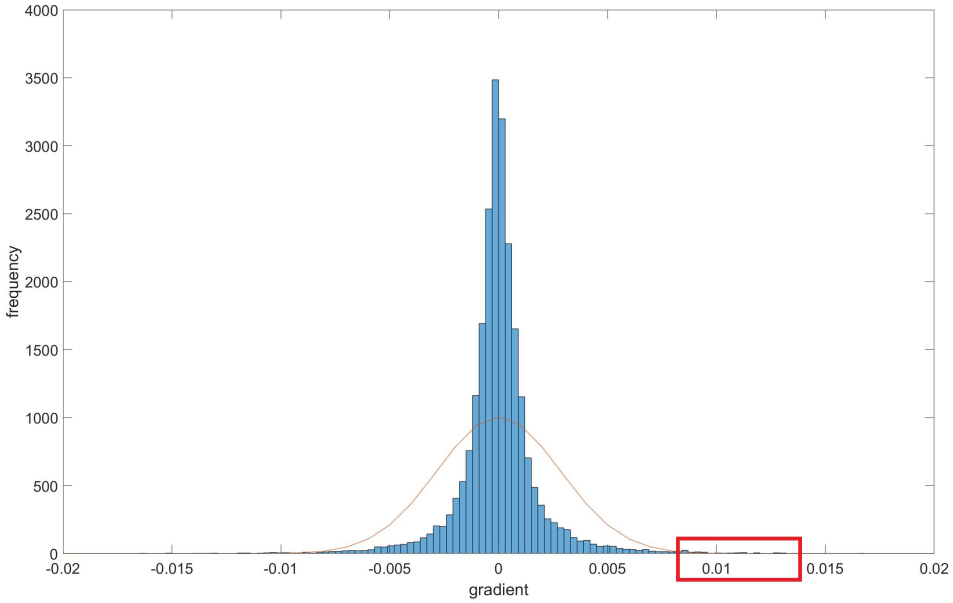

Figure 4.6: The frequency histogram of gradients after 50 iteration steps of ADINM. The orange curve presents the Gaussian probability distribution function with the same mean and variance. The example red rectangle locates the outliers, where the frequency of reprojection errors is higher than corresponding Gaussian distribution.

$$\Psi_{Huber(\gamma)}(\|F_i\|) = \begin{cases} \|F_i\|^2/2, & \text{if } \|F_i\| \leq \gamma, \\ \gamma(\|F_i\| - \gamma/2), & \text{if } \|F_i\| > \gamma. \end{cases} . \tag{4.21}$$

$$\Psi_{TQ(\gamma)}(\|F_i\|) = min(\|F_i\|^2, \|\gamma\|^2)/2 \ . \tag{4.22}$$

The new potential function has a smaller cost value in objective function for large errors in comparison with (2.39), so that the outliers contribute less in objective function. The new optimization process of the objective function can be obtained with a new introduced weight matrix $\mathbf{W}$. $\mathbf{W}$ is a diagonal matrix with positive elements, as

$$w_i = \frac{\Psi'_{\bullet(\gamma)}(\|F_i\|)}{\|F_i\|} \ , \tag{4.23}$$

$$\mathbf{W} = diag(w_i) \ . \tag{4.24}$$

In k-th iteration step, the increment step of parameter vector $\boldsymbol{p}^k$ is obtained by,

$$\boldsymbol{p}^k = \arg\min_{\boldsymbol{p}\in\mathbb{R}^s} \frac{1}{2|\mathcal{S}|}(\boldsymbol{F}^k + \boldsymbol{J}^k\boldsymbol{p})^T\mathbf{W}^k(\boldsymbol{F}^k + \boldsymbol{J}^k\boldsymbol{p}) \ . \tag{4.25}$$

The objective function, $\psi_{f,\bullet(\gamma)}(\boldsymbol{u})$, can be considered as the second norm of the reweighted reprojection error vector,

The corresponding reweighted Hessian matrix and reweighted gradient vector can be considered as,

$$\boldsymbol{H}_{\mathbf{W}} = \boldsymbol{J}^T \mathbf{W} \boldsymbol{J} \ , \tag{4.26}$$

$$\boldsymbol{g}_{\mathbf{W}} = \boldsymbol{J}^T \mathbf{W} \boldsymbol{F} \ . \tag{4.27}$$

The LSE (3.55) in each iteration step of DINM is then replaced with,

$$(\boldsymbol{J}^{kT} \boldsymbol{W}^k \boldsymbol{J}^k + \mu \boldsymbol{D}) \boldsymbol{p}^k = -\boldsymbol{J}^{kT} \boldsymbol{W}^k \boldsymbol{F}^k \ . \tag{4.28}$$

4.3.2 Iteratively Reweighted Least Squares with Local Parameterization

In this section, we combine the local parameterization and reweighted potential function together, and propose an iteratively reweighted least square algorithm with local parameterization (local IRLS) based on the previous algorithm "DINM:inc-I". The method of iteratively reweighted least squares (IRLS) is used to solve certain optimization problems with a least square objective function, by an iterative method in which each step involves solving a weighted least squares problem [wikd]. IRLS can be simply combine with LMA and GNA [wikd]. In our situation, the weighted least square problem in each iteration step is to solve (4.28) with in a bounded trust region. About more detail about IRLS, please see [HW77].
When the mode of the reweighted potential function is set to without reweighting, the algorithm is switched to DINM with local parameterization in Section 4.2. In local parameterization, the partial derivatives (4.16) and (4.17) are different related to their forms in global parameterization. Besides, a new arguments set $\boldsymbol{w}$ is used instead of arguments vector $\boldsymbol{u}$ in global parameterization. The forms of Jacobian matrix and reprojection error vector are also changed.
There are also many different variants in local IRLS as in the previous sections, such as different damping matrix, different scaling matrix, different preconditioner, etc. A complete example algorithm of local IRLS is presented in Algorithm 11.

Algorithm 10 Adaptive DINM

Require: Initial arguments vector $\boldsymbol{u}^0$ with (2.38), which contains $\{\boldsymbol{x}_i^0\}_{i=1}^{\mathrm{m}}$, $\{(\boldsymbol{\phi}_j^0, \boldsymbol{t}_j^0)\}_{j=1}^{\mathrm{n}}$, $\{\boldsymbol{y}_j^0\}_{j=1}^{\mathrm{n}}$; Observation field $(i,j) \in \mathcal{S}$; Observation dataset $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}}$; Scaling matrix $\mathbf{D_c}$ for $\boldsymbol{p_c}$; Scaling matrix $\mathbf{D_p}$ for $\boldsymbol{p_p}$; Damping matrix $\boldsymbol{D}$; Maximum iterative steps in PCG loop, $\#_{PCG}$; Maximum iterative steps in ADINM loop, $\#_{ADINM}$; Termination condition of ADINM, ξ_1, ξ_2; Initial trust region radius Δ_c^0 for $\widetilde{\boldsymbol{p}}_{\boldsymbol{c}}$; Initial damping factor μ^0;

Ensure: Optimal arguments vector $\boldsymbol{u}^k$; Optimal objective function ψ_f^k;

Initialize the objective function $\psi_f^0 \leftarrow \psi_f(\boldsymbol{u}^0)$;
Initialize step number $k \leftarrow 0$
while $k < \#_{ADINM}$ **do**
    Compute current reprojection error vector $\boldsymbol{F}^k$ with (2.37);
    Compute current objective function $\psi_f^k \leftarrow \psi_f(\boldsymbol{u}^k)$ with (2.26);
    Compute current Jacobian matrix $\boldsymbol{J}^k$ (2.40);
    Compute current gradient vector $\boldsymbol{g}^k$ with (3.42);
    if $\|\boldsymbol{g}^k\| < \xi_1$ **or** $\psi_f^k < \xi_2$ **then**
        return $\boldsymbol{u}^k$ and ψ_f^k;
    end if
    Compute current Hessian matrix $\boldsymbol{H}^k$ with (3.43);
    Set flag $update \leftarrow$ "$False$";
    while $update =$ "$False$" **do**
        $\boldsymbol{H}_{\boldsymbol{\mu}}^k \leftarrow \boldsymbol{H}^k + \mu^k \boldsymbol{D}$
        if $cond(\boldsymbol{C}) < cond(\boldsymbol{B})$ **then**
            Compute the increment step $\boldsymbol{p}^k$, and reduction of model function $-m_f^k(\boldsymbol{p}^k)$ with Algorithm 4;
        else
            $\Delta_p^k \leftarrow \frac{\Delta_c^k \times 3\mathrm{m}}{9\mathrm{n}}$; ;
            Compute the increment step $\boldsymbol{p}^k$, and reduction of model function $-m_f^k(\boldsymbol{p}^k)$ with Algorithm 4 in version of Schur complement of $\boldsymbol{B}$;
        end if
        Compute new objective function $\psi_f^{k+1} \leftarrow \psi_f(\boldsymbol{u}^{k+1})$ with (2.26);
        Compute ratio ρ^k with (3.104);
        Update trust region radius Δ_c^k with Algorithm 2;
        Update damping factor μ^k with Algorithm 6;
        if $\rho^k > 0$ **then**
            Update arguments vector $\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{u}^k + \boldsymbol{p}^k$;
            $\mu^{k+1} \leftarrow \mu^k$;
            $\Delta_c^{k+1} \leftarrow \Delta_c^k$
            $k \leftarrow k + 1$
            Set flag $update \leftarrow$ "$True$";
        end if
    end while
end while
return $\boldsymbol{u}^k$ and ψ_f^k;

Algorithm 11 Iteratively Reweighted Least Squares with Local Parameterization

Require: Initial local arguments set $\boldsymbol{w}^0$ with (4.6), which contains $\{\boldsymbol{x}_i^0\}_{i=1}^{\mathrm{m}}$, $\{(\boldsymbol{R}_j^0, \boldsymbol{t}_j^0)\}_{j=1}^{\mathrm{n}}$, $\{\boldsymbol{y}_j^0\}_{j=1}^{\mathrm{n}}$; Observation field $(i,j) \in \mathcal{S}$; Observation dataset $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}}$; Scaling matrix $\mathbf{D_c}$; Damping matrix $\boldsymbol{D}$; Weight factor γ; Maximum iterative steps in PCG loop, $\#_{PCG}$; Maximum iterative steps in IRLS loop, $\#_{IRLS}$; Termination condition of IRLS, ξ_1, ξ_2; Initial trust region radius Δ_c^0; Initial damping factor μ^0;

Ensure: Optimal local arguments set $\boldsymbol{w}^k$; Optimal reweighted objective function $\psi_{f,\bullet(\gamma)}^k$;

Initialize the objective function $\psi_f^0 \leftarrow \psi_{f,\bullet(\gamma)}(\boldsymbol{w}^0)$;
Initialize step number $k \leftarrow 0$
while $k < \#_{IRLS}$ **do**
    Compute current reprojection error vector $\boldsymbol{F}^k$ with (2.37);
    Compute current weight matrix $\mathbf{W}$ with (4.23) and (4.24);
    Compute current reweighted objective function $\psi_{f,\bullet(\gamma)}^k \leftarrow \psi_{f,\bullet(\gamma)}(\boldsymbol{w}^k)$ with (4.20);
    Compute current Jacobian matrix $\boldsymbol{J}_{\boldsymbol{l}}^k$ (4.10);
    Compute current reweighted gradient vector $\boldsymbol{g}_{\mathbf{W}}^k$ with (4.27);
    if $\|\boldsymbol{g}_{\mathbf{W}}^k\| < \xi_1$ **or** $\psi_{f,\bullet(\gamma)}^k < \xi_2$ **then**
        return $\boldsymbol{w}^k$ and $\psi_{f,\bullet(\gamma)}^k$;
    end if
    Compute current reweighted Hessian matrix $\boldsymbol{H}_{\mathbf{W}}^k$ with (4.26);
    Set flag *update* $\leftarrow$ "$False$";
    while *update* = "$False$" **do**
        Compute reweighted damped Hessian matrix $\boldsymbol{H}_{\mathbf{W},\boldsymbol{\mu}}^k \leftarrow \boldsymbol{H}_{\mathbf{W}}^k + \mu^k \boldsymbol{D}$
        Compute the local increment step $\boldsymbol{p}_{\boldsymbol{l}}^k$ in (4.8), and reduction of model function $-m_f^k(\boldsymbol{p}_{\boldsymbol{l}}^k)$ with Algorithm 4;
        Compute new objective function $\psi_{f,\bullet(\gamma)}^{k+1} \leftarrow \psi_{f,\bullet(\gamma)}(\boldsymbol{w}^{k+1})$ with (4.20);
        Compute ratio ρ^k with (3.104);
        Update trust region radius Δ_c^k with Algorithm 2;
        Update damping factor μ^k with Algorithm 6;
        if $\rho^k > 0$ **then**
            Update local arguments set $\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k \boxplus \boldsymbol{p}^k$ with (4.9);
            $\mu^{k+1} \leftarrow \mu^k$;
            $\Delta_c^{k+1} \leftarrow \Delta_c^k$
            $k \leftarrow k+1$
            Set flag *update* $\leftarrow$ "$True$";
        end if
    end while
end while
return $\boldsymbol{w}^k$ and $\psi_{f,\bullet(\gamma)}^k$;

Chapter 5

Experiments and Evaluation

In this chapter, firstly, we provide a novel method to construct a controllable synthetic dataset for bundle adjustment. Then, we implement four typical algorithms presented in this thesis and two baseline algorithms in Matlab. At last, we evaluate the performance between these algorithms on different datasets.

5.1 Synthetic Bundle Adjustment Dataset

In this section, we provide a method which generates a synthetic bundle adjustment dataset with defined size and defined sparsity. The noise of observations, the outliers of observations and the initial point are also controllable. Besides, this synthetic dataset contains the ground truth which overcomes the shortage of the datasets provided by [ASSS].

5.1.1 Feature Points

Considering the convenience of the later verification, we select a ball with radius r as the real 3D object, whose center is located on the origin point of system O. All feature points are randomly distributed on the surface of the ball. However, if one feature point does not appear in at least two image frames, this feature point is deleted. The number of feature points can be arbitrarily determined to control the sparsity and the scale of the bundle adjustment.

5.1.2 Camera and Image Frames

Still, the pinhole camera with circle convex lens is chosen. The focus length of the convex length f is recommended to be set to a smaller value than r. The radial distortion factors, k_1 and k_2, are set to a smaller value, around 10^{-2}, and 10^{-4} respectively.
The camera location is presented as C, which is also the origin point of camera system C. C must be located outside of the ball, and not far away from the ball,

so that the observed feature points are not distributed only in a small area in the image frame. In other words, the image frame should be covered by the observed feature points as much as possible. Each image frame is also bounded by a certain rectangle as usual, e.g. a rectangle with the size of 800×600. The image is framed on the plane $(0, 0, -f)$ in camera system C with the center point of $\begin{pmatrix} 0 \\ 0 \\ -f \end{pmatrix}$.

To ensure enough feature points can be captured by the camera, the intersection angle between $\overrightarrow{OC}$ and $\boldsymbol{n}_{Cz}$ (the unit vector of z-axis in system C) is limited smaller than 30°. Only the feature points located on the surface of a certain spherical cap are visible to the camera. These feature points are perspective projected on the plane of $(0, 0, -f)$ in camera system C, seeing Section 2.1.1. The projected feature points, which are located in the image frame (rectangle defined before), can be captured by this image frame, i.e. become observed feature points. To help understand this process, an example is given in Figure 5.1 to demonstrate the structure of the camera and the 3D ball (object).

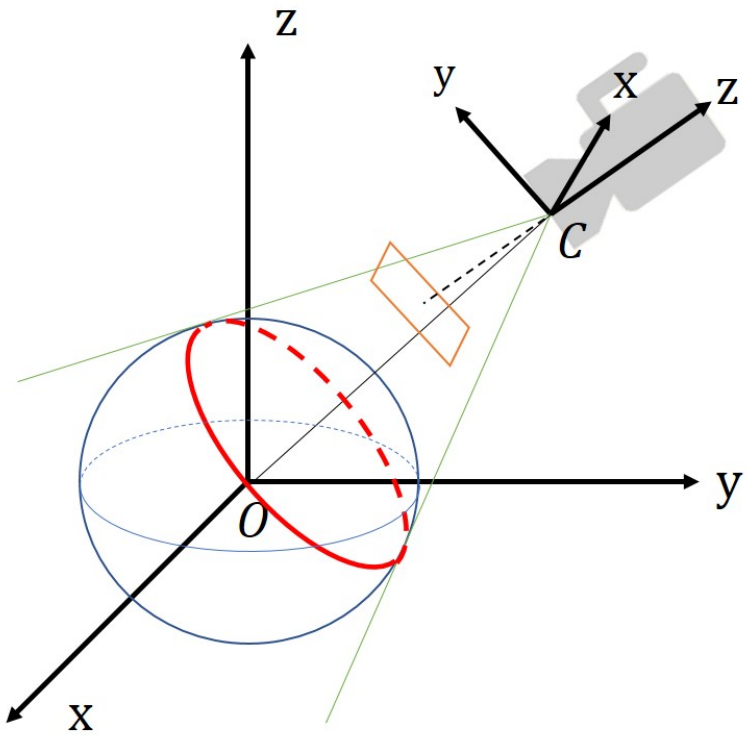

Figure 5.1: The structure of constructing a synthetic dataset. The orange parallelogram presents the image frames. The red circle defines a spherical cap, in which the feature points are visible for the current camera.

The locations and the orientations of the camera are also randomly chosen, but meet the above requirements. Besides, if one image frame does not capture enough

feature points, or its area covered by the observed feature points is not large enough, this image frame is discarded. The range of the camera's locations, the number of the image frames can be also arbitrarily determined to control the sparsity and the scale of the dataset.

The image frames should also be arranged, such that the sub-matrix $\boldsymbol{E}$ in Hessian matrix is block diagonal dominant distributed, which makes some matrix operations in optimization algorithm more efficient [TMHF99]. $\boldsymbol{E}$ is block diagonal dominant distributed means the successive image frames have a large overlap in commonly observed feature points. In other words, the image frames with a similar viewpoint should be arranged together. The distances between every two camera locations are computed, and the indexes of image frames are sorted according to these distances. The order of observations is also updated such that its corresponding camera indexes and its corresponding feature point indexes have a similar structure as the dataset provided by [ASSS].

A example of generated synthetic dataset with 100 image frames and 5000 feature points is plotted in Figure 5.2.

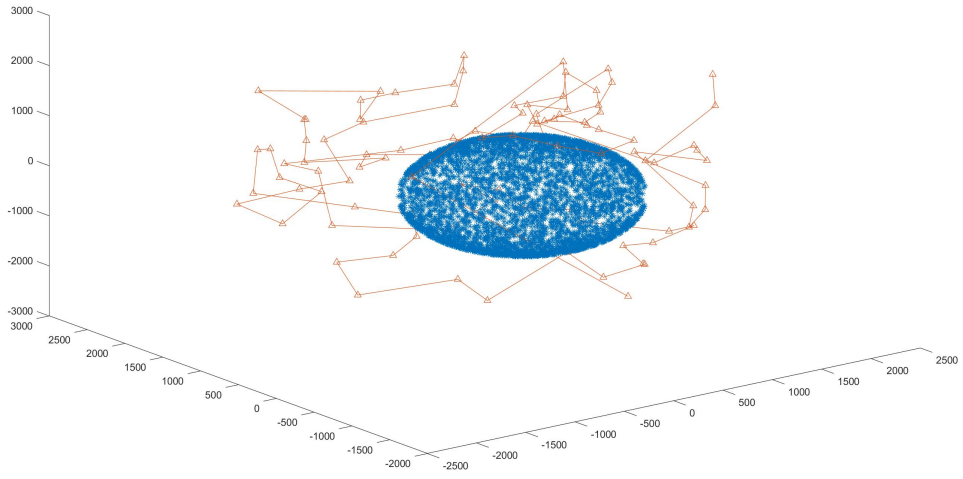

Figure 5.2: Synthetic dataset with 100 image frames and 5000 feature points in system O. The feature points are presented by blue "*"; and the camera locations are presented as red "$\triangle$". All camera locations are connected together with red line according to its arranged order.

5.1.3 Noises and Outliers

Based on the valid feature points and the valid camera positions acquired above, the reprojected feature point positions in image frames can be easily calculated according to the formulas in Section 2.1.1 as the observations. The original feature point coordinates in system O, and the camera parameters (camera pose parameters and camera characteristic parameters) are collected as the ground truth.
To test the robustness of the optimization algorithms, we also add some noises and outliers in the observations, such that even if with ground truth, the reprojection error is still not equal to zero.

5.1.4 Initial Value

The initial arguments of the optimization are obtained through adding a small Gauss distributed variation onto the ground truth. The arguments which is closed to zero, e.g. k_1 and k_2, are set to zero in initialization.

5.2 Experiment implementations in Matlab

In this section, the experiment's implementation details in Matlab programming are presented.

Pseudo Inverse

Since the matrices in bundle adjustment are often near singular and bad conditioned, using the general command *inv* to compute the inverse of matrix in Matlab gets an inexact solution sometimes, or even generates a matrix with NaN or Inf. Therefore, the pseudo inverse is used to compute the inverse of matrix in all implementations of the thesis.
In linear algebra, a pseudo inverse of a matrix is a generalization of the inverse matrix. The term pseudo inverse is often used to indicate the Moore-Penrose inverse [wikf]. Pseudo inverse finds a "best fit" (least squares) solution to an inverse of a matrix, especially if the matrix is singular. The pseudo inverse is defined and unique for all matrices whose entries are real or complex numbers [wikf]. It can be computed using the singular value decomposition. For more details about pseudo inverse, please see [GK65].
The command *pinv* in Matlab is used to compute Moore-Penrose pseudoinverse of a matrix.

Virtual Memory

Since some general commands costing real memory in Matlab, e.g. *for*-loop, *if-else*, *repmat* are inefficiency in computation, the Matlab commands, such as, *bsxfun*,

arrayfun, *cellfun*, etc. are used to replace these general commands in all implementations during the whole thesis.
bsxfun applies an element-by-element binary operation to two arrays with singleton expansion enabled. [matb]
arrayfun applies the function to the elements of input arrays, one element at a time, then it concatenates the outputs from function into the output array. The output from function can have any data type, so long as objects of that type can be concatenated. [mata]
cellfun applies the function to the contents of each cell of input cell arrays, one cell at a time, then it concatenates the outputs from function into the output array. The output from function can have any data type, so long as objects of that type can be concatenated. [matc]
These above commands save much more time and memory than *for*-loop, *if-else*, *repmat*, etc. When the input data array is in large-scale, since these commands only apply for a virtual memory for computation, and computation is done by kernel of Matlab, it is much faster than other commands.
A comparing of execution time between *bsxfun* and *repmat* is demonstrated in Figure 5.3.

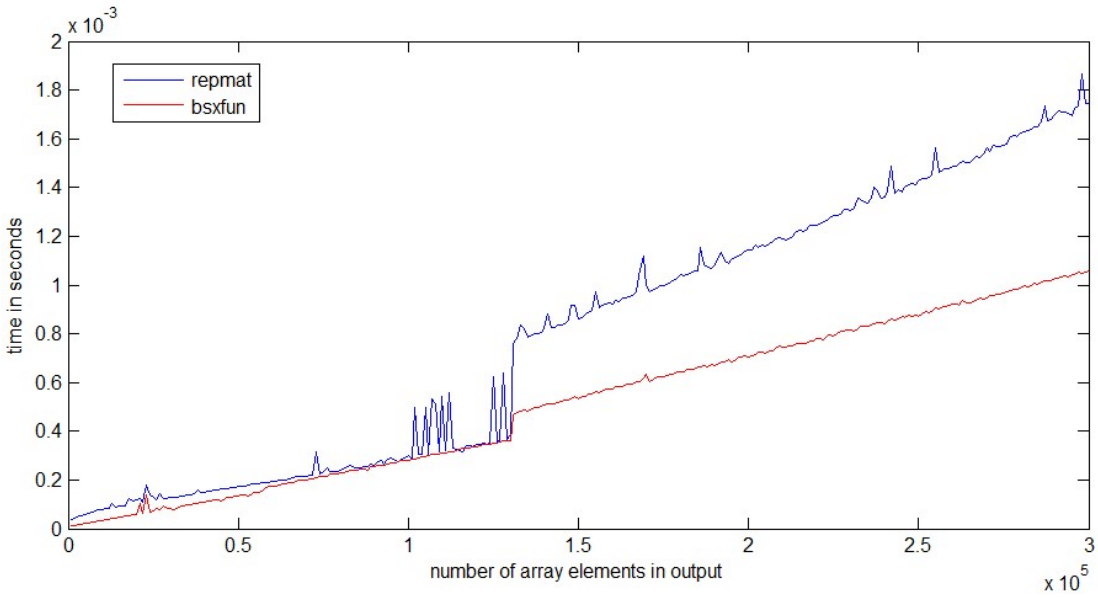

Figure 5.3: Execution time comparing between *repmat* and *bsxfun* in Matlab along the increasing of the scale of the input data. [Jon] *bsxfun* is used in this thesis.

5.3 Evaluation

We implement four typical algorithms proposed in this thesis and two baseline implementations proposed in [TMHF99] and [ASSS10], listed below respectively.

- "DINM": DINM algorithm couple with SC of $\boldsymbol{C}$, "inc" scaling matrix, identity damping matrix, without preconditioner.
- "ADINM": Adaptive DINM algorithm with "inc" scaling matrix, identity damping matrix, without preconditioner.
- "local DINM": DINM algorithm coupled with SC of $\boldsymbol{C}$, local parameterization with "inc" scaling matrix, identity damping matrix, without preconditioner.
- "IRLS": Iteratively Reweighted Least Squares (Huber loss function) couple with SC of $\boldsymbol{C}$, local parameterization, based on "DINM:inc-I".
- "LMA-CD": LMA couple with SC of $\boldsymbol{C}$ solved by CD [TMHF99], seeing more details in Section 3.2.2.
- "BAL": TNM proposed in [ASSS10], seeing more details in Section 1.1.

The above algorithms are tested in the Ladybug dataset provided by [ASSS] and synthetic datasets generated by the process in Section 5.1. In order to better comparing the performance of different algorithms, we generate the bundle adjustment datasets with different scales and sparsities.

5.3.1 Reprojection Error Plots

To demonstrate the effectiveness of the algorithms, we plot the reprojection errors before optimization (initialization) and after optimization. Since the objective function sums up the reprojection errors, and the optimization aims to find a local minimum of the objective function, the differences between the observations and the reprojected points in each frame are the important index to indicate the performance of the algorithms.

Reprojection Error Plots on Ladybug Dataset

Firstly, the implementations are tested based on the Ladybug dataset provided by [ASSS], which is also the dataset we used in the previous sections. We select the most typical algorithm in this thesis, "DINM:inc-I" and run the algorithm for 50 iteration steps. More details about the setting of algorithm, please see Section 3.3.5. Since there are altogether 49 image frames in dataset, it is impossible and unnecessary to plot them all. We randomly chose two image frames, and plot the reprojection errors in both frames before and after 50 optimization steps, seeing Figure 5.4.
From the reprojection error plots, it is obvious that after 50 iteration steps of "DINM:inc-I", the reprojected points have already fitted to the observations much better than initialization. This result indicates the optimization process is effect.

Figure 5.4: The reprojection errors on the Ladybug dataset are compared between the initialization and after 50 iterative optimization steps from "DINM:inc-I". The results are plotted in two randomly chosen frames. The red circles present the observations; the blue stars present the reprojected feature points; the green lines between circles and stars present the reprojection errors. The first column presents the reprojection at the initialization; the second column presents the reprojection after 50 iteration steps.

Reprojection Error Plots on Synthetic Datasets

Secondly, we generate a synthetic dataset with 10 image frames, 253 feature points and 1326 observations. The implementation of "DINM:inc-I" also runs 100 iteration steps. The reprojection error in all 10 image frames are plotted, seeing Figure 5.5. During 100 iterations, the objective function reduces from 2.44×10^5 to 25.2; the gradient norm reduces from 4.12×10^{12} to 6.53. The average reprojection error of each observation after optimization is $\frac{25.24}{1326} = 0.019$, and the average gradient norm of each argument is $\frac{6.53}{10\times9+253\times3} = 0.0077$. Both orders are in a low level, which can be considered as yielding a local optimal.

5.3.2 Performance Comparison

In this section, the performance of our implementations and baseline implementations is compared. Like we have talked in Section 1.2, the performance is evaluated from three parts.

Accuracy: accuracy is evaluated by the order of magnitude of the objective function and the gradient norm after optimization, since all implementations run with the same initial point. Besides, for synthetic datasets, the optimized arguments are also compared with the ground truth, to present how closed the optimization trajectory approaches the ground truth.

Efficiency: time efficiency is evaluated by the rate of descent of the objective function and the gradient norm. The memory efficiency is dependent on the memory consumption during the optimization.

Robustness: robustness is evaluated if the algorithms can smoothly and effectively run on different datasets.

Among these six implementations, "LMA-CD" spends the most memory in computation, since it needs to explicitly construct the matrix $\boldsymbol{S_C}$ for decomposition. Other implementations with PCG solver use some matrix-vector multiplications instead of this explicit construction. Furthermore, decomposition itself costs more memory than PCG in solving LSE [NW06].

Real Dataset (Small-Scale)

Firstly, the implementations run on the real dataset, the Ladybug dataset provided by [ASSS], which is also the dataset used in the previous sections. Actually, the results of some implementations have already been demonstrated in Chapter 3 and Chapter 4.

All implementations run for 500 seconds. The maximum iteration step used in PCG loop is 3000. However, "IRLS" algorithms can only run around 20 steps. Then, it computes an increment step with a NaN and/or Inf value, so that the optimization process is terminated. The reason of this problem is that after several steps, the matrices in iteration become near singular or bad conditioned, and "IRLS" are

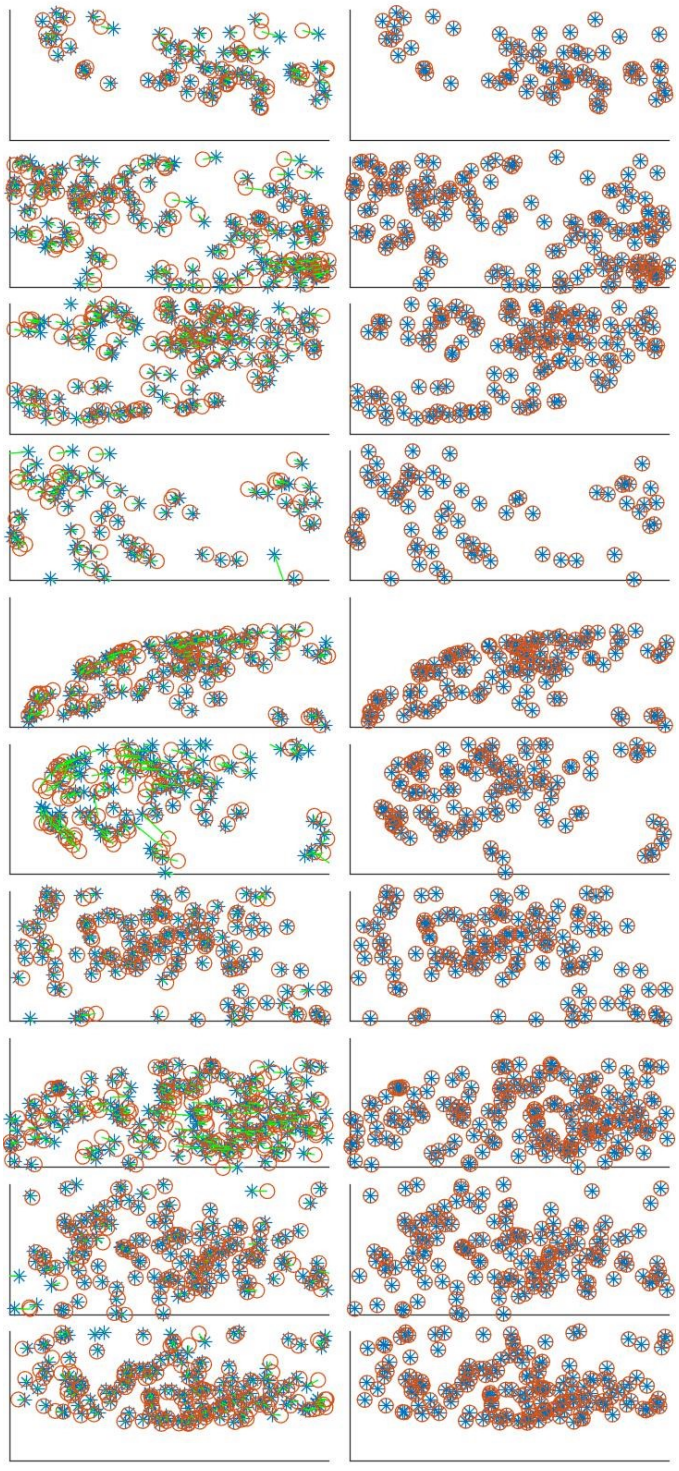

Figure 5.5: The reprojection errors on the small-scale synthetic dataset. The results are plotted in all 10 image frames. The first column presents the reprojection at the initialization; the second column presents the reprojection after 100 iteration steps of "DINM:inc-I".

too unrobust to get a reasonable solution. The objective function curves and the gradient norm curves are plotted in Figure 5.6 and Figure 5.7.
The curves of "DINM" and "ADINM" are almost the same in the first several steps, because the condition number of $\boldsymbol{C}$ is smaller than $\boldsymbol{B}$ at the beginning, and both implementations are identical. For this Ladybug dataset, we can conclude from the figures, that all implementations except for "BAL" and "IRLS" have a similar and fast reduction in objective function; "IRLS" yields the lowest objective function curve, but the potential from Huber loss function is always smaller equal to the original square function; the gradient norm of "IRLS" is high and with large fluctuation; "LMA-CD" obtains the steepest decline of gradient norm, however, its gradient norm does not decrease anymore after around 100 seconds; "ADINM" generates the smallest value of gradient norm (smaller than 1); "local DINM" has a faster reduction in gradient norm than "DINM".
All taken into account, "ADINM" and "local DINM" have a better performance than others. Both implementations have a fast and deep decline in objective function and gradient norm.

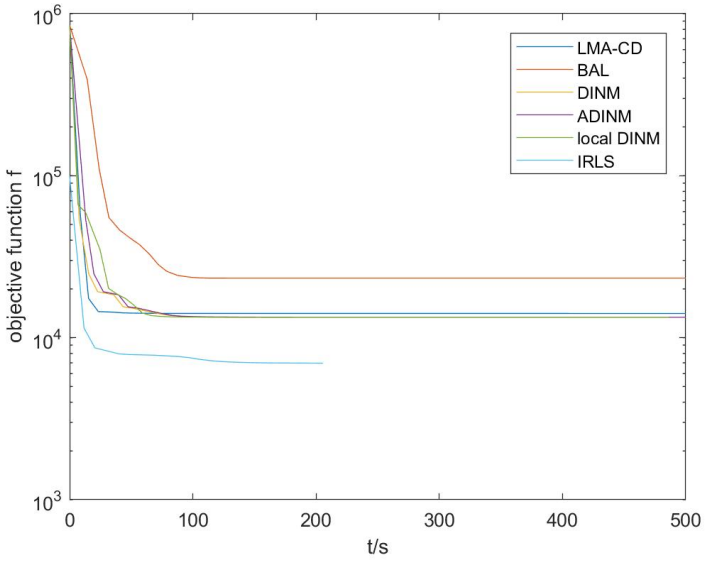

Figure 5.6: Comparison of the objective function curves between our implementations and baseline implementations running on the Ladybug dataset provided by [ASSS] with 49 image frames, 7776 feature points, and 31843 observations. "DINM", "ADINM", and "local DINM" obtain the smallest value of the objective function.

Synthetic Dataset (Small-Scale)

In this section, we test the implementations on synthetic datasets.

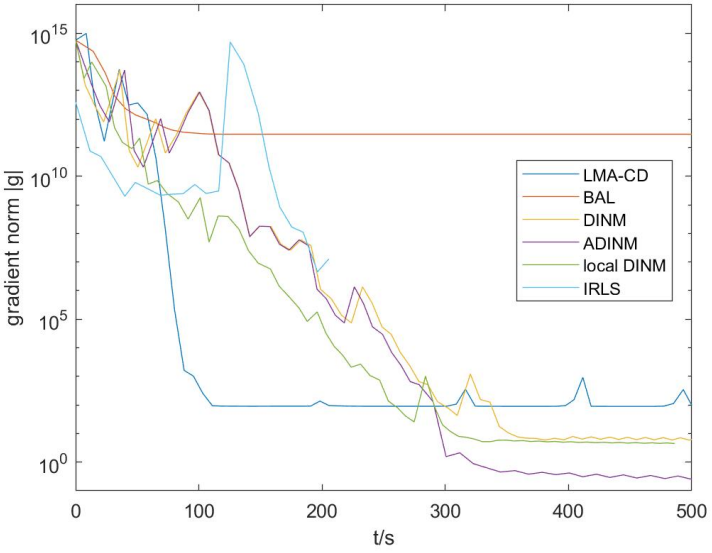

Figure 5.7: Comparison of the gradient norm curves between our implementations and baseline implementations running on the Ladybug dataset provided by [ASSS] with 49 image frames, 7776 feature points, and 31843 observations. "ADINM" yields the deepest reduction curve of the gradient norm.

Firstly, a small-scale dataset is generated with 10 image frames, 289 feature points, and 1834 observations. The schematic diagram of this dataset is plotted in Figure 5.8. The objective function curves and the gradient norm curves are plotted in Figure 5.9 and Figure 5.10. Each implementation runs on this dataset for 50 seconds. The maximum iteration step used in PCG loop is 1000. However, "LMA-CD" and "IRLS" algorithms can only run around 10 steps and around 30 steps. Then, both implementations compute an increment step with a NaN and/or Inf value, so that the optimization process is terminated. Both implementations are not robust enough facing such a dataset.

The curves of "DINM" and "ADINM" are almost the same. This is because the condition number of $\boldsymbol{C}$ is always smaller than $\boldsymbol{B}$ in the first several steps on this dataset. For this small-scale dataset, we can conclude from the figures, that in the first 1 seconds, baseline implementation "LMA-CD" yields a faster reduction in gradient norm than our algorithms; after one seconds, our implementation "DINM" ("ADINM") obtains the steepest decline in both gradient norm and objective function; finally, "DINM" also owns the smallest value of gradient norm and objective function. Even if "local DINM" also has a fast decline of objective function, its gradient norm has a great oscillation almost all the time. "ADINM" and "DINM" perform better than others on this dataset.

In order to compare the optimized arguments with the ground truth, we introduce a metric, absolute trajectory error (ATE). The ATE is defined as the root mean

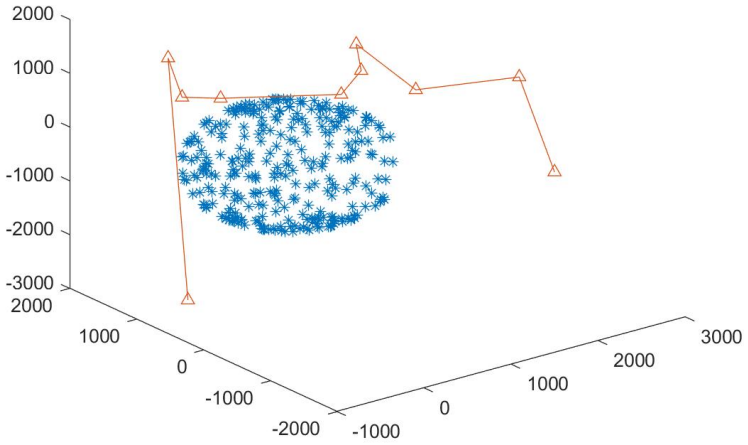

Figure 5.8: Schematic diagram of the small-scale synthetic dataset with 10 image frames, 289 feature points, and 1834 observations.

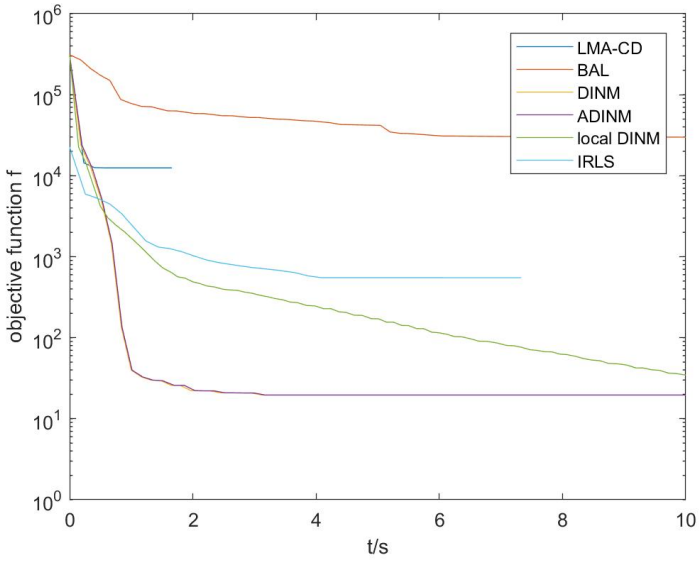

Figure 5.9: Comparison of the objective function curves between our implementations and baseline implementations running on the small-scale synthetic dataset with 10 image frames, 289 feature points, and 1834 observations. "DINM" and "ADINM" have the fastest and the deepest decline of the objective function.

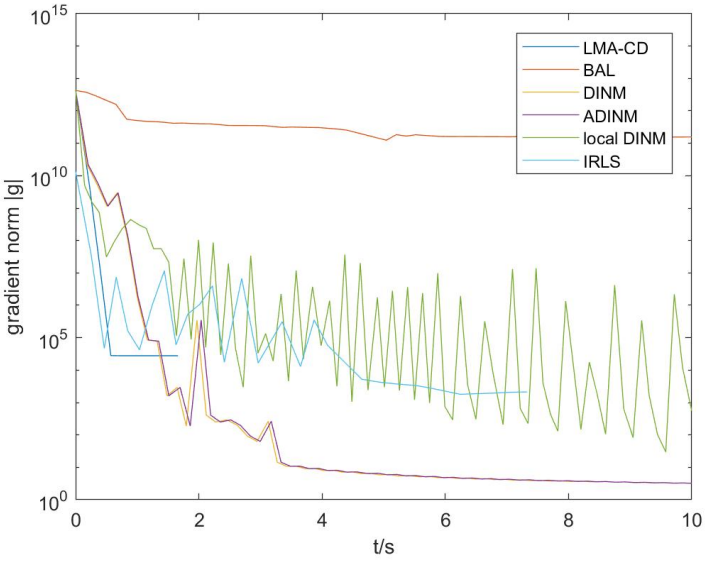

Figure 5.10: Comparison of the gradient norm curves between our implementations and baseline implementations running on the small-scale synthetic dataset with 10 image frames, 289 feature points, and 1834 observations. "DINM" and "ADINM" have the deepest and the most stable decline of the gradient norm.

square error (RMSE) in translational between two point clouds after Sim(3) alignment, i.e. alignment with translation, rotation and scaling [EUC16]. We directly use the Matlab code of RMSE provided by [git] to compute RMSE. Here, we separately compute two RMSEs for camera positions and feature points positions respectively. How small the RMSE for feature points is presents how close the estimated feature points is to the ground truth of feature points, similar for the RMSE for camera positions. Camera positions are computed with rotation matrices and translation vectors, thus, RMSE for camera positions implies the correctness of rotation parameters and translation parameters of each image frame.
The RMSEs at the initialization and after the optimizations of each implementation are presented in Table 5.1.

Table 5.1: RMSE among implementations on the small-scale synthetic dataset. The results from "local DINM" are most closed to the ground truth.

Implementation	initial	LMA-CD	BAL	DINM	ADINM	local DINM	IRLS
RMSE Camera/1	75.57	3.15	1591	3.86	3.86	3.57	28.17
RMSE Feature/1	14.01	13.74	3805	4.22	4.22	3.18	51.06

From the above table, both (adaptive) DINM and DINM with local parameterization have a better approximation to the ground truth than others after optimization. IRLS performs even worse than implementations without reweighted potential on

the small-scale dataset.

Synthetic Dataset (Medium-Scale)

Secondly, a medium-scale dataset is generated with 50 image frames, 1000 feature points, and 28991 observations. The schematic diagram of this dataset is plotted in Figure 5.11. The objective function curves and the gradient norm curves are plotted in Figure 5.12 and Figure 5.13. Each implementation run on this dataset for 300 seconds. The maximum iteration step used in PCG loop is 2000. However, "LMA-CD" and "IRLS" can only run around 30 steps. Then, they compute an increment step with a NaN and/or Inf value again, similar as before.
For this medium-scale dataset, we can conclude from the figures, that all our algorithms always yield a faster and deeper reduction of objective function than both baseline implementations after 50 seconds; the curves of "DINM" and "ADINM" are still almost the same in the first 30 seconds; after 50 seconds, "ADINM" has the largest decreasing of the gradient norm and the objective function among all implementations, even if with a little fluctuation; "local DINM" also gets a satisfied decline curve, and also more stable than "ADINM". "local DINM" and "ADINM" are the best implementations.

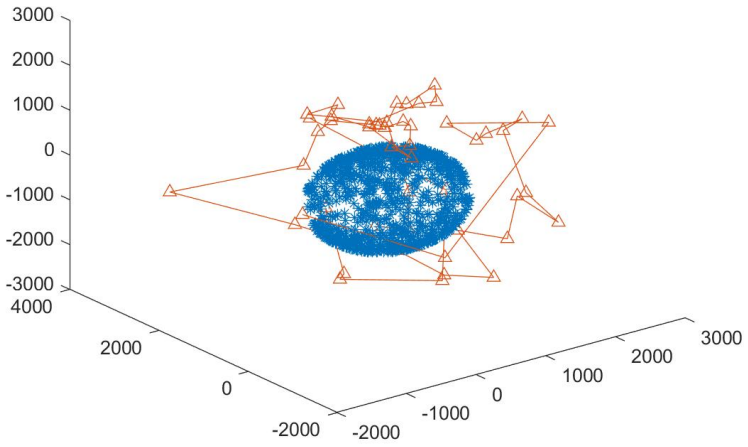

Figure 5.11: Schematic diagram of the medium-scale synthetic dataset with 50 image frames, 1000 feature points, and 28991 observations.

The RMSEs at the initialization and after the optimizations of each implementation are presented in Table 5.2.
From the above table, both (adaptive) DINM and DINM with local parameterization have a better approximation to the ground truth than baseline implementations.

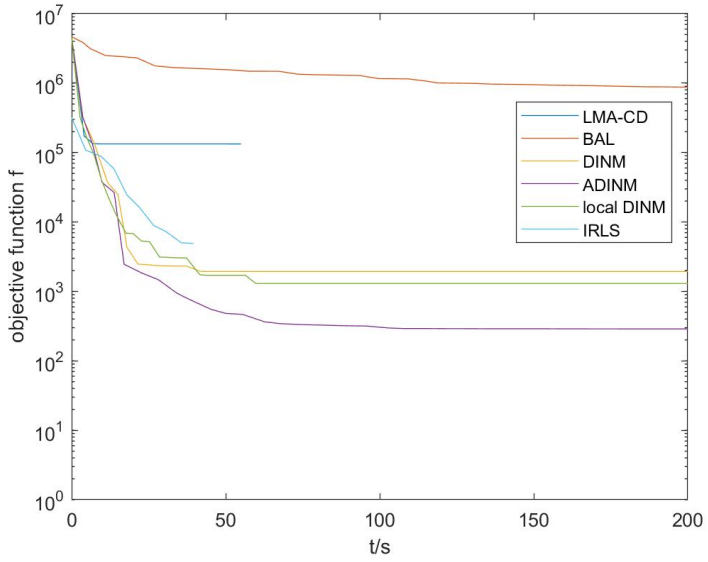

Figure 5.12: Comparison of the objective function curves between our implementations and baseline implementations running on the medium-scale synthetic dataset with 50 image frames, 1000 feature points, and 28991 observations. "ADINM" has the fastest and the deepest decline of the objective function.

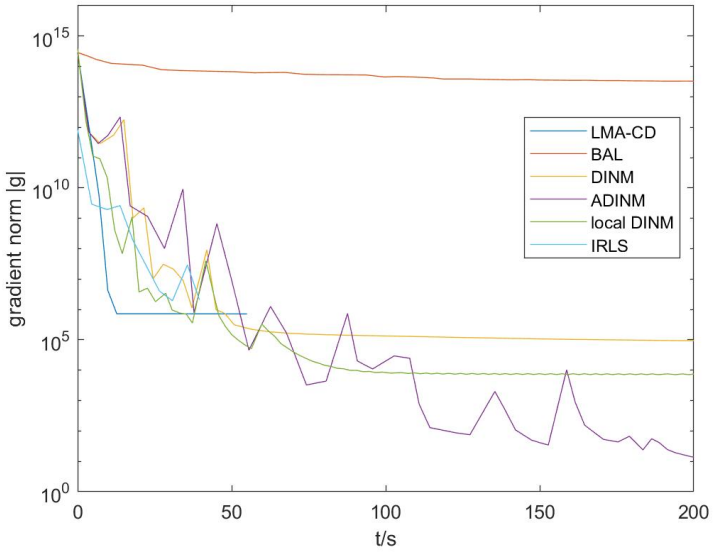

Figure 5.13: Comparison of the gradient norm curves between our implementations and baseline implementations running on the medium-scale synthetic dataset with 50 image frames, 1000 feature points, and 28991 observations. "ADINM" has the fastest and the deepest decline of the gradient norm.

Table 5.2: RMSE among implementations on the medium-scale synthetic dataset. The results from "DINM" are most closed to the ground truth.

Implementation	initial	LMA-CD	BAL	DINM	ADINM	local DINM	IRLS
RMSE Camera/1	44.14	5.80	31.48	4.68	1.49	6.85	11.76
RMSE Feature/1	11.78	10.37	58.37	2.10	13.14	6.85	4.43

Synthetic Dataset (Larger Medium-Scale)

Thirdly, a larger medium-scale dataset is generated with 100 image frames, 4000 feature points, and 228658 observations. The schematic diagram of this dataset is plotted in Figure 5.14. The objective function curves and the gradient norm curves are plotted in Figure 5.15 and Figure 5.16. Each implementation runs on this dataset for 1000 seconds. The maximum iteration step used in PCG loop is 3000. However, "LMA-CD" algorithm still only runs around 10 steps, and gains an increment step with a NaN and/or Inf value.
The curves of "DINM" and "ADINM" are still almost the same. For this larger medium-scale dataset, we can conclude from the figures, that all our algorithms obtain a faster and deeper decline in objective function curves and gradient norm curves than both baseline implementations, except for "DINM"; "local DINM" still yields the lowest objective function and gradient norm among all implementations; "IRLS" also performs nearly as good as "local DINM"; "DINM" ("ADINM") has a steeper reduction in objective function than baseline implementations, however, its gradient norm decreases a little slower than "LMA-CD". Still, "local DINM" performs the best.
The RMSEs at the initialization and after the optimizations of each implementation are presented in Table 5.3.

Table 5.3: RMSE among implementations on the larger medium-scale synthetic dataset. The results from "local DINM" and "IRLS" are most closed to the ground truth.

Implementation	initial	LMA-CD	BAL	DINM	ADINM	local DINM	IRLS
RMSE Camera/1	42.07	4.56	26.23	4.56	4.56	0.33	1.08
RMSE Feature/1	6.00	6.13	72.82	1.26	1.26	1.52	1.35

From the above table, both "IRLS" and "DINM" with local parameterization have a far better approximation to the ground truth than other implementations. However, the reweighted potential function still does not gain a significant improvement.

Synthetic Dataset (Large-Scale)

Finally, a large-scale dataset is generated with 500 image frames, 5000 feature points, and 1477513 observations. The schematic diagram of this dataset is plotted in

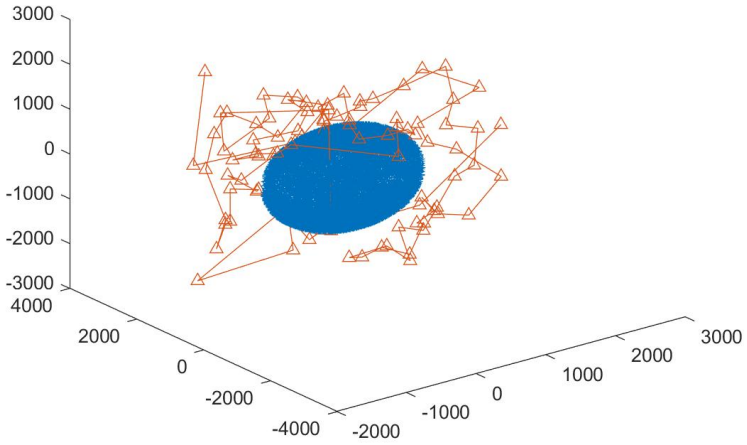

Figure 5.14: Schematic diagram of the larger medium-scale synthetic dataset with 100 image frames, 4000 feature points, and 228658 observations.

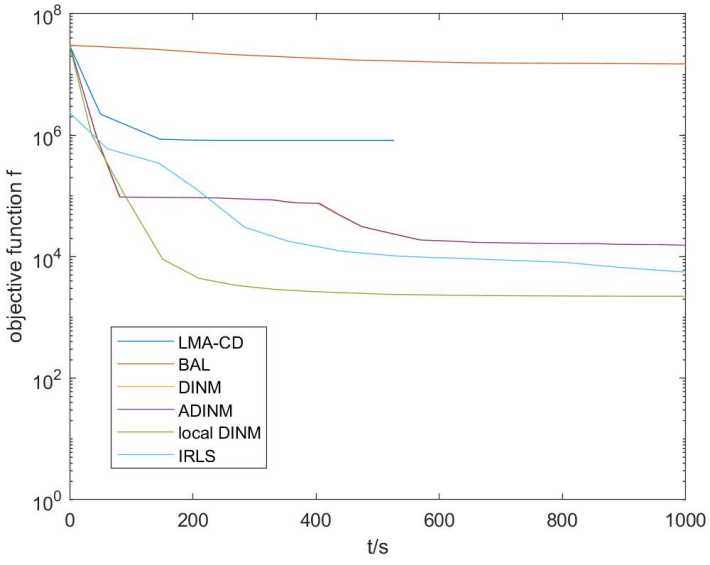

Figure 5.15: Comparison of the objective function curves between our implementations and baseline implementations running on the larger medium-scale synthetic dataset with 100 image frames, 4000 feature points, and 228658 observations. "local DINM" has the fastest and the deepest decline of the objective function.

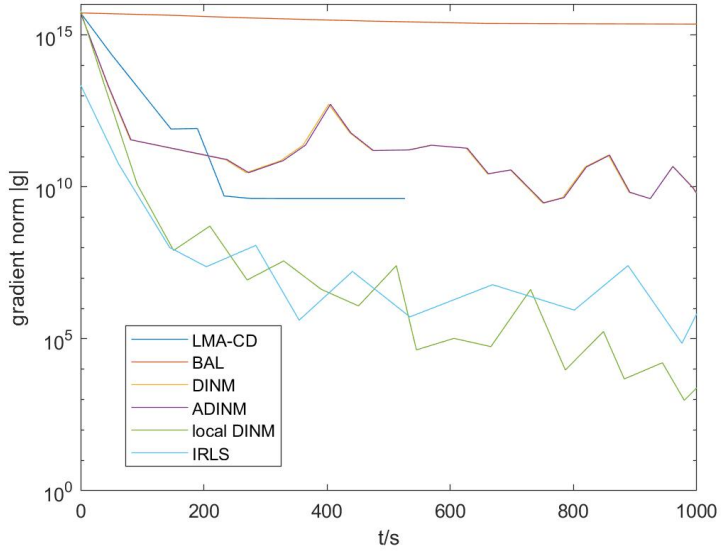

Figure 5.16: Comparison of the gradient norm curves between our implementations and baseline implementations running on the larger medium-scale synthetic dataset with 100 image frames, 4000 feature points, and 228658 observations. "local DINM" has the fastest and the deepest decline of the gradient norm.

Figure 5.17. The objective function curves and the gradient norm curves are plotted in Figure 5.18 and Figure 5.19. Each implementation runs on this dataset for 5000 seconds. The maximum iteration step used in PCG loop is 5000. "LMA-CD" algorithm can also run for 5000 seconds, but this is because each step costs too much time than before. It is unknown if "LMA-CD" faces the unrobust problem after first 5000 seconds or not.
For this large-scale dataset, we can conclude from the figures, that all our algorithms yield a faster and deeper decline in objective function curves and gradient norm curves than both baseline implementations; "local DINM" wins again among all implementations.
The RMSEs at the initialization and after the optimizations of each implementation are presented in Table 5.4.

Table 5.4: RMSE among implementations on the large-scale synthetic dataset. The results from "IRLS" are most closed to the ground truth.

Implementation	initial	LMA-CD	BAL	DINM	ADINM	local DINM	IRLS
RMSE Camera/1	26.42	4.50	36.17	9.22	9.22	4.38	1.35
RMSE Feature/1	9.08	4.10	64.16	6.14	6.14	9.94	5.92

From the above table, "IRLS" has a better approximation to the ground truth than other implementations. Besides, the baseline implementation of "LMA-CD" also

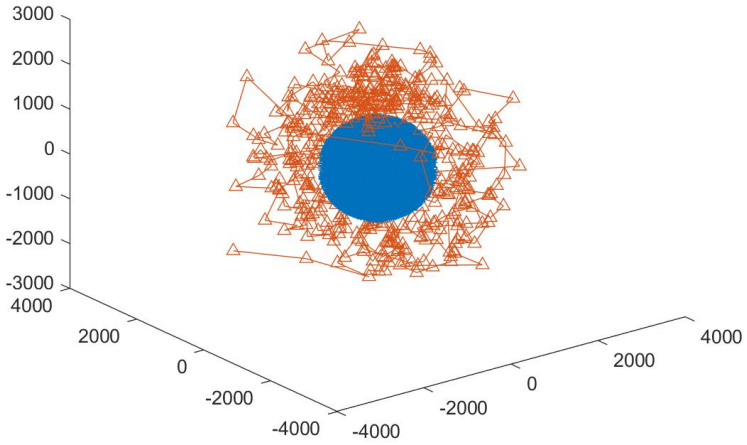

Figure 5.17: Schematic diagram of the large-scale synthetic dataset with 500 image frames, 5000 feature points, and 1477513 observations.

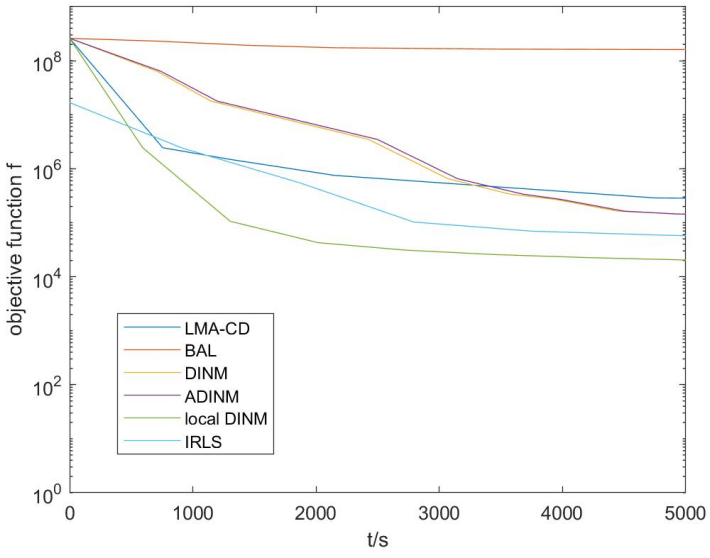

Figure 5.18: Comparison of the objective function curves between our implementations and baseline implementations running on the large-scale synthetic dataset with 500 image frames, 5000 feature points, and 1477513 observations. "local DINM" has the fastest and the deepest decline of the objective function.

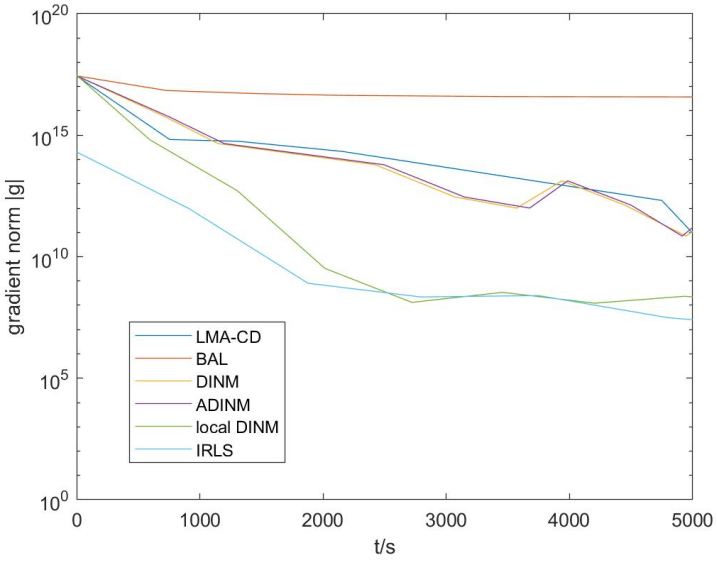

Figure 5.19: Comparison of the gradient norm curves between our implementations and baseline implementations running on the large-scale synthetic dataset with 500 image frames, 5000 feature points, and 1477513 observations. "local DINM" has the fastest and the deepest decline of the gradient norm.

yields a good approximation. Even if "local DINM" owns the best decline curves, its results are not as closed to the ground truth as "IRLS". However, since for this large-scale dataset, we only run 5000 seconds for each implementation (less than 10 steps in general), the estimated arguments are actually far from the respected local optimal. This comparison does not reveal enough information.

Chapter 6

Discussion

The comparison plots in Section 5.3 tell us the following points.

- As mentioned in [ASSS10], dense Cholesky decomposition is still the most popular choice to solve small-scale bundle adjustment or sparse bundle adjustment due to its precise solution and small calculation burden. However, when facing the bad conditioned and/or near singular problem, "LMA-CD" is always not robust enough. Under this situation, the exact LMA step solved by Cholesky decomposition is extremely inaccurate, or even appearing NaN/Inf. This situation also happens in the small-scale situation.

- Inexact increment step solved by PCG is much more robust than CD solver. Besides, when solving a large equation system, PCG can compute a solution with satisfied accuracy much faster than CD. An exact solution is unnecessary and also pretty computation burden in optimization step.

- Local parameterization yields a more quick and more accurate optimization step related to global parameterization, especially on large-scale dataset. On small-scale dataset, the global parameterization may have a better performance.

- The comparing results with the ground truth indicate all our algorithms estimate the arguments more towards to the ground truth related to baseline implementations generally. Especially for medium-/large-scale datasets, "IRLS" can better approach the ground truth by avoiding the outliers problem.

- (Adaptive) DINM algorithm and DINM with local parameterization can almost beat "LMA-CD" and "BAL" in all sizes of datasets considering accuracy, efficiency and robustness. "IRLS" algorithm performs well in many situations, but it is also not robust enough sometimes, which needs to be further improved.

- The testing results using synthetic datasets show that "ADINM" and "DINM" always have the same reduction curves in the first several iterations. On the

other hand, it tells us that the judging of adaptive switching mode does not cost much time related to other operations. Therefore, Adaptive DINM is more suggested to implement, cause the adaptive switching ensure a more robust computation.

In addition, there are some details about our algorithms to mention.

- An enough large maximum iteration steps in PCG loop of DINM is a crucial factor, which helps to gain a iteration step, not only preciser but also faster.
- "inc" scaling matrix and identity damping matrix are the best group for DINM in our testing. Nevertheless, since we do not test all possible selection of both matrices, there may exist better choice for DINM, at least on some datasets.
- The selection of preconditioner is sensitive and confused. Due to the lack of enough knowledge of preconditioner, we do not use preconditioner in the main experiments.
- An implicit cell-structure of matrices saves lots of time and memory during construction in bundle adjustment.

We also have some recommendations for the future research.

- The implementations should be tested on more real datasets with different sizes and different sparsities.
- In our thesis, we do not find a proper preconditioner for CG solver in DINM algorithm, which can significantly improve the performance. We suggest the following researchers should firstly have a deeper comprehension of the preconditioner, and then try to find a robust preconditioner for DINM algorithm.
- Finally, the implementation "IRLS" is still not robust enough in some situations, even if it is built based on "local DINM". The reason comes from the reweighted potential function, i.e. the reweighted objective function deteriorates the robustness. The following researchers are suggested to find how the condition number of matrix is influenced by the reweighted potential function.

Chapter 7

Conclusion

In this thesis, we propose a novel damped inexact Newton's method, which introduces the damped factor into inexact Newton's method. Thus, in damped inexact Newton's method, an inexact Levenberg-Marquardt step solved by (preconditioned) conjugate gradient is bounded by a (scaled) trust region. DINM algorithm combines the advantages from inexact Newton's method and truncated Newton's method. In comparison with both baseline implementations, DINM yields a faster and deeper decline curve of the objective function and the gradient norm. DINM algorithm also presents more robustness on different datasets than baseline implementations, especially "LMA-CD".
About DINM algorithm, we compare different matrix forms, different couples of the scaling matrix and the damping matrix, different preconditioner, and different maximum steps in PCG loop. The results imply that the DINM implementation with implicit cell-structure, "inc" scaling matrix formed from (3.38), identity damping matrix, and without preconditioner has the best performance. Besides, selecting an enough large maximum steps in PCG loop is important to the accuracy and the efficiency of the optimization process.
In addition, we propose three extended algorithms based on DINM. ADINM algorithm adaptively switches Schur complement modes according to the condition number, which is more robust than original DINM, and also obtains a deeper reduction of the gradient norm. "local DINM" utilizes the local parameterization instead of the global parameterization, which helps to gain a better linearization and a faster computation process. IRLS algorithm introduces the reweighted potential function to avoid the outliers problem. DINM with local parameterization and IRLS also yield a better optimization trajectory towards to the ground truth.
All our algorithms acquire a great advancement in accuracy, efficiency, and robustness, related to the state-of-the art solutions in bundle adjustment.

Appendix A

Appendix

A.1 Matlab Code

All Matlab codes are stored in the attached CD-ROM.

A.2 Results Data

All experiment data and corresponding results, including data and plots, are also stored in the attached CD-ROM.

List of Figures

1.1 Bundle adjustment 6

2.1 3D coordinates transformation 12
2.2 Perspective projection 14

3.1 The reminder, $\{\|\boldsymbol{O}(\tau^2)\|\}$ 32
3.2 Jacobian matrix 34
3.3 Hessian matrix 35
3.4 Objective function comparison between different SC solving by CD . 40
3.5 Gradient norm comparison between different SC solving by CD . . . 40
3.6 Objective function comparison between different baselines 42
3.7 Gradient norm comparison between different baselines 43
3.8 Objective function comparison between different INMs 53
3.9 Gradient norm comparison between different INMs 54
3.10 Objective function comparison between different TNMs 56
3.11 Gradient norm comparison between different TNMs 57
3.12 Objective function comparison between INMs and TNMs 58
3.13 Gradient norm comparison between INMs and TNMs 59
3.14 Objective function comparison between different DINMs 60
3.15 Gradient norm comparison between different DINMs 61
3.16 Objective function comparison between DINM and others 62
3.17 Gradient norm comparison between DINM and others 63
3.18 Objective function comparison between different preconditioners in DINM 65
3.19 Gradient norm comparison between different preconditioners in DINM 66
3.20 Objective function comparison between different iteration steps in PCG loop 67
3.21 Gradient norm comparison between different iteration steps in PCG loop 68
3.22 Objective function comparison between different termination factors in PCG loop 69
3.23 Gradient norm comparison between different termination factors in PCG loop 70

4.1 Objective function comparison between DINMs and ADINMs 77
4.2 Gradient norm comparison between DINMs and ADINMs 78
4.3 Objective function comparison between local and global parameterization 81
4.4 Gradient norm comparison between local and global parameterization 82
4.5 Reprojection errors frequency histogram 83
4.6 Gradients frequency histogram 84

5.1 Structure of constructing synthetic dataset 90
5.2 Synthetic dataset 91
5.3 Time comparison between different commands in Matlab 93
5.4 Reprojection error on Ladybug dataset 95
5.5 Reprojection error on synthetic dataset 97
5.6 Objective function comparison among implementations with the Ladybug dataset 98
5.7 Gradient norm comparison among implementations with the Ladybug dataset 99
5.8 Schematic diagram of the small-scale synthetic dataset 100
5.9 Objective function comparison among implementations with the small-scale synthetic dataset 100
5.10 Gradient norm comparison among implementations with the small-scale synthetic dataset 101
5.11 Schematic diagram of the first medium-scale synthetic dataset 102
5.12 Objective function comparison among implementations with the first medium-scale synthetic dataset 103
5.13 Gradient norm comparison among implementations with the first medium-scale synthetic dataset 103
5.14 Schematic diagram of the larger medium-scale synthetic dataset . . . 105
5.15 Objective function comparison among implementations with the larger medium-scale synthetic dataset 105
5.16 Gradient norm comparison among implementations with the larger medium-scale synthetic dataset 106
5.17 Schematic diagram of the large-scale synthetic dataset 107
5.18 Objective function comparison among implementations with the large-scale synthetic dataset 107
5.19 Gradient norm comparison among implementations with the large-scale synthetic dataset 108

Acronyms and Notations

ADINM adaptive damped inexact Newton's method

ADMM alternating direction method of multipliers

ATE absolute trajectory error

BA bundle adjustment

CD Cholesky decomposition

CG conjugate gradients

DINM damped inexact Newton's method

DSO direct sparse odometry

GNA Gauss Newton algorithm

HMM hidden Markov model

INM inexact Newton's method

IRLS iteratively reweighted least squares

LMA Levenberg-Marquardt algorithm

LSE linear system of equations

MLE maximum likelihood estimator

PCG preconditioned conjugate gradient

RMSE root mean square error

SC Schur complement

SfM structure from motion

SLAM simultaneous localization and mapping

Bibliography

[ASSS] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. `http://grail.cs.washington.edu/projects/bal/`. Accessed: 2018-03-01.

[ASSS10] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010.

[CDHR08] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22, 2008.

[Chi11] Gregory S Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*, volume 2. Springer Science & Business Media, 2011.

[EKC18] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2018.

[EUC16] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016.

[FCDS17] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

[GGS+07] Giorgio Grisetti, Slawomir Grzonka, Cyrill Stachniss, Patrick Pfaff, and Wolfram Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3472–3478. IEEE, 2007.

[git] Alignsimefficient. `https://github.com/tum-vision/mono_dataset_code/blob/master/MatlabEvaluationCode/AlignSimEfficient.m`. Accessed: 2018-03-18.

[GK65] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[Hau07a] Raphael Hauser. Line search methods for unconstrained optimisation. University Lecture, 2007.

[Hau07b] Raphael Hauser. Trust region methods for unconstrained optimisation. University Lecture, 2007.

[Hay68] Emilie V Haynsworth. On the schur complement. Technical report, BASEL UNIV (SWITZERLAND) MATHEMATICS INST, 1968.

[HB14] Peter Hansen and Brett Browning. Visual place recognition using hmm sequence matching. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4549–4555. IEEE, 2014.

[HD07] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *IEEE Transactions on robotics*, 23(5):1036–1049, 2007.

[HW77] Paul W Holland and Roy E Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics-theory and Methods*, 6(9):813–827, 1977.

[Jon] Jonas. In matlab, when is it optimal to use bsxfun? `https://stackoverflow.com/questions/12951453/in-matlab-when-is-it-optimal-to-use-bsxfun`. Accessed: 2018-03-10.

[KA08] Kurt Konolige and Motilal Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, 2008.

[Lou05] Manolis IA Lourakis. A brief description of the levenberg-marquardt algorithm implemented by levmar. 2005.

[mata] arrayfun. `https://www.mathworks.com/help/matlab/ref/arrayfun.html`. Accessed: 2018-03-10.

[matb] bsxfun. `https://www.mathworks.com/help/matlab/ref/bsxfun.html`. Accessed: 2018-03-10.

[matc] cellfun. `https://www.mathworks.com/help/matlab/ref/cellfun.html`. Accessed: 2018-03-10.

[Mor78] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[NW06] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2006.

[per] The perspective and orthographic projection matrix. `https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/building-basic-perspective-projection-matrix`. Accessed: 2018-01-02.

[Rix17] Daniel Rixen. Multi-body simulation. University Lecture, 2017.

[RLA+17] Karthikeyan Natesan Ramamurthy, Chung-Ching Lin, Aleksandr Aravkin, Sharath Pankanti, and Raphael Viguier. Distributed bundle adjustment. *arXiv preprint arXiv:1708.07954*, 2017.

[Ryc14] Chris Rycroft. Advanced scientific computing: Numerical methods. University Lecture, 2014.

[SEG+05] Robert Sim, Pantelis Elinas, Matt Griffin, James J Little, et al. Vision-based slam using the rao-blackwellised particle filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, volume 14, pages 9–16, 2005.

[Sha98] Craig M Shakarji. Least-squares fitting algorithms of the nist algorithm testing system. *Journal of research of the National Institute of Standards and Technology*, 103(6):633, 1998.

[Sol17] Joan Sola. Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*, 2017.

[The] Structure from motion (sfm). `http://www.theia-sfm.org/sfm.html`. Accessed: 2018-03-11.

[TMHF99] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment - a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[WACS11] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.

[wika] Bundle adjustment. `https://en.wikipedia.org/wiki/Bundle_adjustment`. Accessed: 2018-01-04.

[wikb] Cholesky decomposition. `https://en.wikipedia.org/wiki/Cholesky_decomposition`. Accessed: 2018-03-01.

[wikc] Distortion(optics). `https://en.wikipedia.org/wiki/Distortion_(optics)`. Accessed: 2018-01-03.

[wikd] Iteratively reweighted least squares. https://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares. Accessed: 2018-03-06.

[wike] Levenberg-marquardt algorithm. https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm. Accessed: 2018-01-10.

[wikf] Moore-penrose inverse. https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse. Accessed: 2018-03-10.

[wikg] Right-hand rule. https://en.wikipedia.org/wiki/Right-hand_rule. Accessed: 2018-01-02.

[wikh] Structure from motion. https://en.wikipedia.org/wiki/Structure_from_motion. Accessed: 2018-03-11.

[Zha06] Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

License

Todo list

In your final hardback copy, replace this page with the signed exercise sheet. 3
Before modifying this document, READ THE INSTRUCTIONS AND GUIDELINES! 3