Master Thesis

An Error Aware RGB-D Visual Odometry

Uğur Bolat

Date:

Supervisors:
Dr.-Ing. Sven Lange
M.Sc. Tim Pfeifer

Faculty of Electrical Engineering and Information Technology
Professorship of Process Automation

Abstract

Robustness of a robot can be defined as an ability to operate under *uncertainty* without the occurrence of a downtime. There are well-designed robots whose task is to solve a defined problem without moving from its assembled position. They can operate safely as the uncertainty of operational components, and its controlled environment are modeled with substantial accuracy and precision. However, once we build robots that move around and interact with the real world, number of unforeseen events increase drastically. In such scenarios, robustness is crucial. The way we increase robustness is to have intelligent agents and accurate uncertainty models of their sensors and estimations. That being said, this work focuses on the latter and aims to investigate the uncertainty of RGB-D camera sensor in the context of Visual Odometry.

So far, researchers and engineers have developed many RGB-D camera based VO applications. In filter-based or graph-based SLAM applications, they are usually combined with other dead-reckoning and landmark measurements because of the drift occurring in relative pose estimations over time. To do so, one should have a reliable uncertainty model of the sensors being measured so that uncertainty of pose estimation can be estimated in the form of a covariance matrix. To my knowledge, there is no open source VO software that provides such covariance matrices for its pose estimations. Thus, the covariance matrix from VO is taken as an identity matrix. This does not offer any meaningful information as to whether the estimation should have specific importance comparing to other sensor measurements during filtering or optimization process. On the other hand, researchers model the uncertainty of RGB-D cameras such as Kinect, but they applied their models on applications that are outside of VO. The primary goal of this work is to build such a VO system that it provides not only relative pose estimations but also a covariance matrix of its estimated poses. To achieve this goal, we estimate the covariance matrix of the predicted pose by propagating metric uncertainty of 3D point features that are modeled with the sensor characteristics of an RGB-D camera.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BRIEF Binary Robust Independent Elementary Features.

DLT Direct Linear Transformation.

FAST Feature From Accelerated Segment Test.

IMU Inertial Measurement Unit.

IR Infrared.

ORB Oriented FAST and Rotated BRIEF.

RANSAC Random Sample Consensus.

RGB-D RGB Color Camera with Depth Sensor.

SIFT Scale-Invarient Feature Transform.

SLAM Simultenous Localization and Mapping.

SURF Speed-up Robust Features.

VO Visual Odometry.

List of Notations

$\mathcal{W}$ index for world coordinate system.

$\mathcal{C}$ index for camera coordinate system.

$\mathcal{D}$ index for disparity image space.

$\mathbf{x}$ unknown state vector.

$\mathbf{x}^*$ desired state vector.

$\mathbf{p}$ position vector.

$\mathbf{q}$ quaternion vector.

$\mathbf{t}$ translation vector.

$\mathbf{u}$ 2D image feature's pixel coordinates.

$\mathbf{X}$ 3D point features's position.

$\mathbf{K}$ instrinsic matrix.

$\mathbf{T}_{A,B}$ homogeneous transformation representing frame A with respect to frame B.

$\mathbf{P}$ projection matrix.

$\mathbf{F_p}$ projection function.

$\mathbf{F_d}$ distortion function.

$\mathbf{F_{dp}}$ projection function combined with distortion effect.

F_d function that converts diparity value to metric depth.

$\mathbf{H}$ homography matrix.

Acknowledgement

Chapter 1

Introduction

For a mobile robot to act autonomously in the real world, the state of the robot must be known accurately. We define this physical state by its *pose*, meaning its position and orientation. The current pose of the robot can be measured by two ways; i.e., *dead-reckoning* that measures *relative* motion and *landmarks* that measures *absolute* position which are known a priori. Without priori-known landmark measurements, dead-reckoning systems are destined to drift from its real position. Thus, the robot must build a map of its environment along with the previously seen landmarks. Then, it will have a chance to recover from drifts by recognizing the same landmarks in the same area in different time. In fact, this problem in robotics is named as Simultenous Localization and Mapping (SLAM). SLAM problem has almost 30 years history Moravec 1980 and vision systems always had great importance. Even though some researchers Frese 2010 consider SLAM to be solved, there are still open research questions regarding robustness, accuracy and real-time operation. To solve SLAM problems, robots are equipped with various kind of both dead-reckoning systems; e.g., IMU, wheel odometry and visual odometry and landmark measurements; e.g., GPS and radio-based positioning systems.

As being part of dead-reckoning systems, VO provides the *ego-motion* estimation of a robot by exploiting only one or multiple cameras. The working principle of VO relies on the idea that if we had two subsequent images captured by the camera in 3D space would tell us about a change in the camera's pose from one image to another as changing patterns of objects' shapes or pixels' intensity occurs during motion. This change in the pose refers as *relative pose*. With this in mind, for the VO system to work accurately, the environment should be adequately illuminated, static and texture-rich. However, in the real world, we deal with shadowed, dynamic and texture-poor environments. Eventually, drifts will occur in VO. Hence, the good design of a mobile robot should contain sufficient amount sensors along with their known biases and calibration parameters.

As opposed to camera, most dead-reckoning sensors are mostly built around one specific purpose and output information that do not need further manipulations (except post-filtering). Whereas, the camera offers rich data by mapping 3D space onto a 2D plane and the output data are quantized pixels according to the intensity of the illumination. This type of data makes camera sensor a multi-purpose device and one can build various kinds of computer vision appli-

cations. Considering the potential, computer vision researchers proposed VO methods that are tailored to the different type of environments. When designing a VO pipeline, one should select a suitable algorithm and camera type based on the operating environment. As to camera types, RGB-D cameras based on structured light (e.g., PrimeSense Carmine, Microsoft Kinect, and Asus Xtion) that are categorized as active stereo cameras are especially intriguing. These cameras gave rise to many 3D vision applications including VO. Even though it has certain limitations; i.e., the depth accuracy being grown with distance to the object or the properties of the projected objects' material, it offers a cheap 3D data, especially for indoor applications. As to algorithms, there are now many open source VO software; e.g., LIBVISO2 Geiger, Ziegler, and Stiller 2011 and FOVIS Huang et al. 2011, that provide relative pose estimation of a stereo camera. The drawback of those early VO systems is that they do not give any information about how uncertain the relative predicted pose is, namely *covariance matrix*. That is why VO systems used in SLAM are either tightly-coupled with other sensors like IMU or given less importance when combining with other sensors. This introduces a disadvantage regarding the robustness of the system since we don't have information about the uncertainty of pose estimations. The objective of this thesis is to build a VO system that can provide metric uncertainty information after estimating a camera pose.

The overall structure of the thesis takes the form of six chapters, including this introductory chapter. In Chapter 2, we study the essential geometrical models and calibration techniques for an RGB-D camera. Then, Chapter 3 outlines the standard feature-based VO pipeline and explains the necessary image processing methods for building such a pipeline. In Chapter 4, we discuss how to model the uncertainty of an RGB-D camera and integrate this model into the VO pipeline so that the uncertainty of a pose estimation can be calculated. Afterward, we evaluate the accuracy and consistency of the proposed VO system with both simulated data and TUM RGB-D dataset in Chapter 5. Finally, the conclusion gives a summary and critique of the findings.

Chapter 2

Camera Models

In this chapter, we will discuss two geometrical model and calibration methods for RGB-D camera. In principle, a camera maps from a 3D world scene to a 2D image plane. We call this process *projection* operation. Since the VO systems process camera image sequences, one has to model this projection operation accurately. One of the basic camera modeling technique is the *Pinhole Model* where the projection of the 3D points are mapped on a 2D image plane.

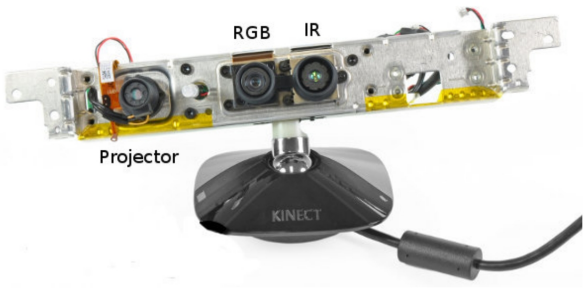


Figure 2.0.1: Microsoft Kinect V1 is a RGB-D camera that has 640 $\times$ 480 pixels spatial resolution, 0.8-4.0 m depth range, 2-40 mm depth resolution and 30 fps Smisek, Jancosek, and Pajdla 2011.

While mapping a point from 3D space to 2D space, we lose the depth information. Although there are ways to recover relative depth scale by taking images from different poses with a monocular camera, they can't provide metric depth. In this thesis, we are interested in cameras that offer metric depth using stereo cameras, more specifically active stereo camera. One can model these active stereo cameras with *Triangulation Model*. That being said, these two models cannot be used without having the camera to be *calibrated* since many anomalies occur in manufacturing.

2.1 The Pinhole Model

The light rays are captured through the camera's lens onto an electronic plate (called *image plane* in computer vision) that convert light intensity to electrical signals. The pinhole model is, on the other hand, an approximation which simplifies this operation. In this model, the camera centre sits behind the image plane. The Z-axis, so-called *principal axis*, of this coordinate system points out through the origin of the image plane, and the point where pierce through image plane is called the *principal point* **p**. We can also see how other two axes; i.e., X and Y, are located in figure 2.1.1 and this is known as the *camera coordinate system*.

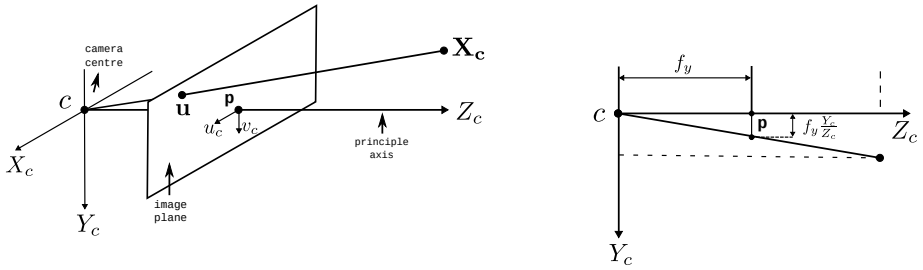


Figure 2.1.1: In the Pinhole Model, C is the camera centre and $\mathbf{p}$ is the principle point. A 3D point $\mathbf{X_c}$ in the camera coordinate system is projected as a 2D point $\mathbf{u}$ onto the image plane. Note that the camere coordinate system is drawn according to right-hand rule. The illustration is redrawn from Richard Hartley 2003 accordingly.

Thanks to geometrical propotion property, we can project the 3D point $\mathbf{X_c} = (X_c, Y_c, Z_c)^T$ in the camera coordinate system to the 2D point $\mathbf{u} = (f_x X_c/Z_c, f_y Y_c/Z_c)^T$ on the image plane, where f_x and f_y are the *focal lengths* between the camera centre and the pricipal point with respect to horizontal and vertical axis of the camera coordinate system respectively. After projection, we obtain a 2D point as *pixel coordinates* $\mathbf{u} = (u, v)^T$ on the image plane. To be more specific, we can write the projection operation as a linear mapping function in the following way if we utilize the homogeneous coordinates:

$$\begin{pmatrix} U \\ V \\ 1 \end{pmatrix} \sim Z_c \begin{pmatrix} f_x \frac{X_c}{Z_c} \\ f_y \frac{Y_c}{Z_c} \\ 1 \end{pmatrix} = \begin{pmatrix} f_x X_c \\ f_y Y_c \\ Z_c \end{pmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.1}$$

This equation applies for the case when 3D points are projected onto a plane where the principal point is the origin. However, the common convention in practice is to have the origin at the corner Richard Hartley 2003, instead of the centre.

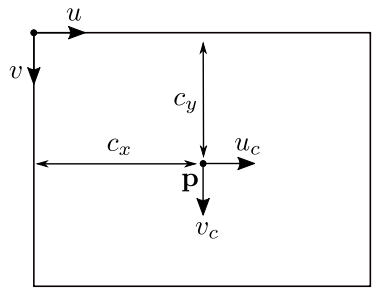


Figure 2.1.2: We take OpenCV's pixel coordinate convention where the origin is located at the upper left corner. In order to shift the origin from camera centre to the corner, c_x, c_y principle point offsets are added.

Thus, we get offsets, which can further be added as:

$$\begin{pmatrix} U \\ V \\ 1 \end{pmatrix} \sim Z_c \begin{pmatrix} \frac{f_x X_c + Z_c c_x}{Z_c} \\ \frac{f_y Y_c + Z_c c_y}{Z_c} \\ 1 \end{pmatrix} = \begin{pmatrix} f_x X_c + Z_c c_x \\ f_y Y_c + Z_c c_y \\ Z_c \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.2}$$

where c_x and c_y are coordinates of the principal point $\mathbf{p}$. In addition to principal offsets, an inaccurately synchronized pixel-sampling process can result in *skewed pixels*. This camera imperfection leads to non-square pixels as seen in figure 2.1.3.

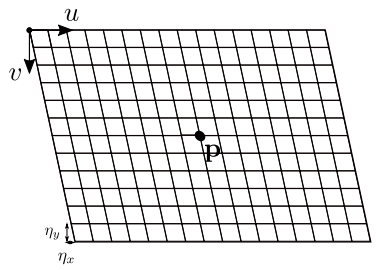


Figure 2.1.3: Skewed pixels occur in earlier versions of CCD cameras because the camera's detector array is not orthogonal to the principle axis. This design issue is mostly fixed in modern camera and thus it is usually neglected by taking $\eta_x = 1$, $\eta_y = 1$ and $s = 0$.

We can scale the square pixels, having 1:1 pixel aspect ratio, with the corresponding skew parameters η_x, η_y and s:

$$\begin{pmatrix} U \\ V \\ 1 \end{pmatrix} \sim \begin{bmatrix} f_x \eta_x & s & c_x & 0 \\ 0 & f_y \eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{bmatrix} \alpha_x & s & c_x & 0 \\ 0 & \alpha_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.3}$$

At this point, we can extract the following matrix:

$$\mathbf{K_{RGB}} = \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

where $\mathbf{K_{RGB}}$ is called *intrinsic matrix*, which represents the characteristics of a camera sensor. With this matrix, one can further reformulate the notation 2.3 in more compact form:

$$\mathbf{u} = \mathbf{K_{RGB}}[\mathbf{I}|\mathbf{0}]\mathbf{X_c} \tag{2.5}$$

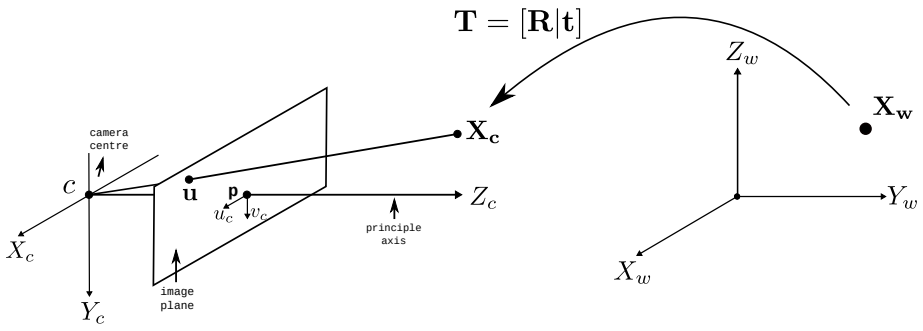


Figure 2.1.4: The Z-axis of the camera coordinate system aligns with the principal axis that is *local* to camera frame. In fact, we have measured 3D points that we know their position in the *world coordinate system* which is also refer to the *global frame*, as opposed to the camera coordinate system referring to *local frame*.

Remember that we measure 3D points in the real-world with respect to the camera centre. Thus, these two coordinates systems, i.e., the camera and world coordinate system, can be transformed one another by a rotation and a translation as it is depicted in figure 2.1.4 and we are interested in converting from the world coordinate system to the camera coordinate system in the context of projection operation. To do so, we perform series of rotations around each axis of the Cartesian coordinate system in $\mathbb{R}^3$ Euclidean space by using *rotation matrices* where $R_x, R_y, R_z \in SO(3)$ is the rotation group:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \tag{2.6}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \tag{2.7}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

Let's concatenate all three rotations about axes z, y, x respectively (also called *yaw, pitch, yaw*) by the matrix multiplication:

$$\mathbf{R} = \mathbf{R_z}(\gamma)\mathbf{R_y}(\beta)\mathbf{R_x}(\alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.9}$$

Then, we add a translation $\mathbf{t} \in \mathbb{R}^{3x1}$:

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.10}$$

For convenience, we stack the rotation matrix and the translation vector into one:

$$\mathbf{T_{RGB}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{2.11}$$

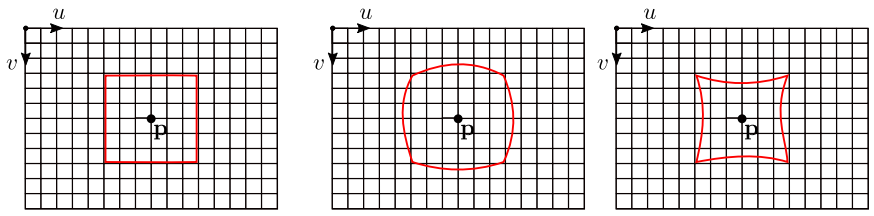
The $\mathbf{T_{RGB}} \in \mathbb{R}^{4x3}$ matrix in fact represents a *transformation* from 3D position in the world coordinate system to the camera coordinate system, which we call *extrinsic camera parameters*. Finally, we combine intrinsic $\mathbf{K_{RGB}}$ and extrinsic $\mathbf{T_{RGB}}$ matrices in the following form:

$$\mathbf{u} = \mathbf{P_{RGB}}\mathbf{X_w} = \mathbf{K_{RGB}}\mathbf{T_{RGB}}\mathbf{X_w} = \mathbf{K_{RGB}}[\mathbf{R}|\mathbf{t}]\mathbf{X_w} = \mathbf{K_{RGB}}\mathbf{X_c} \tag{2.12}$$

where $\mathbf{P_{RGB}}$ is the *projection matrix*. All of these series of linear operations can be written as a function too:

$$\mathbf{u} = \mathbf{F_p}(\mathbf{X_w}) \tag{2.13}$$

where $\mathbf{F_p}(\mathbf{X_w})$ is the *projection function*, which takes the 3D points in the world coordinate system, transforms to the camera coordinate systems and then maps them onto the image plane.



(a) Undistorted Image (b) Barrel Distorted Image (c) Pincushion Distorted Image

Figure 2.1.5: All emitted lights through the camera lens is expected to intersect at the same focus point and the radial distortion is caused by having a defected lens that causes the emitted lights focusing different points. This makes the straight lines appear to be curved.

One last issue with regards to imperfect pixels is the *radial distortion*. The distortion effect has non-linear characteristics. Thus, we introduce polynomial function whose coefficients $\kappa = (k_1, k_2, p_1, p_2, k_3)^T$ can be fitted by optimization. The polynomial function is given as follows:

$$\begin{aligned} x'' &= \mathbf{F_d}(x') = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y'' &= \mathbf{F_d}(y') = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{aligned} \tag{2.14}$$

where $x' = X_c/Z_c$ and $y' = Y_c/Z_c$. Now, we can update the projection function:

$$\begin{pmatrix} U \\ V \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x x'' + s y'' + c_x \\ \alpha_y y'' + c_y \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{F_d}\left(\begin{pmatrix} X_c/Z_c \\ Y_c/Z_c \\ 1 \end{pmatrix}\right) = \mathbf{K}\mathbf{F_d}\left([\mathbf{R}|\mathbf{t}]\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}\right) \tag{2.15}$$

Let's combine projection and distortion function into one:

$$\mathbf{u} = \mathbf{F_{dp}}(\mathbf{X_w}) \tag{2.16}$$

To build any reliable computer vision application with digital cameras, it is essential to find the parameters of the projection matrix and the radial distortion. The next section describes one of many numerical methods for estimating them in literature.

2.2 The Triangulation Model

Projected images captured by RGB cameras lack depth and angle information. To acquire this information, two main techniques are developed; e.g., passive stereo cameras and active stereo cameras. For passive stereo cameras, typically two synchronized cameras are placed horizontally with a known distance to each other. Whereas, for an active stereo camera, one typically has one light projector and one camera sensor. For example, in Kinect, an Infrared (IR) laser projects structured IR speckle light pattern on an object, and then the deformed light due to 3D geometry of the object is captured with a monochrome IR camera from a different position.

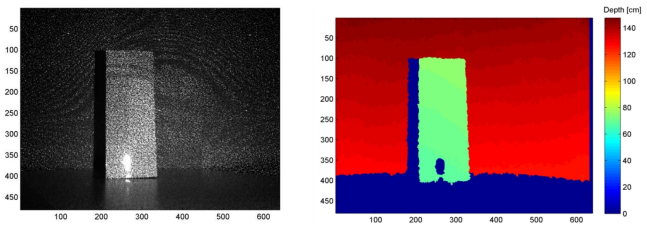


Figure 2.2.1: On the left, the IR camera capture the patterns of speckles projected on an object. On the right, we see the resulting depth image if disparity data is converted accordingly Khoshelham and Elberink 2012.

Since we use Kinect V1 to retrieve depth information for our experiments, we will be modeling active stereo vision principle even though the basic principle behind them is the same mathematical model, which is the *triangulation model*. As shown in figure 2.2.2, this model is a geometrical model that takes advantages of similarity triangles to calculate the depth.

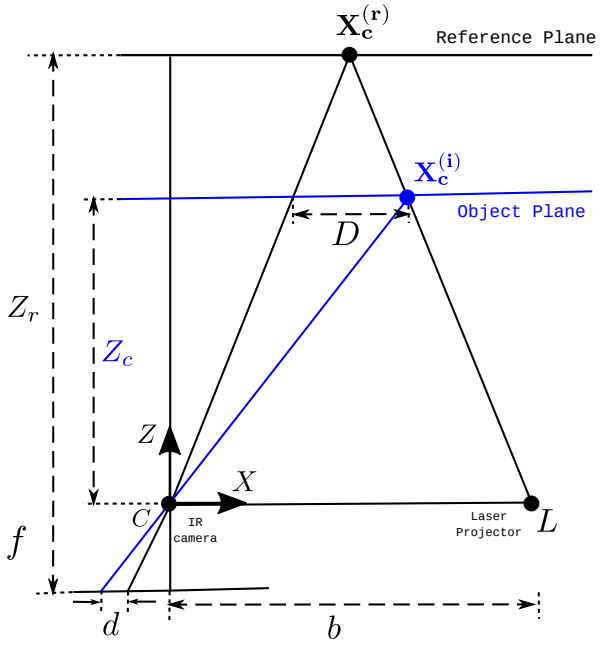


Figure 2.2.2: The Depth Measurement Model

In this setup, we again utilize the camera coordinate system similar to RGB camera and IR camera with f focal length is placed by facing perpendicular to the (principle axis) Z-axis at the origin. Then, IR laser projector is also placed along the X-axis parallel to the IR camera with *baseline* b distance. Additionally, we measure d as a *disparity* data and the maximum range that can be measured refers to Z_r. Ultimately, we are interested in finding Z_c distance if depth information of point $\mathbf{X_c}^{(i)}$ is desired. For doing so, we build two useful relationships using similarity of triangles:

$$\frac{D}{b} = \frac{Z_r - Z_c}{Z_r} \text{ and } \frac{d}{f} = \frac{D}{Z_c} \tag{2.17}$$

If the depth camera parameters such as f, b, and Z_r is calibrated and we get d disparity data, we can easily extract Z_c depth information with the following formula:

$$Z_c = \frac{Z_r}{1 + \frac{Z_r}{fb} d} \tag{2.18}$$

Another critical point to note is that Kinect or other depth cameras might not provide the depth metric information directly in practice. For instance,

Kinect provides us disparity image data that correspond to inverse depth quantized with 11 bits and the relationship between disparity data and real depth is non-linear as shown in figure 2.2.3.

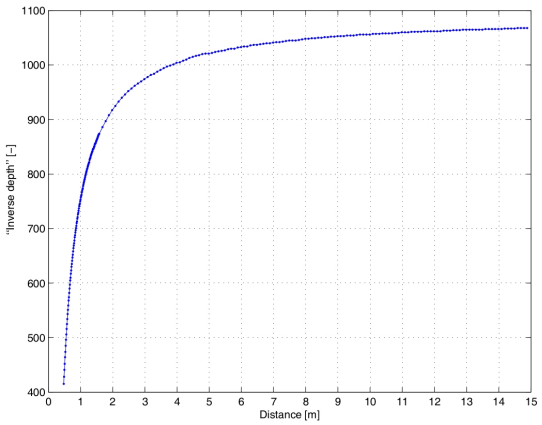


Figure 2.2.3: Non-linear Relationship Between Inverse Depth and Disparity Smisek, Jancosek, and Pajdla 2011

Thus, we need to update depth equation 2.18 by taking its inverse and introducing normalization factor replacing with d with $md' + n$:

$$Z_c^{-1} = (\frac{m}{fb})d' + (Z_r^{-1} + \frac{n}{fb}) \tag{2.19}$$

To make it convenient to calculate, we can take its inverse again:

$$Z_c = \frac{1}{(\frac{m}{fb})d' + (Z_r^{-1} + \frac{n}{fb})} \tag{2.20}$$

The relationship between disparity data and metric depth measurement can be written as a function in the following form:

$$Z_c = F_d(d') \tag{2.21}$$

Now that we know how to get metric depth information Z_c from disparity data by utilizing the triangulation model, we can briefly write down similar projection operation for the IR camera.

$$\mathbf{u} = \mathbf{K_{IR}}\mathbf{F_d}(\mathbf{T_{IR}}\mathbf{X_w}) \tag{2.22}$$

Keep in mind that we first need to calibrate the IR camera's intrinsic and extrinsic paratemeters to find related depth related parameters, i.e., f, b, Z_r, m, n. In the following section, we will discuss how calibration processes are performed for RGB-D camera to measure quality data.

2.3 RGB-D Camera Calibration

The calibration process is a crucial part of any computer vision applications, and there are many sophisticated techniques to achieve accurately. However,

it is important to note that full derivations of the calibration formulation are not provided in this thesis, but only the important points are given. Therefore, I refer readers to Z. Zhang 2000, Smisek, Jancosek, and Pajdla 2011, Karan 2015 and C., Kannala, and Heikkila 2016 for the details of a RGB-D camera calibration. Since we are about to perform 3 calibration operation: RGB camera, IR camera, and depth measurement, we assume that both RGB and IR camera's image planes are aligned for simplicity (in practice, one must perform transformation $\mathbf{T}_{RGB,IR}$ between RGB and IR camera if a feature from RGB camera used in IR camera) and they have 1:1 pixel correspondences. Under these assumptions, let's start with RGB and IR camera.

RGB and IR Camera Calibration

We calibrate RGB and IR camera since they both project 3D space to 2D space, but only measuring the different light spectrum. Thus, the calibration process for the RGB camera that we will discuss applies for the IR Camera as well. To begin with, the simplest case which can help us to understand the calibration process is that we assume that we know the exact position of 3D points in world coordinate system and exact position of 2D points on the image plane. One can build a constraint between them by exploiting the projection function:

$$\mathbf{u_{RGB}}^{(i)} = \begin{pmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{pmatrix} = \mathbf{P_{RGB}}\mathbf{X_w} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix} \begin{pmatrix} X_w^{(i)} \\ Y_w^{(i)} \\ Z_w^{(i)} \end{pmatrix} \tag{2.23}$$

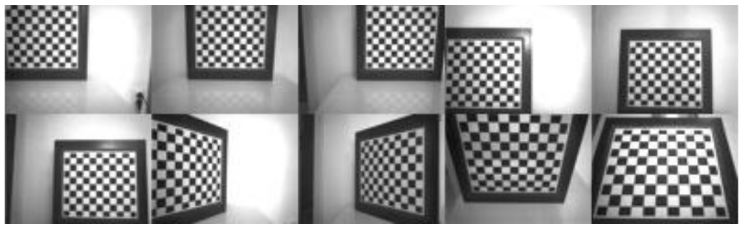
Let's now distribute the projection matrix onto the 3D point measurement to retrieve individual pixel coordinates:

$$u^{(i)} = \frac{p_{00}X_w^{(i)} + p_{01}Y_w^{(i)} + p_{02}Z_w^{(i)} + p_{03}}{p_{20}X_w^{(i)} + p_{21}Y_w^{(i)} + p_{22}Z_w^{(i)} + p_{23}} \tag{2.24}$$

$$v^{(i)} = \frac{p_{10}X_w^{(i)} + p_{11}Y_w^{(i)} + p_{12}Z_w^{(i)} + p_{13}}{p_{20}X_w^{(i)} + p_{21}Y_w^{(i)} + p_{22}Z_w^{(i)} + p_{23}} \tag{2.25}$$

Since $\mathbf{u}$ and $\mathbf{X_w}$ are known, we can find elements of the $\mathbf{p} = (p_{00}, p_{01}, \ldots, p_{23})^T$ matrix by solving $\mathbf{Ap} = \mathbf{0}$ liner system of equations from 2.24 and 2.25. For *minimal solution* of this linear system of equations, we need at least $n \geq 6$ measurement points to solve the problem as the $\mathbf{P_{RGB}}$ matrix has 12 unknowns (11 if scale is ignored). Note that this accounts for having noise-free measurement which does not hold in reality. Then, the problem becomes *over-determined*.

(a) RGB Images



(b) IR Images

Figure 2.3.1: When calibrating RGB and IR camera with a checkerboard, one should produce many measurement points by placing checker board at different orientation and different tilted posture Karan 2015.

In noisy measurement case, the problem is usually solved with *singular value decomposition (SVD)* with $n > 6$ measurement points. This method is called the *Direct Linear Transformation (DLT)*. Disadvantage of the DLT methods, it is still sensitive errors since it only considers algebraic errors (that are the residuals of $\mathbf{Ap}$). Another drawback of DLT is that it cannot compensate non-linearities of projection function due to the radial distortions. Thus, instead of DLT, the non-linear least squares optimization is usually performed for the better accuracy:

In practice, the checkerboard is used to get many good measurement points as we can easily extract edge features from the image as 2D points. Plus, we also know the corresponding 3D positions in the world coordinate system. Also, we have prior knowledge about some of the parameters of intrinsic matrix, e.g., pixels are squared, skew factor is trivial, optical center near the center of the image. All of these can increase our chance for successful convergence when optimizing:

$$\underset{\mathbf{F_{dp}} \to \mathbf{K_{RGB}}, \kappa_{RGB}, \mathbf{R_{RGB}}^{(i)}, \mathbf{t_{RGB}}^{(i)}}{\operatorname{argmin}} \sum_{i=1:n} ||\mathbf{u_{RGB}}^{(i)} - \mathbf{F_{dp}}(\mathbf{X_w}^{(i)})||^2 \tag{2.26}$$

where $\mathbf{F_{dp}}$ is the projection function along with distortion (see notation 2.16), $\mathbf{u}_{(i)}$ is the measured pixel coordinates of a feature on image plane and $\mathbf{X_{w}}_{(i)}i$ is the 3D coordinates of a feature in world coordinate system to identify the following parameters:

- $\mathbf{K_{RGB}}$ is the intrinsic matrix for RGB camera,
- κ_{RGB} is the distortion coefficients for RGB camera,

- $\mathbf{R_{RGB}}^{(i)}$ is the corresponding orientation for RGB camera and
- $\mathbf{t_{RGB}}^{(i)}$ is the corresponding translation for RGB camera.

As mentioned earlier, we can apply the same optimization process for IR camera to find $\mathbf{K_{IR}}$ intrinsic matrix for IR camera, κ_{IR} distortion coefficients for IR camera, $\mathbf{R_{IR}}^{(i)}$ corresponding orientation for IR camera and $\mathbf{t_{IR}}^{(i)}$ corresponding translation for IR camera.

Depth Measurement Calibration

As we discussed in the previous section, disparity data for every pixel on IR image corresponds to the metric depth information. We can only build the relationship between pixel coordinates of the IR image and disparity value when the IR camera is calibrated and its $\mathbf{u_{IR}}$ pixel measurements are corrected accordingly.

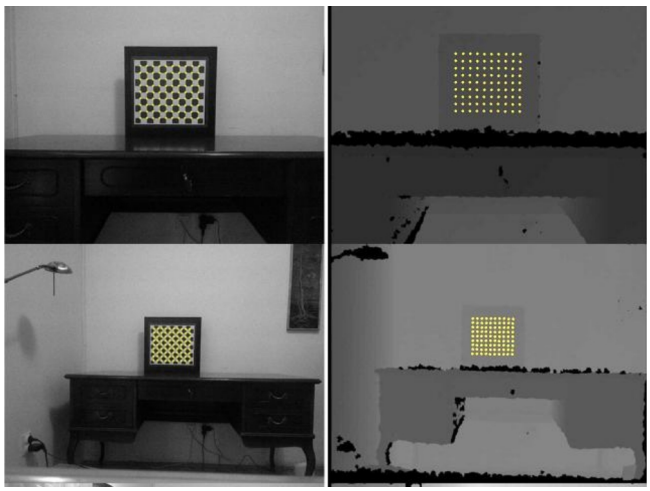


Figure 2.3.2: When calibrating the depth parameters, checkerboard should be placed at the different distances to the camera so that we get enough measurement points, especially for fitting m, n parameters Karan 2015.

To get the disparity for i^{th} feature point, a simple mask function can be used:

$$d'^{(i)} = M(\mathbf{u_{IR}}^{(i)}) \tag{2.27}$$

where M is a function that return the disparity value at pixel coordinates $\mathbf{u_{IR}}^{(i)} = [u_{IR}^{(i)}, u_{IR}^{(i)}]^T$ of calibrated IR image. Then, our one last step will be to convert disparity measurement to depth for every feature on the checkboard and minimize the error between measured depth $\hat{Z}_c = F_d(d')$ (see notation 2.22) and real depth Z_c:

$$\underset{F_d \to f,b,Z_r,m,n}{\operatorname{argmin}} \sum_{i=1:n} ||Z_c^{(i)} - F_d(d'^{(i)})||^2 \tag{2.28}$$

The calibration processes for cameras are a well-studied problem in computer vision literature. Fortunately, there are many open software libraries, such as OpenCV, that offers such implementations.

Chapter 3

Fundamentals of Visual Odometry

3.1 Related Work

First research in estimating camera ego-motion was done in a Ph.D. Thesis Moravec 1980 in the 1980s. He used a sliding camera as a stereo fashion that captures images when the robot stopped after moving for some small distance. Another early study, from Matthies and Shafer Matthies and Shafer 1987, formulated the ego-motion estimation for stereo vision. The interest in VO peaked after NASA's Mars expeditions with a rover in 2003. Researchers and engineers in NASA JPL further improved the robustness of their mobile robots with the VO system Olson et al. 2003.

In computer vision, Structure from Motion (SFM), that tackles 3D reconstruction of an environment with moving camera, is a well-studied topic. For SFM, it is critical to estimate camera pose accurately to build the 3D environment. VO can be thought of subset problem of SFM. The reason VO parted from SFM is that VO systems started to empower other applications such SLAM and are required to operate in real-time, all of which introduces a great challenge.

For the first time, Visual Odometry term was introduced in Nist and Bergen 2004. It estimates the transformation matrix by solving the P3P problem between consecutive frames. Nister also demonstrated a VO pipeline that became a defacto system configuration even for today's VO applications. With the help of early robotics applications in NASA and computer vision community, the VO became a quite popular field. That being said, two of the most influential papers were Fraundorfer and Scaramuzza 2011 and Scaramuzza and Fraundorfer 2011. By publishing these tutorials, the authors gave researchers and engineers a design recipe for building the VO system for different kind of environment settings since there is no such a system that works under any conditions. One can also find a survey of VO types, approaches and challenges in Aqel et al. 2016.

Over the years, research on VO is progressively increased and various types of VO systems are proposed. Therefore, VO systems can be categorized based on different phenomena such as camera sensor types and solver algorithms choice. One can utilize monocular or stereo cameras to capture images. The stereo

cameras are further grouped into two categories; i.e., active and passive cameras. For the active camera, besides color or monochrome camera, there is another camera such IR camera that measures the depth information combination with IR laser projector. For the passive camera, the depth is calculated from two color or a monochrome camera. In the case of solver algorithms, one can group VO into two categories; i.e., feature-based and appearance-based.

The feature-based VO algorithms are interested in extracting distinct and repeatable feature points from images and finding correspondences in extracted features either between consecutive frames or keyframes. The challenging part in feature-based VO is to build a system that can match features across different frames without errors. However, this does not hold in reality, so one typically needs to remove these errors with outlier rejection algorithms such as RANSAC Fischler and Bolles 1981. Among many VO systems, there are two favorite open source feature-based VO tools which stand out; i.e., LIBVISO2 Geiger, Ziegler, and Stiller 2011 and FOVIS Huang et al. 2011. Briefly, LIBVISO2 uses both simple blob and corner filters to extract features. The extracted features are filtered by non-maximum suppression to increase the robustness of the matching process. Then, it calculates the depth information by triangulation technique as it uses a passive stereo camera. Finally, it minimizes the reprojection error. The reprojection error function is constructed by projection from features on the left camera onto the right camera and vice-versa, to estimate the pose. Note that RANSAC applied on feature matches to remove outliers. Whereas, FOVIS is more accurate and faster than LIBVISO2 according to Fang and Y. Zhang 2015. FOVIS uses only FAST corner detectors to extract features. For matching features, FOVIS take advantage of keyframes instead of consecutive images to reduce the drift. Instead of using RANSAC for outlier rejection, it constructs a graph of consistent feature matches and updates the features that obey the idea that the Euclidean distance between two features at one time should match their distance at another time. Finally, several refinement processes are applied during motion estimation to improve accuracy.

The most significant disadvantage of feature-based VO is that the accuracy of the pose estimations decreases if the operating environment lacks texture-rich scenes such as corridors or the measured images are blurry due to fast motions. Thus, appearance-based VO algorithms utilize the entire image instead of extracted features. Initially, Iterative Closest Point (ICP) Besl and McKay 1992 was used to minimize the geometrical error between 3D surfaces. Then, various kinds of ICP algorithms are built for improving efficiency Rusinkiewicz and Levoy 2001. Even though ICP is useful for creating 3D shapes with point clouds, it is slower and less accurate compared to feature-based methods. Then, another type of appearance-based approach, so-called Dense Visual Odometry (DVO) Kerl and Cremers 2013, was emerged by minimizing the photometrical error based on the pixel intensity between consecutive frames. Although the original techniques developed with appearance-based approaches produces less accurate results, we can say that it recently started to gain popularity with carefully designed hybrid methods, so-called Semi-Dense VO Zhou, Kneip, and Li 2017, and it can outperform feature-based VO in some cases.

In this chapter, we will focus on the standardized feature-based VO pipeline, which will serve us as a foundational basis so that we comprehend the algorithmic errors. Hence, We structured this chapter according to four main components of the pipeline: (1) feature extraction, (2) feature matching, (3) outlier

rejection and (4) pose estimation.

3.2 Feature Extraction

Image features are a collection of regions of interest or points of interest that describe the image. In this way, we compress the necessary information from images so that we can deal with computationally expensive image processing tasks. Points of interest, which also can be called as *keypoints*, *features* or *landmarks* interchangeably, are particularly valuable because their location in the image can be measured accurately. This is useful for localization-related tasks such as VO.

In feature-based VO, the critical task is to find good features. What defines good features from others is that they are distinct, repeatable, computationally cheap and invariant to geometrical changes. One has many options to produce such image , but two common methods in VO systems are blobs and corners. Blobs are image patterns that contain distinct image response comparing to their neighborhood pixels. Blobs take advantage of pixel intensity or color to decide whether it has a distinct response or not. In the VO literature, SIFT Lowe 2004 and SURF Bay et al. 2008 are popular choices for detecting blob features. In contrast, corners are the meeting points where two or many edges intersect. Corners take advantage of the geometrical structure of an image. FAST Rosten and Drummond 2006 and Harris Harris and Stephens 1988 are widely used for detecting corners.

Fundamentally, a two-step process is needed to extract good features. First, you take a response function called *image filter*, shift this filter through the image and save the one that has greater response than your previously defined threshold. This might be a Gaussian filter for blobs or a corner detector filter for corners. In the second step, you perform non-maxima suppression on the resulting image features to find local minima of the function. This step will help to remove similar image features and to choose the ones having maximum confidence so that distinctiveness of the features are ensured. Some VO might skip non-maxima suppression because of the efficiency reasons.

Inherently, each feature detector has certain limitations, and one has to choose which detector to use depending on task objectives. Therefore, we may ask the following question: Does the localization environment involve more texture-oriented objects like floors, walls, etc. or geometrical shapes like urban areas where many lines exist? However, the rule of thumb is that blobs are distinct but slow to compute and corners are fast to compute but less distinct.

After extracting features, one needs to encode the detected image features into a format that we can perform comparison or search operation among them. This is done by taking the neighboring pixels around the image features and convert into a more compact form. For example, SIFT, most well-known feature descriptor, creates a patch around an image feature, divides this patch into smaller grids, calculate the gradient of each grid and saves them as a histogram. This procedure makes feature descriptor robust against scale or rotation changes. Then, one can use these descriptors for various comparison operations such as matching or tracking in VO. However, we utilize the ORB that combines feature extraction and description process in our VO.s

ORB

One of the most strict requirements of VO is the real-time constraints since it is expected to work at similar to low-level inertial sensors, i.e., accelerometer, gyroscopes, etc. As previously discussed, blobs detectors are computationally expensive. Therefore, corner-based feature detectors are more prevalent in VO. *ORB* Rublee et al. 2011 combines FAST detector Rosten and Drummond 2006 and BRIEF descriptor M. et al. 2010 by tuning the necessary parameters to produce robust and reproducible features. In the end, it mostly performs as accurate as SIFT, plus faster. Here are steps on how to extract features and create descriptors with ORB:

1. **Detect corners with FAST**: FAST take each pixel on the image and compare with its adjacent pixels. More specifically, ORB uses FAST-9, which takes a patch of a discrete circular radius of $r = 9$.

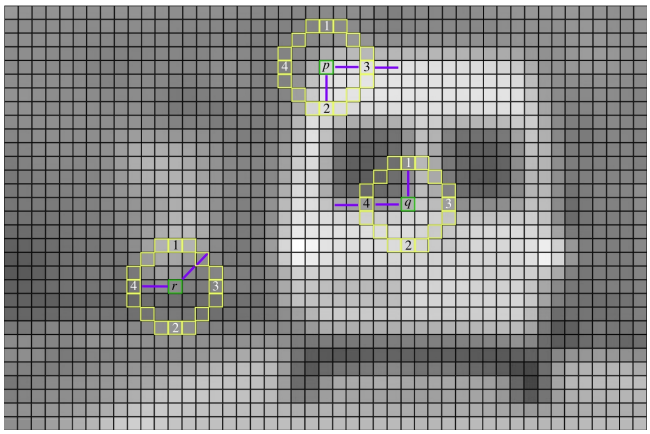


Figure 3.2.1: Point of interest or feature points are at the intersection of edges as seen for the p, q and r pixels and the blue lines correspond to the directions of the edges.For this figure, $r = 4$ is taken Klette 2014.

The basic principle behind FAST is if the selected pixel p is $\pm t$ darker or brighter than adjacent pixels $r \in 1, 2, \ldots, n$, we call it a corner, where t is our empiric threshold. Comparison pixel set r is chosen as a circle around in figure 3.2.1.

$$S_{p\to r} = \begin{cases} S_b, & I_{p\to r} \leq I_p - t \\ S_d, & I_p - t \leq I_{p\to r} \\ S_s, & otherwise \end{cases} \tag{3.1}$$

If a set of N contiguous pixels are either dark S_d or bright S_b, we call interest point p as a corner $\mathbf{u_c} = M_c(p)$, where M_c is a function that returns the pixel coordinates of the corner pixel and N is another empiric parameter whose purpose is to ensure majority of the comparison either dark or bright.For more details about efficient ways to calculate FAST corners, I refer readers to Rosten and Drummond 2006.

2. **Rank FAST corners with Harris**: After FAST detection, we might get many corner candidates around the interest point. However, FAST does not measure how good a corner is. Thus, we use Harris detector to rank corner candidates:

$$\mathbf{A} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{3.2}$$

The $\mathbf{A}$ matrix is calculated by the I_x and I_y partial derivatives with respect to x and y direction on the image plane and $w(x,y)$ weighting window.

$$R_c(\mathbf{A}) = det(\mathbf{A}) - k(trace(\mathbf{A}))^2 \tag{3.3}$$

where $det(\mathbf{A}) = \lambda_1 \lambda_2$, $trace(\mathbf{A}) = \lambda_1 + \lambda_2$ and k is an empiric tuning parameter. Then, we use the resulting $\mathbf{A}$ to find a ranking score for each corner. Now, it is possible to take top N corners from if desired.

3. **Calculate orientation of corners with image moments**: ORB uses BRIEF to create feature descriptors, but BRIEF fails in rotated images. Therefore, ORB modifies the BRIEF by adding orientation information. To get orientation, an *image moment* are calculated for each patch $\mathbf{S}^{(n)}$:

$$m_{a,b}(\mathbf{S}^{(n)}) = \sum_{x,y \in \mathbf{S}^{(n)}} x^a y^b I(x,y) \tag{3.4}$$

where $a + b$ defines the order of the moment of n^{th} patch and $I(x,y)$ is the intensity of the pixel at the corner. Next, we calculate the moments of order one:

$$m_{1,0}(\mathbf{S}^{(n)}) = \sum_{x,y \in \mathbf{S}^{(n)}} x \cdot I(x,y) \ , \ m_{0,1}(\mathbf{S}^{(n)}) = \sum_{x,y \in \mathbf{S}^{(n)}} y \cdot I(x,y) \tag{3.5}$$

Then, we get the orientation of the patch $\mathbf{S}^{(n)}$:

$$\theta(\mathbf{S}^{(n)}) = atan2(m_{0,1}, m_{1,0}) \tag{3.6}$$

4. **Form BRIEF descriptors with their corresponding orientation**: Once the top N corners and their orientations are detected, descriptions can be formed with BRIEF.

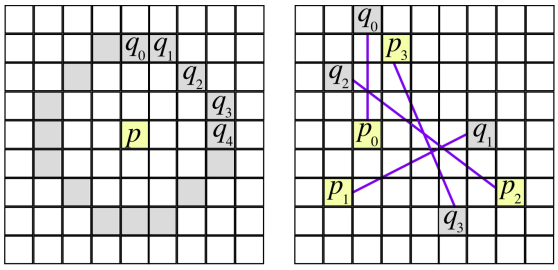


Figure 3.2.2: In order to form BRIEF descriptor, one needs to create group of pixel pairs from patch $\mathbf{S}^{(n)}$. There are two common ways to create pair. As seen in the left figure, we take the feature point and pair with pixels around the circle patch. In the right figure, another method where we pair randomly selected pixels can be seen. ORB prefers the method on the right Klette 2014.

To do so, we randomly (normal distribution) selected 256 pairs ($p^{(i)} = (u^{(i)}, v^{(i)}, q^{(j)} = (u^{(j)}, v^{(j)})$) inside the patch $\mathbf{S}^{(n)}$:

$$\mathbf{S}^{(n)} = \begin{pmatrix} p^{(0)}, \dots, p^{(255)} \\ q^{(0)}, \dots, q^{(255)} \end{pmatrix} \tag{3.7}$$

Next, we rotate each $(\mathbf{p}^{(0:255)}, \mathbf{q}^{(0:255)})$ pair points in $\mathbf{S}^{(n)}$ with the corresponding corner's orientation:

$$\mathbf{p}_\theta^{(i)} = \mathbf{R}_\theta \mathbf{p}^{(i)} \text{ and } \mathbf{q}_\theta^{(j)} = \mathbf{R}_\theta \mathbf{q}^{(j)} \tag{3.8}$$

It is important to note that the authors in Rublee et al. 2011 suggested to rotate each point in increments of $2\pi/30$. Therefore, orientation θ is mapped to nearest multiple of $2\pi/30$. To form steered (or rotated) BRIEF descriptors, we compare pixel densities of pair points that are selected randomly:

$$\tau(\mathbf{p}_\theta^{(i)}, \mathbf{q}_\theta^{(j)}) := \begin{cases} 1 & I(\mathbf{p}_\theta^{(i)} < I(\mathbf{q}_\theta^{(j)}), \\ 0 & I(\mathbf{p}_\theta^{(i)} \geq I(\mathbf{q}_\theta^{(j)}) \end{cases}$$

Finally, we sum comparison results with binary form to get the descriptor of the patch $\mathbf{S}^{(n)}$:

$$\mathbf{D}^{(n)} = f(\mathbf{S}^{(n)}) := \sum_{0 \leq i,j \leq 255} 2^{i-1} \tau(\mathbf{p}_\theta^{(i)}, \mathbf{q}_\theta^{(j)}) \tag{3.9}$$

3.3 Feature Matching

Now that we know how to extract a distinct feature and to form a descriptor, we can start building a relationship across images to estimate how the camera moves. Remember that the camera can procedure a video stream consisting of usually ranging from 30 to 60 frames per second. The second task in VO

after extracting features is to form a group of image pairs information between each image pair, continuously. In literature, there are two ways to select image pairs: frame to frame or keyframe to frame. In the former case, one groups consecutive frames across video stream. In the latter case, one selects a reference frame and keep matching it with subsequent frames as long as the pair has a sufficient amount of feature matchings. The latter has certain advantages over the former; however, we choose the former as we wish to model the uncertainty of our motion estimation algorithm as accurate as possible.

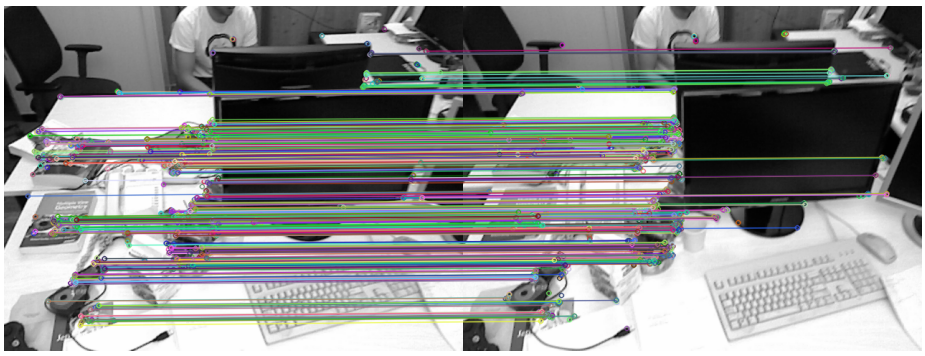


Figure 3.3.1: An example feature matching scene from TUM RGB-D (freiburg xyz) dataset with ORB.

All being said, let's assume we pair consecutive images $(\mathbf{I}^{(k)}, \mathbf{I}^{(k+1)})$. In each image, we gather feature descriptors $(\mathbf{D}^{(1:n,k)}, \mathbf{D}^{(1:m,k+1)})$ that pass through our ORB filter. The goal is to find the feature correspondences based on their rotated BRIEF descriptor values. Again, there are many efficient ways to perform this matching task such as FLANN (Fast Library for Approximate Nearest Neighbors). Instead, in this thesis, we choose the Brute-Force matching algorithm, which produces fewer outliers but it is less efficient in terms of time complexity. The Brute-Force, as its name states, is a straightforward technique that compares each $\mathbf{D}^{(1:n,k)}$ descriptors in k^{th} image with $\mathbf{D}^{(1:m,k+1)}$ descriptors in $k+1^{th}$ image by calculating *Hamming* distance:

$$d_h(\mathbf{D}^{(i,k)}, \mathbf{D}^{(j,k+1)}) = \mathbf{D}^{(i,k)} \oplus \mathbf{D}^{(j,k+1)} \tag{3.10}$$

where $\oplus$ corresponds to an 'exclusive or' logic operation. After calculating Hamming distances, the minimum distance are kept as best matches. On top of that, we perform cross-check validation by ensuring that matches with value (i, j) such that i^{th} descriptor in image k has j^{th} descriptor in image $k+1$ as the best match and vice-versa.

3.4 Outlier Rejection

In reality, not all feature matches are correct, and it is critical that we detect wrong ones as the optimization algorithm that estimates the camera motion is sensitive to even a small number of wrong matches. In technical terms, we call these wrong matches *outliers* (or *false positives*). Hence, we need an algorithm to reject those outliers from *inliers*. The most common way is to

use RANSAC Fischler and Bolles 1981, which is an abbreviation to Random Sample Consensus. RANSAC is an iterative algorithm which fits the desired model with the presence of outliers by selecting a subset of dataset randomly and improving parameters of model each iteration. Note that RANSAC works well if at least half of the dataset contains inliers.

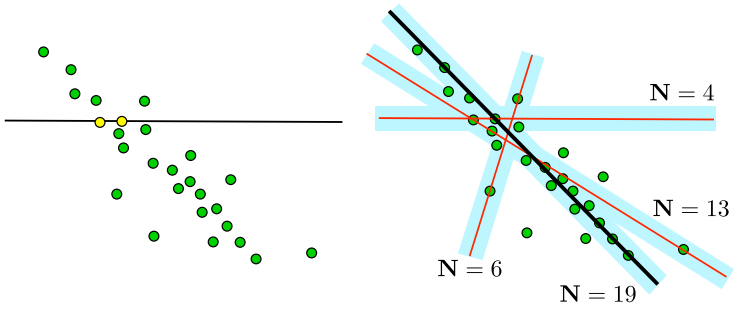


Figure 3.4.1: RANSAC identifies the outliers by fitting the parameters of the homography iteratively. For simplicity, the figure depicts the outlier rejection scenario with a simple line fitting example instead of homography. As seen in the figure, after the number of iteration $N = 14$, RANSAC is able to fit line by excluding the outliers in the optimization process. Note that we can still have outliers within the RANSAC inlier threshold which is the area shaded with light blue color.

In section 2.1, we discussed how to model rotation and translation motion along with the intrinsic matrix. Parameters that we aim to identify are elements of projection matrix in notation 2.3. Similarly, the model being fitted in RANSAC case is elements of a *homography matrix*, which transforms a 2D image point to another 2D image point. Remember that we have feature matches from image pairs. One of the pair is the transformed version of the other pair. We can model this relationship in the following way:

$$k \begin{bmatrix} \hat{u}^{(i)} \\ \hat{v}^{(i)} \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} u^{(i)} \\ v^{(i)} \\ 1 \end{bmatrix} \tag{3.11}$$

The goal is to fit parameters of $\mathbf{H}$ with the selected subset of the matching under the condition where majority of selected point matchings are inliers. In this way, we can easily detect outliers by testing whether they fit to model parameters or not. To make notation 3.11 more obvious, we form linear system of equations:

$$\underbrace{\begin{bmatrix} u^{(i)} & v^{(i)} & 1 & 0 & 0 & 0 & -\hat{u}^{(i)}u^{(i)} & -\hat{u}^{(i)}v^{(i)} - \hat{u}^{(i)} \\ 0 & 0 & 0 & u^{(i)} & v^{(i)} & 1 & -\hat{v}^{(i)}u^{(i)} & -\hat{v}^{(i)}v^{(i)} - \hat{v}^{(i)} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\mathbf{h}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.12}$$

The first step of RANSAC is to select a subset that contains a minimum number of matching points to determine the parameters of the model. In homography $\mathbf{H}$ case, we need at least 4 point pairs ($\mathbf{u} = (u^{(i)}, v^{(i)}), \hat{\mathbf{u}} = (\hat{u}^{(i)}, \hat{v}^{(i)})$) to solve this linear system of equation. However, due to the noise, we require more than 4 point pairs, but this makes the problem over-determined. Therefore, we might find an approximated solution by solving the least squares problem: In doing so, we minimize so-called *algebraic distance error*:

$$\underset{h}{\operatorname{argmin}} \, ||\mathbf{Ah} - \mathbf{0}||^2. \tag{3.13}$$

In the second step, after estimating $\mathbf{h}$ parameters, we test every matchings that are outside the subset which randomly selected at the first step whether they fit to the model with the certain $\mathbf{d}$ threshold we define:

$$||\hat{\mathbf{u}} - \mathbf{Hu}||^2 > d \tag{3.14}$$

In the third step, we include the points that passed our test procedure in the second step into our subset. In the fourth step, we have another test in which we check whether the number of matching points in our subset is large enough to prove that we include the majority of the inliers. If not, we go back to the first step and repeat the whole process until we fulfill the fourth step. The following listing summarizes the algorithm:

Algorithm 1 Rejecting outlier matches with RANSAC

Input

S: the smallest number of points
N: the number of iteration
d: the threshold used to identify a point which fits the model
T: the number of nearby points to notify that there is a good fit

Output

C: the (consensus) set of inliers

```
1:  procedure RANSAC(S,N,d,T)
2:
3:      while iterations < N do
4:          select random sample subset of S points
5:          estimate parameters to fit homography with S
6:          for each points outside S do
7:              calculate error between estimated point and measured point
8:              if error < d then
9:                  add point into S
10:             if S > T then
11:                 return C = S
```

As it is seen in **??**, there are four empiric parameters that we need to define; S,N,d and T. In order to make the algorithm as efficient as possible, these parameters must be chosen carefully. As we discussed, S is the subset of matchings that we randomly select, and initial value should be at least 4 so that we can solve the least squares problem.

For N, it is insufficient to iterate through every matching points. Thus, we at least select N number of matching points with respect to following condition:

$$\mathrm{N} = log(1-p)/log(1-(1-\epsilon)^s) \tag{3.15}$$

where $p = 0.99$ is the probability of covering all inliers, s is the minimum number of iteration that likelihood of choosing a subset with only outliers and ϵ is the probability that the match is an outlier.

For d, it is chosen empirically if the distribution of outliers is unknown. If it is known, i.e., Gaussian with mean μ and σ, the threshold should be $d = 5.99\sigma^2$ so that there is a 95% probability that the point is an inlier.

For T, we might have a case where we reach the expected ratio of inliers; thus we don't have to iterate through N number of times. That means we can terminate it earlier if the following condition is satisfied:

$$\mathrm{T} = (1-\epsilon)n \tag{3.16}$$

where n is the total number of matching points.

It is crucial to note that we may still have outliers after RANSAC. However, our motion estimation will be greatly improved since the majority of the outliers are removed. Finally, we will discuss how we can utilize the carefully selected features and its matches to estimate the camera motion.

3.5 Pose Estimation

Pose estimation is the core part of the VO system. After extracting and matching features, we finally are ready to compute *transformation* information. The transformation corresponds to relative camera motion between two images that are recorded in different poses (see figure-3.5.1). Let's assume; we have consecutive camera poses $\mathbf{x}_k = [\mathbf{p}_k, \mathbf{q}_k]$ and $\mathbf{x}_{k+1} = [\mathbf{p}_{k+1}, \mathbf{q}_{k+1}]$ where $\mathbf{p} = [p_x, p_y, p_z]^T$ is the position of the camera in $\mathbb{R}^3$ and $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ is the orientation of the camera in quaternion form in $SO(3)$. Notice that, in camera model chapter 2, we use rotation matrix to represent orientations, which made it convenient to combine intrinsic matrix and extrinsic matrix into a single projection matrix (see notation 2.5) so that we optimize for parameters of the single matrix. On the other hand, when estimating relative motion, we use quaternions to represent orientations, which is less intuitive way but has certain advantage over rotation matrix such as requiring less storage. As a result, we can represent the transformation $\mathbf{T}_{k,k+1}$ $(= [\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}])$ between two camera poses $(\mathbf{x}_k, \mathbf{x}_{k+1})$ with the rotation $\mathbf{q}_{k,k+1}$ in $SO(3)$ and the translation $\mathbf{t}_{k,k+1}$ in $\mathbb{R}^3$. That being said, one can formulate the transformation $\mathbf{T}_{k,k+1}$ between two camera pose if the initial pose $\mathbf{x}_0$ is known. Then, let's first find the camera position:

$$\mathbf{p}_{k+1} = \mathbf{q}_{k,k+1} \otimes \mathbf{p}_k \otimes \mathbf{q}^*_{k,k+1} + \mathbf{t}_{k,k+1} \tag{3.17}$$

where $\mathbf{q}_{k,k+1} \otimes \mathbf{p}_k \otimes \mathbf{q}^*_{k,k+1}$ is the *hamilton product* that rotates the camera position at the k^{th} pose and $\mathbf{t}_{k,k+1}$ is the simple vector addition that translates the camera position. Next, the camera orientation can be found as follows:

$$\mathbf{q}_{k+1} = \mathbf{q}_{k,k+1} \otimes \mathbf{q}_k \tag{3.18}$$

where $\mathbf{q}_{k,k+1} \otimes \mathbf{q}_k$ is the product of two quaternions that the former is the rotation and that the latter is the orientation of the camera at the k^{th} pose.

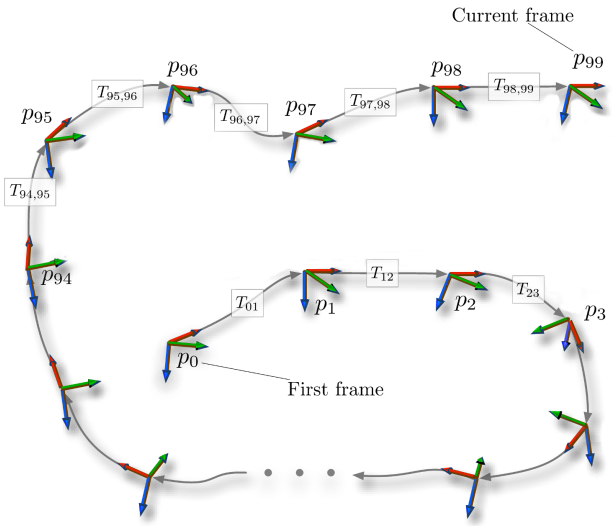


Figure 3.5.1: VO estimates the ego-motion of the camera by relative poses estimated from image pairs. Thus, it assumes that the initial pose p_0 of the camera is known. Transformation $T_{0,1}$ from one image to another corresponds to the camera movement from one position to another. As long as VO is able to detect a sufficient number of features, it keeps estimating relative transformations. With all of these transformations, one can build a trajectory $(T_{0,1}, \ldots, T_{98,99})$ which the camera follows.

The ultimate goal in VO is to compute transformation $\mathbf{T}_{k,k+1}$ in multiple consecutive images and concatenate them to build a trajectory of the camera. As a consequence, we can track any agent on which the camera is placed rigidly. For example, concatenated transformation $\mathbf{T}_{0,n}$ can be used to calculate n^{th} camera pose that is relative to the initial pose:

$$\mathbf{p}_n = \mathbf{q}_{n,n-1} \otimes (\ldots (\mathbf{q}_{2,1} \otimes (\mathbf{q}_{1,0} \otimes \mathbf{p}_0 \otimes \mathbf{q_{1,0}}^* + \mathbf{t}_{1,0}) \otimes \mathbf{q}_{2,1}^* + \mathbf{t}_{2,1}) \ldots) \otimes \mathbf{q}_{n,n-1}^* + \mathbf{t}_{n,n-1} \tag{3.19}$$

$$\mathbf{q}_n = \mathbf{q}_{n,n-1} \otimes \cdots \otimes \mathbf{q}_{2,1} \otimes \mathbf{q}_{1,0} \otimes \mathbf{q}_0 \tag{3.20}$$

To find transformation, we take advantage of image features as they can inform us how the camera moves if we detect them across multiple frames. All the aforementioned step such as feature extraction and feature matchings are performed so that we can compute relative motion. Similar to projection matrix 2.28 in camera calibration, we utilize the least squares method for estimating an approximated transformation information due to the noise.

So far, we discussed how we could process images so that we have adequate information to compute camera pose. However, we only mentioned 2D image features. To estimate the pose in the 3D world, we require corresponding depth

information for features. Note that there are methods that retrieve relative scale information using only 2D image features and its epipolar constraints with monocular cameras, but we are interested in having metric depth information rather than relative scale in this thesis. Therefore, we have two common choices regarding camera types: Stereo Camera or RGB-D Camera. In our experiments, we experimented on RGB-D Camera to retrieve the depth information.

At this point, one generally has two ways to compute relative camera poses. and the reason we have different kinds of way to compute transformation arises from the cost function we define for the least squares problem. In the end, all we wish to find a good model for our optimization problem so that we settle on the best possible local minimum. The design choice for cost function comes from the fact that we build our cost function either on $\mathbb{R}^2$ space (image plane) or $\mathbb{R}^3$ space (camera coordinate system). Therefore, in VO literature, there are two different cost functions for modeling the least problem:

- 3D-to-2D correspondences,
- 3D-to-3D correspondences.

The 2D term refers to 2D image features. Whereas, the 3D term refers to 3D point features that composed of back-projection of 2D features and depth information. Note that this thesis does not engage with 2D-to-2D correspondences method since it is used in monocular camera; thus it will not be discussed here. On the other hand, we will discuss and compare 3D-to-2D correspondences and 3D-to-3D correspondences. Although 3D-to-2D correspondences outperform 3D-to-3D correspondences in VO, we show the accuracy of 3D-to-3D correspondences can be significantly improved if the uncertainty of the 3D point features is modeled properly and used in the optimization process. On the plus side, this technique allows us to propagate the uncertainty of the 3D feature points to get the uncertainty of estimated pose. With being said, let's explain both methods.

3D-to-2D Correspondences

Remember that, after feature matching step for the consecutive image frames, we have only2D-to-2D correspondences information and the transformation which we wish to compute is in $\mathbb{R}^3$. Therefore, we require transformation involving in 3D point features. That being said, we can estimate transformation 3D-to-2D Correspondences in four steps:

1. Back-project 2D image features from $k + 1^{th}$ frame to form 3D point features,
2. Back-transform 3D point features from $k + 1^{th}$ frame towards k^{th} frame with transformation matrix,
3. Reproject the back-transformed 3D point features onto the k^{th} image plane,
4. Minimize 2D Euclidean distance error between reprojected and measured 2D image features.

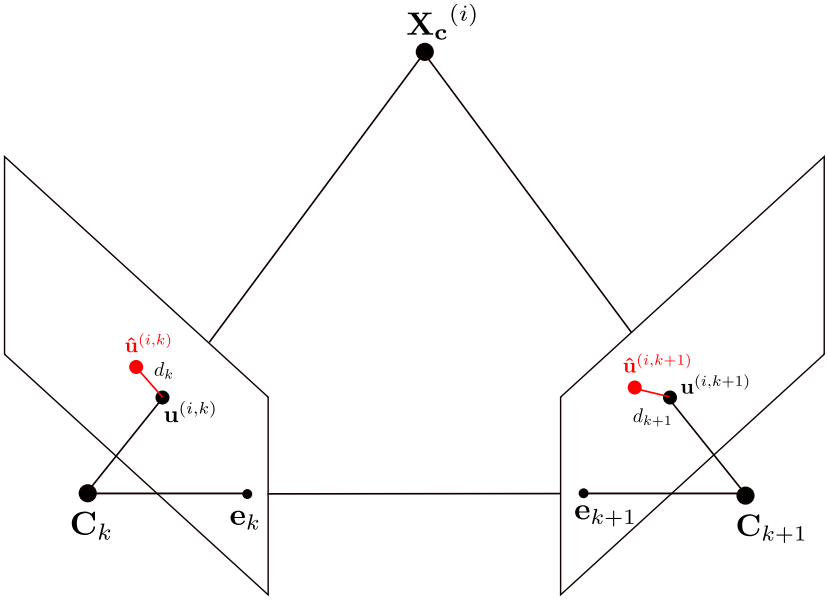


Figure 3.5.2: Assuming that we have two frames taken in the different poses. What 3D-to-2D correspondences method does is that it takes the pixel $\mathbf{u}^{(i,k+1)}$ measurement from $k+1^{th}$ back-projects to get the 3D point feature, then rotates and translates with the transformation information, and projects the 3D feature to the 2D feature as $\hat{\mathbf{u}}^{(i,k)}$ on image plane at k^{th} frame. Afterward, one will get Euclidean error distance $d_k \in \mathbb{R}^2$ between the reprojected pixel and the measured pixel. This behavior applies for reprojecting image from k^{th} to $k+1^{th}$ frame to calculate d_{k+1}. For the sake of efficiency, most VO algorithms assume that measurement errors occur in only one image pair and only uses d_k to minimize the error, but not both $d_k + d_{k+1}$. Note that this choice might decrease the accuracy of the pose estimation.

To formulate the problem more clearly, let's assume we have a 3D point feature $\mathbf{X_c}^{(i)} = [x, y, z]^T$ in camera coordinate system and we measure the projections of this exact feature point as a 2D image feature, $\mathbf{u}^{(i,k)} = [u, v]^T$ and $\mathbf{u}^{(i,k+1)} = [u, v]^T$ on subsequent camera poses k^{th} and $k+1^{th}$ respectively. What we also know is that we can back-project measured 2D image features as 3D point features using projection matrix (see notation 2.12) and depth information. Let's write again projection function that converts 3D point features to 2D image features: $\mathbf{u}^{(i)} = \mathbf{K}\mathbf{X}^{(i)}$. One can also back-project 2D image features to 3D point features $\mathbf{K}^T\mathbf{u}^{(i)} = \mathbf{X}^{(i)}$. Now, we can formulate the previously discussed four steps of 3D-to-2D correspondences as follows:

1. $\mathbf{X}^{(i,k+1)} = \mathbf{K}^T\mathbf{u}^{(i,k+1)}$
2. $\hat{\mathbf{X}}^{(i,k)} = f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}^{(i,k+1)}) = \mathbf{q}_{k,k+1} \otimes \mathbf{X}^{(i,k+1)} \otimes \mathbf{q}^*_{k,k+1} + \mathbf{t}_{k,k+1}$
3. $\hat{\mathbf{u}}^{(i,k)} = \mathbf{K}\hat{\mathbf{X}}^{(i,k)}$
4. minimize $\sum_i ||\mathbf{u}^{(i,k)} - \hat{\mathbf{u}}^{(i,k)}||^2$ where $\mathbf{u}^{(i,k)}$, $\hat{\mathbf{u}}^{(i,k)} \in \mathbb{R}^2$

The second and third steps can be encapsulated to a function f so that we form the problem as an optimization problem in the following form:

$$\underset{\mathbf{T^*}_{k,k+1}=[\mathbf{t^*}_{k,k+1},\mathbf{q^*}_{k,k+1}]}{\operatorname{argmin}} \sum_i ||\mathbf{x}^{(i,k)} - f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}^{(i,k+1)})||^2 \tag{3.21}$$

3D-to-3D Correspondences

Another way of modeling the cost function is to utilize only 3D point features correspondences, and one can estimate transformation with 3D-to-3D correspondences in three steps:

1. Back-project both 2D image features from k^{th} and $k+1^{th}$ frames as 3D point features,
2. Back-transform the back-projected 3D point features from $k+1^{th}$ frame,
3. Minimize 3D Euclidean error distance between back-transformed and measured 3D point features.

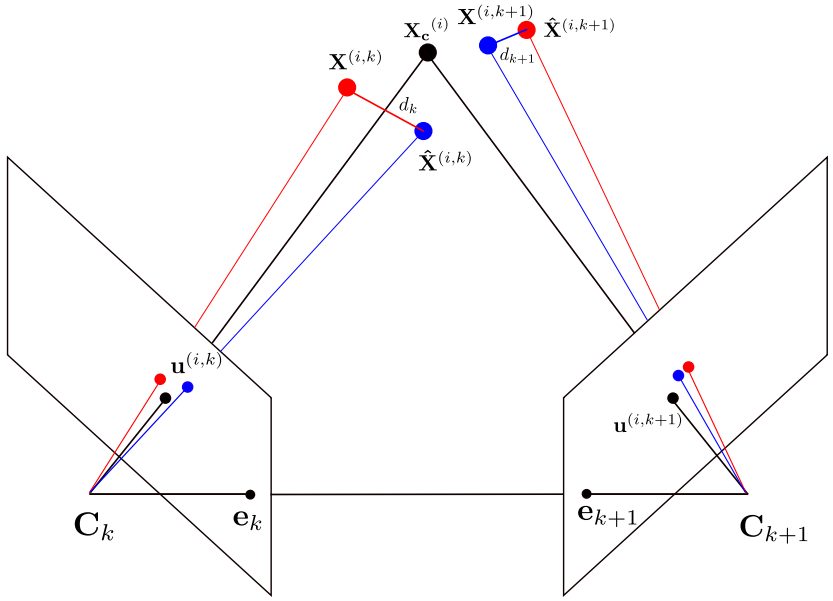


Figure 3.5.3: 3D-to-3D correspondences method calculates the Euclidean error distance in 3D space. For example, all image features $\mathbf{u}^{(i,k)}, \mathbf{u}^{(i,k+1)}$ from both frames are back-projected to get 3D point features $\mathbf{X}^{(i,k)}, \mathbf{X}^{(i,k+1)}$. Then, $\mathbf{X}^{(i,k+1)}$ is rotated and translated with the transformation information $\mathbf{T}_{k,k+1}$ to align the same 3D point feature $\hat{\mathbf{X}}^{(i,k)}$ with respect to k^{th} camera's centre.In this way, one can calculate the Euclidean distance error $d_k \in \mathbb{R}^3$.Reverse behavior applies on moving features from k^{th} to $k+1^{th}$ frame to calculate d_{k+1} as well.

The similar formulation that we previously did for 3D-to-2D correspondences applies for the 3D-to-3D correspondences as well but only with few changes. Let's again assume that we have two corresponding 2D image features, $\mathbf{u}^{(i,k)}$ and $\mathbf{u}^{(i,k+1)}$. However, rather than minimizing error on the 2D image plane, we want to minimize them on 3D space. Therefore, we need to back-project both image features. With that in mind, we can formulate the three steps of 3D-to-3D correspondences as follows:

1. $\mathbf{X}^{(i,k+1)} = \mathbf{K}^T\mathbf{u}^{(i,k+1)}$ and $\mathbf{X}^{(i,k)} = \mathbf{K}^T\mathbf{u}^{(i,k)}$
2. $\hat{\mathbf{X}}^{(i,k)} = f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}^{(i,k+1)}) = \mathbf{q}_{k,k+1} \otimes \mathbf{X}^{(i,k+1)} \otimes \mathbf{q}^*_{k,k+1} + \mathbf{t}_{k,k+1}$
3. minimize $\sum_i ||\mathbf{X}^{(i,k)} - \hat{\mathbf{X}}^{(i,k)}||^2$ where $\mathbf{X}^{(i,k)}$, $\hat{\mathbf{X}}^{(i,k+1)} \in \mathbb{R}^3$

The second step can be encapsulated to a function f to form the optimization problem:

$$\underset{\mathbf{T}^*{}_{k,k+1}=[\mathbf{t}_{k,k+1},\mathbf{q}_{k,k+1}]}{\operatorname{argmin}} \sum_i ||\mathbf{X}^{(i,k)} - f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}^{(i,k+1)})||^2 \tag{3.22}$$

In VO literature, this method is usually discarded since it performs poorly comparing to 3D-to-2D correspondences. However, we proved that it can be improved if feature covariances are properly estimated and included in the optimization problem.

Chapter 4

Motivation

The essence of a mobile robot is autonomy. A fully autonomous robot first *senses* its environment, then *interprets* the collected data and finally *acts* based on the insights that it gathered over time. If we look at nature, throughout the evolution, certain animals gained certain abilities regarding spatial awareness in all sorts of ways. For instance, homing pigeons navigate by magnetic field, bats use sound to map their surroundings or bees smell with their special chemoreception to find their way back home. Even though each sense helps us with particular importance, we humans are good at navigating ourselves by relying on our vision. As the incredible capability of a human eye (locating and recognizing objects within milliseconds even in ambiguous situations) stays a mystery, robotics researchers has always been keen on exploiting the computer vision field, especially Visual Odometry and Visual SLAM.

Visual Odometry outputs relative ego-motion information similar to IMU and wheel odometry. Apart from absolute sensor measurments like known a priori set of landmarks or GPS, all of three low-level sensors are heavily used in localization applications as relative sensors. Nowadays, sensor fusion applications like SLAM and 3D reconstruction are designed accordingly to compensate each other's biases in order to obtain the best possible result. In the case of an accurate and robust sensor fusion framework, one selects sufficient amount absolute and relative positioning sensors. The ultimate goal is to fuse both relative and absolute sensor types along with their noise characteristics in a way that the error is minimized. Hence, it is critical to model each sensor's uncertainties. IMU and wheel odometry sensors' uncertainty are modeled by their vendors based on the working principles and worst-case scenario tests beforehand. Whereas, widely used open source VO tools do not provide any uncertainty information about their pose estimations. Even though there are several VO papers Endres et al. 2014, Konolige and Agrawal 2008, Di et al. 2016, Belter, Nowicki, and Skrzypczyński 2018 that deals with uncertainty of RGB-D sensors, they only intend to improve accuracy of the pose estimation, but not to provide any uncertainty information about their estimation. Therefore, my aim, in this thesis, is to build a feature-based RGB-D Visual Odometry system that outputs pose estimations along with their metric uncertainties. As a result, VO system can be treated as a low-level relative motion sensor in a sensor fusion application.

Chapter 5

An Error-Aware RGB-D Visual Odometry

5.1 Related Work

An *error-aware* Visual Odometry estimates an ego-motion of a camera along with the uncertainty of its estimations. Remember that the primary goal of a sensor fusion application is to combine multiple sensors to get a better estimation of its pose. In this case, we need metric uncertainty representation for each sensor. That is why we require a covariance matrix for every pose estimation from all sensors so that we can compensate sensors' biases dynamically throughout the trajectory. Ideally, a good error-aware VO system should tell us how confident its pose estimation at a specific time instance by taking factors, such as a number of detected features and their noises, into account.

To build such an error-aware VO with an RGB-D camera, one needs to be able to model the noise characteristics of the sensor used in the camera. The passive stereo camera and its error propagation application for the uncertainty estimation are already well-studied problems in Leo, Liguori, and Paolillo 2011 and Miura and Shirai 1993. However, in this thesis, I am interested in implementing the spatial error propagation of the active stereo for the VO application since there is, to my knowledge, little work being done.

Because our VO algorithm is a feature-based technique, the uncertainty of image features must be modeled. Generally, this is known as a quantization error referring to a pixel noise Richard Hartley 2003 which is assumed to be maximum half pixel and normally distributed in literature. To represent the pixel uncertainty of features in the 3D world, the conic ray model Solà 2007 is used by build on the pinhole model of a camera.

Moreover, one should consider depth uncertainty as well. In this perspective, In Mallick, Das, and Majumdar 2014, authors gathered existing studies about the depth camera's noise of Kinect. Since Kinect has two version that uses different technologies for measuring depth information, researchers compared the performance of both versions in Wasenmüller and Stricker 2017 and Kinect et al. 2015. Another extensive study is Khoshelham and Elberink 2012, outlining both a theoretical and experimental background for accuracy and resolution of the Kinect's depth camera based on the geometrical model. Conversely, Choo

et al. 2014 conducts a comprehensive statistical analysis to build a high-order polynomial mathematical model for the standard deviation of depth error.

Even though the studies Park et al. 2012 and Nguyen, Izadi, and Lovell 2012 that deals with Kinect model the uncertainty, they don't utilize them on a VO system. In Park et al. 2012, the author contributed an important methodology on how to model confidence ellipsoids of 3D point features measured by Kinect. In this paper, the standard deviation of the depth noise is assumed to be constant. In fact, it increases with distance from camera quadratically. Nguyen, Izadi, and Lovell 2012, however, addresses this issue by defining a quadratic function whose parameters are identified with the optimization process. They even evaluate the performance of their proposed model for pose estimation, which is quite relevant to our task, but settings of their experimentation are unclear since they focus mostly on 3D reconstruction.

For applications such as SLAM and 3D Reconstruction, there are other papers Endres et al. 2014, Konolige and Agrawal 2008, Di et al. 2016 that incorporate uncertainty models, but they assume half pixel Gaussian noise for all features. Naturally, this increases their localization accuracy. However, it still unclear how one can propagate covariance of the detected features to the covariance of the estimated pose in the presence of outliers of matched features. The only exception that attempts to model the pixel uncertainty of features is Belter, Nowicki, and Skrzypczyński 2018. They implemented a so-called reserve-SLAM simulation tool to identify the pixel noise caused by feature descriptor. They too did not propagate feature covariance to get pose covariance. They only used features covariance to improve pose estimation accuracy. In this thesis, I am interested in propagating the covariance of the matched features to the covariance of the estimated camera pose in the presence of outliers.

5.2 Implementation Details of CoVO

CoVO (Covariance-enabled Visual Odometry) is the proposed error-aware VO algorithm based on an RGB-D camera. It utilizes 3D-to-3D correspondences method along with the uncertainty of 3D point features to estimate relative camera poses. Most importantly, it exploits the metric covariance of 3D point features to propagate the pose covariance. An overview of the CoVO pipeline is illustrated in figure 5.2.1.

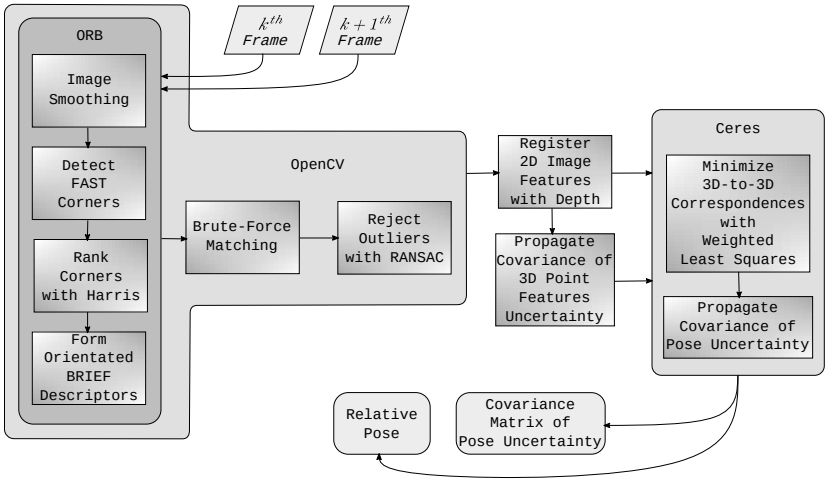


Figure 5.2.1: CoVO: The source code of the implementation can be found in: https://github.com/ugurbolat/CoVO

Many VO systems require parameter tuning process and have algorithmic variations. Thus, building a reliable VO can be a daunting task. To minimize implementation errors and development time, we take advantage of two commonly used open source libraries; i.e., OpenCV (Itseez 2019) for handling image feature manipulations and Ceres (Agarwal and Mierle 2019) for optimization. We chose these two libraries because they are mature open source projects that ensure efficiency and allow required customization. In the following part of this section, we will explain the essential steps that we take to build CoVO:

1. **Extracting Feature:** We choose ORB due its efficiency and repeatability. Before extracting ORB features, we convert RGB images to grayscale. Then, we detect corners with FAST-9 that takes a patch of circular radius as 9 pixels around the corner. Next, the detected corners are ranked with Harris filter according to their image derivatives. In this way, we can query top N corners. Finally, orientated BRIEF descriptors are formed by randomly selected pairs. Note that these functionalities are already available in OpenCV and they are built based on Rublee et al. 2011.

2. **Matching Features:** After extracting ORB features from consecutive images, we matched them with Brute-Force algorithm that uses Hamming window for the comparison. Before calling two features as a match, the cross-check is performed to make sure both features are identified as a match in their own comparison set. In the end, a filtering operation is applied on the matches by removing worst 10% matches based on the distance calculated by Hamming window.

3. **Rejecting Outliers:** With the remaining matches, we apply RANSAC to reject outliers. Note that we choose RANSAC threshold to be 10 pixels. Also, remember that we will have pseudo inliers even after applying RANSAC and the pixel errors are not particularly bounded with 10 pixels. The effect of pseudo inliers in pixel uncertainty is discussed in section 6.3.1.

4. **Register 2D Features With Depth:** At this step, we simply combine inlier 2D features with their corresponding depth information. It is critical to note that Kinect has invalid depth measurement which is measured as 0 disparity level for certain regions of the object surface. Thus, we remove 2D features having invalid depth value from match set. Plus, we remove 2D features that have a depth value greater than $5m$ since it is Kinect's accurate depth distance range.

5. **Preparing Covariance Matrix of 3D Feature Points:** For every inlier matched features, we propagate pixel uncertainties $\mathbf{Q_{uvz}}$ from image plane to camera coordinate system $\mathbf{Q_{xyz}}$ with the Jacobian of back-projection function $\mathbf{J_{bp}(u)}$ (see notation 5.22). This step is crucial for both improving for pose estimation results and estimating pose covariance.

6. **Minimizing 3D-to-3D Correspondences With Weights:** To able to take advantages of feature covariances of both consecutive images, we expand the residuals by defining both back- and forward transformation function (see notation 5.16). It improves the optimization process because we take measurement error in both images. The technique is taken from [Richard Hartley 2003, p.101]. Then, the weighted least squares problem is solved by Levenberg-Marquardt by minimizing the error between 3D-to-3D correspondences to calculate relative camera pose.

7. **Calculating Covariance Matrix of the Estimated Pose:** After completing least squares optimization process, we calculate the covariance $\mathbf{Q_{tq}}$ of the estimated relative camera pose by propagating it from the feature covariances $\mathbf{Q_{xyz}}$ with the Jacobian of residuals function $\mathbf{J'(x^*)}$ at the optimal solution (see notation 5.20). Due to the non-linearity of the projection function, an approximated version of the error propagation law produces overconfident covariance estimations. Thus, we scale the resulting $\mathbf{Q_{tq}}$ with ϕ to keep the estimator conservative. The reason why we have this heuristic parameter is discussed in section 6.3.2.

Note that all of the parameters mentioned above of CoVO are given as a list in appendices 9.6. In the end, building such a VO system require many tuning parameters and many VO algorithms do not emphasize the effect of this phenomena. The fact that we aim to develop an error-aware VO that produces metric pose covariances means that each parameter must be examined. In the following sections of this chapter, we will discuss the necessary components and parameters of the CoVO pipeline to highlight what is required for an error-aware system.

5.3 Modeling Uncertainty of RGB-D Camera

The main reason why conventional VO applications do not provide any uncertainty information (namely covariance matrix) for its pose estimation is that it is hard to model error characteristics of the whole VO pipeline. This is because we perform many preprocessing, each of which eventually introduces different types of error. What we aim is to define potential error source of the VO and

to model them. Thus, we will investigate the noise characteristic of sensors in RGB-D camera in this section. In particular, we examine Kinect, having three sensors: RGB camera, IR camera, and IR laser projector. In our experiments and evaluations, we assume that these sensors are calibrated such that there is no registrations error when mapping RGB pixels to disparity pixels.

Furthermore, we assume that measurements with these sensors are independent of each other. Under this assumption, we split the source of errors into two categories: feature related and depth-related uncertainties. The former is caused by point feature location and matching algorithms. The latter is caused by the depth camera sensor. Let's see how we can model these two error sources and how to form an uncertainty model for Kinect so that we estimate metric covariance matrices for each relative camera pose.

5.3.1 Feature Related Uncertainty

Our VO pipeline heavily relies on the detected features. For a VO algorithm to operate reliably, it is expected that you will have a video stream that has small translation and rotation differences at the consecutive images. Thus, when pairing images to find common features, we expect not to have high-degree rotation or a large amount of scaling on image features so that matching algorithm would not suffer from a high number of outliers. Under these circumstances, we identify two main error sources related to features; i.e., interest point location uncertainty and outliers in feature matching.

To understand these two types of errors, we need to remind ourselves how we detect and describe features section 3.2 in the first place. In ORB, FAST corner filter is performed by selecting pixel coordinates of an interest point and comparing it with its surrounding pixels. In an ideal scenario where we match features in consecutive image perfectly, we would assume that the error will be a half pixel due to the quantization process of the RGB camera. The ideal scenario breaks once we have outliers. In other words, if we had perfect matches, all the reprojected features would be situated closely to their matches and the error distance between matches would be a half pixel. However, we still have outliers even after applying RANSAC. Thus, the error distance for those outliers would be greater that one pixel. In this thesis, we call those outliers that appear after RANSAC *pseudo inliers*. Moreover, instead of taking pixel errors a half pixel, we investigate the pixel error caused by pseudo inliers in section 6.3.1 and treat them as pixel uncertainty.

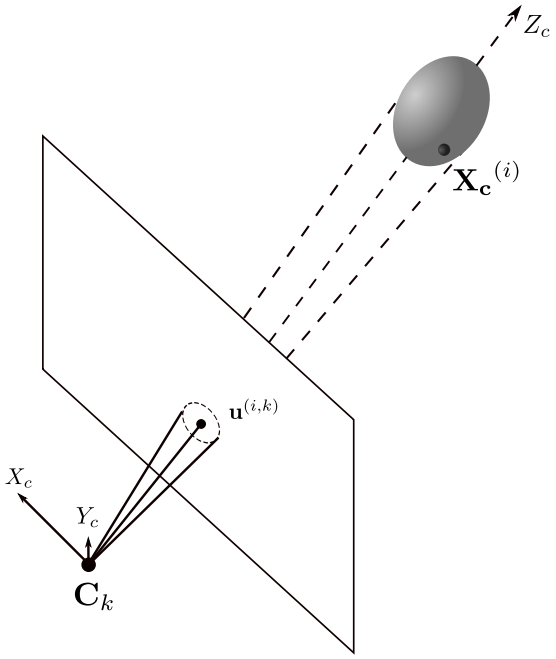


Figure 5.3.1: The conic ray model is inspired from Solà 2007 and we expand a confidence ellipse to a confidence ellipsoid by adding depth uncertainty since we use a RGB-D camera.

Having all these in mind, let's model uncertainty related to features. First of all, remember that 3D point features in the camera coordinate system are projected to 2D points on an image plane with the pinhole model from section 2.1. When the aperture of a digital camera opens, it captures incoming light rays using its light detector sensor (typically CMOS image sensors) and turns them into electrical signals, which is an over-simplified definition of a camera. With the help of the pinhole model, one can build a model for light ray coming from a 3D point. Because we have the errors as mentioned above, we can't measure the exact location of the 3D point feature. However, it is likely that the point is within the dash line region (see figure 9.3.1) As a consequence, we form an uncertainty region which we call *conic ray*. Within this conic ray, we can represent the uncertainty of a point with a *confidence* ellipsoid if the depth uncertainty is included. To formulate this uncertainty, we need to find parameters of the confidence ellipsoid which can be represented with multi-dimensional Gaussian distributions in 3D space:

$$\mathbf{g}_{xyz}(\mathbf{X}^{\mathcal{C}}) = \frac{1}{\sqrt{(2\pi)^3|\mathbf{Q_{xyz}}|}} \exp(-\frac{1}{2}(\mathbf{X}^{\mathcal{C}} - \mathbf{m_x}^{\mathcal{C}})^{\intercal}\mathbf{Q_{xyz}}^{-1}(\mathbf{X}^{\mathcal{C}} - \mathbf{m_x}^{\mathcal{C}})) \quad (5.1)$$

where $\mathbf{X}^{\mathcal{C}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ is the real position of the point in the camera coordinate system, $\mathbf{m_x}^{\mathcal{C}} = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix}$ is the measured position, and $\mathbf{Q_{xyz}} = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y & \sigma_x\sigma_z \\ \sigma_y\sigma_x & \sigma_y^2 & \sigma_y\sigma_z \\ \sigma_z\sigma_x & \sigma_z\sigma_y & \sigma_z^2 \end{bmatrix}$

is the covariance of measurement error. Notice that errors in x, y and z direction are correlated to each other as the ellipsoid can be tilted with respect to the focal point of the camera. With regards to measurement error in x and y direction, we only have indirect knowledge since we measure features on image plane in u and v direction. As to measurment error in z direction, we have direct knowledge (we assume that disparity data is already converted to depth information). As a matter of fact, (u, v, z) form a special space $\mathbb{R}^3$ called *disparity image space*. To get errors in x and y directions, we need to propagate pixel uncertainties from image plane to camera coordinate system. Thus, we remind ourselves with the back-projection function:

$$\mathbf{X}^{\mathcal{C}} = \mathbf{F_{bp}}(\mathbf{u}) \tag{5.2}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{z}{f_x} & 0 & -\frac{zc_x}{f_x} \\ 0 & \frac{z}{f_y} & -\frac{zc_y}{f_y} \\ 0 & 0 & z \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5.3}$$

One can represent the probability distribution of the pixel and depth error with another multivariate Gaussian distribution formed by pixel uncertainties.

$$\mathbf{g}_{uvz}(\mathbf{X}^{\mathcal{D}}) = \frac{1}{\sqrt{(2\pi)^3|\mathbf{Q_{uvz}}|}} \exp(-\frac{1}{2}(\mathbf{X}^{\mathcal{D}} - \mathbf{m_u}^{\mathcal{D}})^{\intercal}\mathbf{Q_{uvz}}^{-1}(\mathbf{X}^{\mathcal{D}} - \mathbf{m_u}^{\mathcal{D}})) \tag{5.4}$$

where $\mathbf{X}^{\mathcal{D}} = \begin{bmatrix} u \\ v \\ z \end{bmatrix}$ is the real pixel coordinates and the corresponding depth value in disparity image space, $\mathbf{m_u}^{\mathcal{D}} = \begin{bmatrix} m_u \\ m_v \\ m_z \end{bmatrix}$ is the noisy pixel measurements along with the noisy depth measurement, and $\mathbf{Q_{uvz}} = \begin{bmatrix} \sigma_u^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}$ is the covariance of these measument errors as pixel and depth measurements are considered independent.

To convert disparity image space to camera coordinate system, we utilize the error propagation low. In this respect, we need the partial derivative of the back-projection function:

$$\mathbf{J_{bp}}(\mathbf{u}) = \frac{\partial \mathbf{F_{bp}}(\mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} \frac{z}{f_x} & 0 & (\frac{u}{f_x} - \frac{c_x}{f_x}) \\ 0 & \frac{z}{f_y} & (\frac{v}{f_y} - \frac{c_y}{f_y}) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

Then, we propagate the pixel covariances as follows:

$$\mathbf{Q_{xyz}} = \mathbf{J_{bp}}(\mathbf{u})^T \mathbf{Q_{uvz}} \mathbf{J_{bp}}(\mathbf{u}) \tag{5.6}$$

Apart from pixel uncertainties, what we haven't discussed is how we get the σ_z depth uncertainty. It deserves own explanation so we will cover in the next section.

5.3.2 Depth Related Uncertainty

Modeling noise in depth measurements is more complicated than an RGB camera. In section 2.2, remember that we explained how structured IR light speckles are projected onto an object so that IR camera can capture its deformed patterns. During this process, many things can go wrong. For example, (1) specific ambient background would make Kinect suffer from over-saturated disparity image, (2) having multiple Kinect in the same environment can lead to interference issue, (3) multi-path propagation of the light might change the expected illumination, or (4) measuring in dynamic scene might result in improper IR light patterns. All of these non-deterministic events makes it harder to model the uncertainty. However, we assume that operating conditions and environment are chosen carefully to avoid these events as much as possible. What we aim to model in this section is mostly systematic errors in Kinect. In this regard, we rely on the experiments conducted by Nguyen, Izadi, and Lovell 2012.

Kinect's Systematic Depth Noise

The experimental analysis contributed to model Kinect's depth noise. Nguyen, Izadi, and Lovell 2012 calculated the difference between ground truth and Kinect measurements and, found out that there are two types of systematic noise occurring in Kinect's depth measurements: *axial* noise and *lateral* noise. Note that our depth noise model is based on their work.

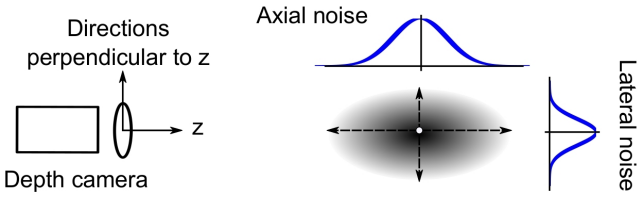


Figure 5.3.2: According to Nguyen, Izadi, and Lovell 2012, axial noise corresponds to noise along the z axis in the camera coordinate system. In other words, it is the depth uncertainty model we search for our conic ray model. On the other hand, lateral noise corresponds to directions perpendicular to the z axis. This noise creates an uncertainty in the pixel coordinates of the disparity image. The figure is taken from Nguyen, Izadi, and Lovell 2012.

To detect the lateral and axial noise, they built an experimental setup with a Kinect that projects its IR speckle patterns onto a planar surface. Then, they collected depth measurements at a different distance to the planar surface positioning at different angles. For calculating the axial noise, they first removed the lateral noise cropping edges. Then, remaining depth region was fitted to a plane that has the minimum error to the ground truth. Finally, they calculated distance difference between measured depth and ground truth. Whereas, the lateral noise was simply calculated by taking pixel difference between fitted straight edge passing through the center of distribution and measured (zigzag-like shape) pixels. The illustration of this experiment is given in figure 5.3.3.

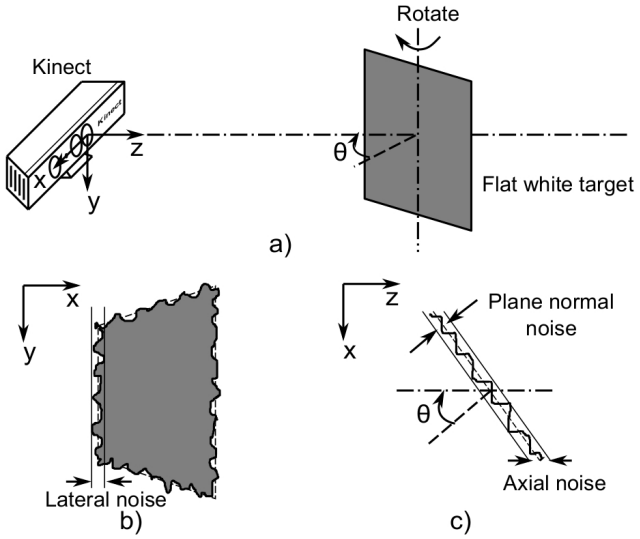


Figure 5.3.3: In order to measure the depth noise, an object that has flat white surface was placed at different known distances to Kinect. During the experiments, they also discovered that depth noise changes when the angle of the object changes with respect to Kinect's focus point. Thus, they included different poses with different angles by rotating the object (a). The effect of lateral and axial noise is depicted in (b) and (c). The figure is taken from Nguyen, Izadi, and Lovell 2012.

After calculating errors between measurements and ground truth, they realized that the axial and lateral noise have different noise characteristics. The lateral noise error distribution stays constant with the distance, while the axial noise distribution gets wider with the increased range.

Besides, the axial noise has another property, which is the response to the different angles. It was observed that the standard deviation of the axial noise increases drastically after 60 degrees. In the light of these experiments, they proposed two empirical models that fit corresponding measurements for the axial and lateral noise. For axial noise, they define a quadratic relationship between the standard deviation of the error and distance along the z-axis.

$$\sigma_z(z, \theta) = 0.0012 + 0.0019 \cdot (z - 0.4)^2, \text{ if } 10^\circ \leq \theta \leq 60^\circ \tag{5.7}$$

where z is the measured depth in meters. Plus, they added a hyperbolic parameter to represent the behavior measurement error beyond 60 degrees:

$$\sigma_z(z, \theta) = 0.0012 + 0.0019 \cdot (z - 0.4)^2 + \frac{0.0001}{\sqrt{z}} + \frac{\theta^2}{(\pi/2 - \theta)^2}, \text{ if } \theta \geq 60^\circ \tag{5.8}$$

For lateral noise, its noise was almost constant with respect to distance along the z-axis and had the similar hyperbolic effect after 60 degrees of angle. Hence, they defined lateral noise with the following equations:

$$\sigma_L(\theta) = 0.8 + 0.035 \cdot \theta/(\pi/2 - \theta) \text{ (in pixels)} \tag{5.9}$$

To validate these experimental noise models' correctness, they implemented a 3D reconstruction and a camera pose tracking scenario. As a consequence, they observed that the noise models improved the overall accuracy of both applications. It is important to note that they cooperated the iterative closest point (ICP) to estimate the camera poses. The ICP is one of many algorithms to solve VO problem. With the ICP, one utilizes all or most of the 3D point clouds instead of selecting a distinct feature in each frame. Now that we know how depth noise is modeled, we can plug it into our noise model:

$$\mathbf{Q_{uvz}} = \begin{bmatrix} \sigma_u^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & (\sigma_z^{(i)}(z,\theta))^2 \end{bmatrix} \tag{5.10}$$

While the axial noise can be embedded as the third dimension along the z-axis, the lateral noise has an indirect relationship to the overall uncertainty of a point feature. This indirect relationship occurs when associating depth pixel coordinates with the pixel coordinates, namely *registration*. Therefore, one can avoid this registration error by applying a smoothing filter on depth images. According to their experiments, the lateral noise of the Kinect is around one pixel. Thus, a 3x3 smoothing filter can be used on extracted features or edges in disparity image.

In short, the probabilistic model, we built with the conic ray, and the confidence ellipsoids, based on normal distribution can now allow us to estimate relative camera poses with the weighted least squares optimization. To construct a cost function for the least squares problem, we need to gather (1) measured pixels, (2) measured depth, (3) calibrated intrinsic camera parameter and most importantly (4) covariance matrices of the measurements. Having covariance matrices for the point feature measurements not only improves the convergences of the optimization algorithm but also enables us to estimate a covariance matrix for the estimated relative pose.

5.4 Pose Estimation with Uncertainties

Before diving into the formulation, it is a good idea to refresh our knowledge about how we estimated relative pose of a camera using 3D-to-3D correspondences in section 3.5. Remember that after pre-processing extracted features, we would have m number of measured feature matches in camera coordinate system with respect to k^{th} and $k+1^{th}$ consecutive camera frames and we stored them as $(\mathbf{X}_k^{(1:m,\mathcal{C})}, \mathbf{X}_{k+1}^{(1:m,\mathcal{C})})$. Note that we drop $\mathcal{C}$ superscript for simplicity and assume that all point features are in the camera coordinate system. The transformation relationship between the k^{th} frame and the $k+1^{th}$ frame was defined with the translation $\mathbf{t}_{k,k+1}$ and rotation $\mathbf{q}_{k,k+1}$ information, which we want to know. As discussed earlier, the main idea was to back-transform the $\mathbf{X}_{k+1}^{(1:m)}$ features onto $\mathbf{X}_k^{(1:m)}$ features so that they are aligned with respect to the same camera frame. Then, we minimize the error while optimizing the translation and rotation. Here, we define the back-transform function as follows:

$$f_b(\mathbf{x}_{k,k+1}, \mathbf{X}_{k+1}^{(i)}) = \mathbf{q}_{k,k+1} \otimes \mathbf{X}_{k+1}^{(i)} \otimes \mathbf{q}_{k,k+1}^{*} + \mathbf{t}_{k,k+1} \tag{5.11}$$

In the traditional VO problem, the residuals function of the optimization problem is defined by the difference only between back-transformed point features from $k+1^{th}$ and measured point features from k^{th}.

$$\mathbf{t}_{k,k+1}^{*}, \mathbf{q}_{k,k+1}^{*} = \underset{\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}}{\operatorname{argmin}} \sum_{i} ||\mathbf{X}_{k}^{(i)} - f_b(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_{k+1}^{(i)})||^2 \tag{5.12}$$

The important part of the error-aware VO that we propose in this thesis lies on having estimated uncertainty of the features and then to propogate it through uncertainty of estimated relative pose. If we utilized our conic ray model, we would have different covariance matrices for each feature. Thus, let's include covariance matrices into the optimization problem:

$$\underbrace{\mathbf{t}_{k,k+1}^{*}, \mathbf{q}_{k,k+1}^{*}}_{\mathbf{x}_{\mathbf{k},\mathbf{k+1}}^{*}} = \underbrace{\underset{\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}}{\operatorname{argmin}}}_{\mathbf{x}_{k,k+1}} \sum_{i} || \underbrace{\mathbf{X}_{k}^{(i)} - f_b(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_{k+1}^{(i)})}_{\mathbf{r}_b^{(i)}(\mathbf{x}_{k,k+1})} ||^2_{\mathbf{Q}_{\mathbf{xyz},k+1}^{(i)}} \tag{5.13}$$

However, this residuals function builds on the assumption that features from the k^{th} frame are noise-free and features from the $k+1^{th}$ frame are noisy. Thus, we could only weight with $\mathbf{Q}_{\mathbf{xyz},k+1}^{(1:m)}$ as seen in the above equation. In reality, we know that the features from both frames are noisy. We can consider including forward-transform function such that we also take all covariance matrices $(\mathbf{Q}_{\mathbf{xyz},k}^{(1:m)}, \mathbf{Q}_{\mathbf{xyz},k+1}^{(1:m)})$ for all features into account during optimization. In this case, one needs to calculate both back-transform and forward-transform for the residuals function.

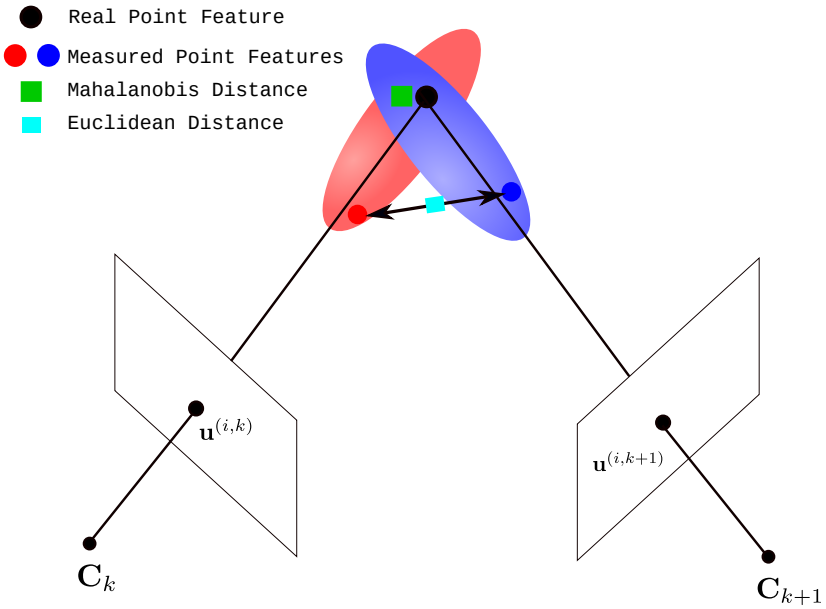


Figure 5.4.1: When matched image features are back-projected onto the camera coordinate system, they are supposed to overlap. In practice, they are located at different positions due to the noise. This effect is illustrated with red and blue points, both of which refers to the black point that is the real position of the measured point feature. In the absence of an uncertainty model of these point features, one can only minimize the Euclidean error distance, which degrades pose estimation accuracy. Conversely, one can model the point feature uncertainty with the conic ray model and minimize the Mahalanobis distance. This is the method we utilize in our VO.

With being said, we can now extend the residuals function of our optimization problem by adding forward-transform. Let's define the auxilary function in the following form:

$$\mathbf{r_f}^{(i)}(\mathbf{x}_{k,k+1}) = \mathbf{X}_{k+1}^{(i)} - f_f(\mathbf{x}_{k,k+1}, \mathbf{X}_k^{(i)}) \tag{5.14}$$

where the forward-transform function is defined as:

$$f_f(\mathbf{x}_{k,k+1}, \mathbf{X}_k^{(i)}) = \mathbf{q}_{k,k+1}^* \otimes (\mathbf{X}_k^{(i)} - \mathbf{t}_{k,k+1}) \otimes \mathbf{q}_{k,k+1} \tag{5.15}$$

Now we can reformulate our optimization problem by adding forward and back-transformation to each other along with corresponding covariance matrices:

$$\mathbf{x}^*{}_{k,k+1} = \underset{\mathbf{x}_{k,k+1}}{\operatorname{argmin}} \sum_i ||\mathbf{r_b}^{(i)}(\mathbf{x}_{k,k+1})||^2_{\mathbf{Q}^{(i)}_{\mathbf{xyz},k+1}} + ||\mathbf{r_f}^{(i)}(\mathbf{x}_{k,k+1})||^2_{\mathbf{Q}^{(i)}_{\mathbf{xyz},k}} \tag{5.16}$$

With the new auxiliary function, we can minimize the error on both forward- and back-transformation. In this way, we include noise occurring in both images

into the optimization process. This technique is taken from [Richard Hartley 2003, p.101] and is called *error in both images*. Furthermore, we expand the technique, whose original version is to minimize the Euclidean error distance, to minimize *Mahalanobis* error distance by including feature covariances (see figure 5.4.1).

Let's reformulate the residuals functions in the form of a matrix to simplify the notation for the optimization problem. Hence, a single back-transformation operation for a single feature match is a ${\mathbf{r_b}}^{(i)}$ single residual block function. The same definition applies for the forward-transformation with ${\mathbf{r_f}}^{(i)}$. Note that we only use $\frac{1}{2}$ in front of the residuals function for cosmetics reasons as it does not effect convergence of the optimization process. Here, we reorginize the residual blocks by stacking residual blocks by columns:

$$\mathbf{x}^*_{k,k+1} := \underset{\mathbf{x}_{k,k+1}}{\operatorname{argmin}} \frac{1}{2} \left\| \begin{matrix} {\mathbf{r_b}}^{(1)}(\mathbf{x}_{k,k+1}) \\ {\mathbf{r_f}}^{(1)}(\mathbf{x}_{k,k+1}) \\ \vdots \\ {\mathbf{r_b}}^{(m)}(\mathbf{x}_{k,k+1}) \\ {\mathbf{r_f}}^{(m)}(\mathbf{x}_{k,k+1}) \end{matrix} \right\|^2_{\mathbf{Q_{xyz},k},\mathbf{Q_{xyz},k+1}} \tag{5.17}$$

Above equation is the final residuals function that we are going to minimize. The LM algorithm can be used for the optimization. I refer readers for the fundamental theory behind the LM algorithm to appendices 9.2. However, we also need to extend regular least squares problem to weighted least squares problem since we multiply residual blocks with inverse covariance matrices. Let's omit $(k, k+1)$ part and generalize the problem for convenience:

$$\begin{aligned} \mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\, F(\mathbf{x}) &= \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2} ||\mathbf{r}(\mathbf{x})||^2_{\mathbf{Q}} \\ &= \frac{1}{2} \mathbf{r}(\mathbf{x})^T {\mathbf{Q_{xyz}}}^{-1} \mathbf{r}(\mathbf{x}) \\ &= \frac{1}{2} \mathbf{r}(\mathbf{x})^T \mathbf{\Omega_{xyz}} \mathbf{r}(\mathbf{x}) \end{aligned} \tag{5.18}$$

where $\mathbf{\Omega_{xyz}} = \mathbf{Q^{-1}_{xyz}}$ is the *information matrix*, repsesent a relationship between covariance matrices and weighting process. That is, the smaller covariance (smaller the uncertainty in other words) for features, the greater weight will have in the optimization. In this respect, the LM will attempt to converge to an optimal solution in which the rotation and translation information is sufficient by iteratively updating the state vector:

$$\mathbf{x}^{n+1} = \mathbf{x}^n \boxplus \Delta\mathbf{x} \tag{5.19}$$

where $\boxplus$ refers to a manifold operation. The translational part of the state vector is in Euclidean space. Thus, one can update the state vector with a regular vector addition operation. However, this addition operation does not produce good results for the rotational part. This is because the rotation is the tangent space. For the further details of the optimization on a manifold for our VO, I refer readers to appendices 9.3.

To sum up, we explained how we form the residuals function for the optimization by including back-transform and forward-transform function. We also

extended the regular least squares to weighted least squares problem by incorporating the feature covariances. In the following section, we will describe how to propagate the pose covariances.

5.5 Covariance of the Estimated Pose

Another important question one can ask in any odometry applications is that what is the uncertainty, namely covariance, of the odometry measurements? Generally, traditional VO algorithms do not provide an answer to this question since it does not take the uncertainty of the features into account. This issue introduces a big drawback, especially in sensor applications. However, we are now able to estimate a covariance matrix of the estimated pose since we use the conic ray to model feature uncertainties. With the help of each pose covariance, we can represent the possible accumulated covariances along the trajectory with 3σ ellipsoids if the relative poses are concatenated as seen in figure 5.5.1.

The fact that we use a metric uncertainty model to represent 3D point features' noise enables us to estimate a metric pose uncertainty. This property will produce more accurate pose covariance matrices. For example, if we had only detected 3D point feature matches positioned far from Kinect, the pose covariance would get larger compared to the situation where the features are placed close to Kinect. Another determinant is the number of detected feature matches. Having an insufficient number of features would increase the covariance as well. All of these will contribute to the robustness of sensor fusion applications. Now that we discuss the importance of having such a dynamic uncertainty estimation, we will provide the formulation for estimating the pose covariance matrix in the following part.

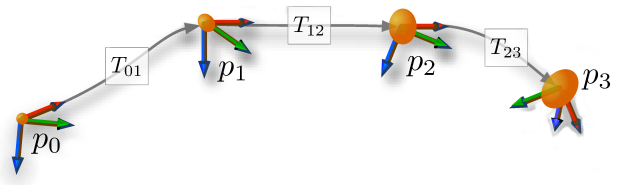


Figure 5.5.1: This is an illustration to ellipsoids for pose uncertainties that are growing with every relative pose estimations. In the absence of absolute sensors that measure the priori-known landmarks, the pose drift will grow forever.

Calculating covarinace matrix for the estimated state vector parameters is fairly straightforward if the optimization for the relative camera pose is performed as described in the previous section. We can simply utilize the *error propagation law*, which I refer readers for to appendices 9.4 for further details of the theory. In this case, we require the Jacobian matrix at the optimal solution and corresponding covariance matrices for each point feature. Then, we can propagate them to get the covariance for the estimated parameters by applying the following equation:

$$\mathbf{Q}_{\mathbf{tq},k,k+1} = (\frac{1}{2})^2 \cdot \mathbf{J'}(\mathbf{x^*}_{k,k+1})^T \mathbf{Q}_{\mathbf{xyz},k,k+1} \mathbf{J'}(\mathbf{x^*}_{k,k+1}) \tag{5.20}$$

where $\mathbf{J'}(\mathbf{x^*}_{k,k+1}) \in \mathbb{R}^{6mx6}$ is the Jacobian of extended residuals function with backward-, forward-transformation and manifold in 9.3,

$$\mathbf{Q}_{\mathbf{xyz},k,k+1} = \begin{bmatrix} \mathbf{Q}^{(1)}_{\mathbf{xyz},k} & \cdots & & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^{(1)}_{\mathbf{xyz},k+1} & \cdots & \vdots \\ \vdots & & \vdots & \\ & & \mathbf{Q}^{(m)}_{\mathbf{xyz},k} & \mathbf{0} \\ \mathbf{0} & & & \mathbf{Q}^{(m)}_{\mathbf{xyz},k+1} \end{bmatrix} \in \mathbb{R}^{6mx6m} \tag{5.21}$$

is the covariance matrix that comprised of all covarince matrices for all matched features from k^{th} and $k+1^{th}$ consecutive frames,

$$\mathbf{Q}^{(i)}_{\mathbf{xyz},k} = \mathbf{J_{bp}}^T(\mathbf{u}^{(i,k)}) \begin{bmatrix} \sigma_u & 0 & 0 \\ 0 & \sigma_v & 0 \\ 0 & 0 & (\sigma^{(i)}_z(z,\theta))^2 \end{bmatrix} \mathbf{J_{bp}}(\mathbf{u}^{(i,k)}) \in \mathbb{R}^{3x3} \tag{5.22}$$

is the covariance matrix for the i^{th} matched feature from k^{th} frame where $\mathbf{J_{bp}}^T(\mathbf{u}^{(i,k)})$ is the Jacobian matrix of back-project function (see notation 5.5).

$$\mathbf{Q}_{\mathbf{tq},k,k+1} = \begin{bmatrix} \sigma^2_{t_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{t_y} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{t_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{q_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{q_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{q_z} \end{bmatrix} \in \mathbb{R}^{6x6} \tag{5.23}$$

is the resulting covariance matrix for the estimated state vector parameters. In other words, we are now able to calculate the uncertainty of the estimated pose of the camera.

Notice $(\frac{1}{2})^2$ in front of 5.20. In the original error propagation law, this does not exist. It comes from the fact that we use both the backward- and forward-transformation in residuals function. To be more clear, remember that we represented a relative camera motion with both back and forward transformation to take errors on both k^{th} and $k+1^{th}$ consecutive images into account. This resulted in having two residuals $\mathbf{r_b}^{(i)}$ and $\mathbf{r_f}^{(i)}$ for one feature match (see notation 5.16). Therefore, we divide by 2 to average. The reason we square it is because of the covariance.

Finally, let's summarize this chapter. At first, we provided a literature background for an error-aware VO. Then, we argued that all the related work being done did not use feature uncertainty to produce metric pose covariance. With this motivation, we gave a recipe for building an error-aware VO which we call CoVO. Afterwards, we explained how we include feature and depth-related noise models into the optimization process so that we propagate a covariance matrix for the estimated relative camera pose. In the next chapter, we will evaluate the proposed CoVO algorithm with simulated and real-world data.

Chapter 6

Evaluation

Computer vision applications such as VO require many approximations and linearization techniques to cope with its dynamic nature for the sake of efficiency. In practice, many corner cases might occur, no matter how good such estimation algorithms are modeled. It is therefore critical to verify such systems, especially in the real-world environment. Considering that the proposed error-aware VO algorithm outputs two information: relative pose estimation and its covariance estimation, this chapter will mainly build around two following questions that I am going ask:

1. What is the *accuracy* of relative pose estimations?
2. How *consistent* do the algorithm estimate covariance of its pose?

In order to tackle these questions, simulation environment will be used to validate the model at first. Then, TUM RGB-D datasets will be used to test the algorithm in real-world scenarios.

6.1 Error Metrics

There are already well-established error evaluation methods in literature so I am going to follow them to better understand the characteristics of the proposed algorithm. If we want to apply these methods on our algorithm, we need to have the ground truth pose sequences $\mathbf{x}_{1:n} \in SE(3)$ for comparing with the estimated pose sequences $\mathbf{x}^*{}_{1:n} \in SE(3)$. In our calculations, we assume that both pose sequences are time-synchronized, equally sampled and have the same length n. However, it is important to note that none of these assumptions is held so in reality and we need to be aware of the error caused by these issues. We attempt to minimize these issues by linearly interpolating the ground truth.

Relative Pose Error

For evaluating the accuracy of a VO algorithm, it is better to calculate the relative pose error rather than comparing with the absolute (whole) estimated trajectory. This is also referred to Relative Pose Error (RPE). On the other hand, some VO estimate a relative pose with frame-to-frame and some VO with

keyframes. To able to compare all kinds of VO with each other, a fixed time interval Δ is chosen for measuring the local accuracy of the pose estimation. In this way, we take small local drifts into account to compare both types of VO algorithms. Let's define RPE at time instance i as follows:

$$\mathbf{E}_i := (\mathbf{x}_i^{-1}\mathbf{x}_{i+\Delta})^{-1}(\mathbf{x^*}_i^{-1}\mathbf{x^*}_{i+\Delta}) \tag{6.1}$$

One can take a mean of all $E_{1:n}$, but this hides the effect of outliers when n is large. Instead, we calculate Root Mean Error Squared (RMSE) which encodes the error with much more information since it takes mean deviation of the error into account. In other words, RMSE will show how the mean deviation of the error. To calculate RMSE, $m = n - \Delta$ number of RPE are required among n pose sequences:

$$RMSE(\mathbf{E}_{1:n}, \Delta) := \sqrt{\frac{1}{m}\sum_{i=1}^{m} ||\mathbf{E}_i||^2} \tag{6.2}$$

Then, we take a mean of all RMSE over whole trajectory:

$$RMSE(\mathbf{E}_{1:n}) := \frac{1}{n}\sum_{\Delta=1}^{n} RMSE(\mathbf{E}_{1:n}, \Delta) \tag{6.3}$$

It is important to note that we take $\Delta = 1$ when we want to know a drift per frame. This is useful when comparing simulation results with real-world experiment results for the same algorithm such as CoVO itself. Conversely, we take $\Delta = 30$ when comparing the proposed algorithm other VO such as FOVIS. This case corresponds to a drift per approximately 1 second.

Normalized Estimation Error Squared

The main focus of this thesis is to provide metric uncertainty of the relative pose estimations. We can estimate dynamic covariances thanks to propagation property of the feature covariances based on the conic ray error model. In this respect, it is critical to assess the consistency of estimated covariance values to use them safely in sensor fusion applications. One of the good metrics to evaluate consistency is to calculate Normalized Estimation Error Squared (NEES). This method measures the credibility of the provided covariance, and it helps us to decide whether the predicted uncertainty values are optimistic or pessimistic. One can calculate NEES if $\mathbf{x}_i$ real pose, $\hat{\mathbf{x}}_i$ predicted pose and $\Omega_i = \mathbf{Q}_i^{-1}$ information matrix are known at time instance i:

$$\epsilon_i = (\mathbf{x}_i - \mathbf{x^*}_i)^T \mathbf{\Omega}_i (\mathbf{x}_i - \mathbf{x^*}_i) = ||\mathbf{e}_i||^2_{\Omega_i} \tag{6.4}$$

For comparison reasons, we also take an average of NEES (ANEES) over whole trajectory:

$$d\hat{\epsilon} = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{e}_i||^2_{\Omega_i} \tag{6.5}$$

If the system is linear, has degree of freedom $d = 1$ and Gaussian noise, then the expected value $\hat{\epsilon}$ is 1.

However, in practice, this does not hold. Therefore, besides ANEES, another metric when deciding on whether the estimator is consistent or not is to look at the distribution of NEES over trajectory. It is expected that it is distributed as a chi-square χ^2_d with d degrees of freedom. For an estimator with 3 degrees of freedom, acceptance region is $\hat{\epsilon} \in [2.5, 3.5]$ when significance level α of χ^2_d is chosen 2.5% and 50 Monte Carlo runs according to [Yaakov Bar-Shalom, X. Rong Li 2001, pp.234-235]. Thus, when evaluating the consistency of the estimator in simulation, we aim to get $\hat{\epsilon_t} = [2.5, 3.5]$ for the translation and $\hat{\epsilon_q} = [2.5, 3.5]$ for the rotation since both have 3 degrees of freedom. On the other hand, when testing the estimator with real-world data, we only take upper boundary of acceptance region $\hat{\epsilon_t} \in [0, 3.5]$ and $\hat{\epsilon_q} \in [0, 3.5]$ since it is acceptable to have a conservative estimator rather than overconfident one.

6.2 Simulation Environment

Before testing the proposed algorithm, we will validate the relative pose estimation, and its covariance with the simulated data. Hence, we create a 3D simulation environment that consists of a camera pose, 3D point features and their confidence ellipsoids. Our test scenario will be comprised of the following steps:

- 500 3D point features $\mathbf{X}^{(1:500,\mathcal{C})}$ are created within camera's observable space.
- By utilizing the pinhole model (see notation 2.5), all 3D point features are projected onto the camera frame whose initial pose $\mathbf{x}_0^{\mathcal{C}}$ is known. The instrinsic matrix $\mathbf{K}$ is taken similar to TUM RGB-D FR1 dataset (see table 9.5).
- Projected image features are stored in the form of $(u_{k=0}^{(1:500)}, v_{k=0}^{(1:500)}, z_{k=0}^{(1:500)})$ sensor measurements as you would usually get it from a regular RGB-D camera.
- The camera is transformed (rotated and translated) with a known transformation information $\mathbf{T}_{0,1} = \mathbf{x}_{0,1} = [0.6, 0.6, 0.05, -0.183, -0.183, 0, 0.966]$ to its next pose $\mathbf{x}_1^{\mathcal{C}}$ (see notations 3.17 and 3.18).
- The same 3D point features are projected with respect to the new pose $\mathbf{x}_1^{\mathcal{C}}$ and stored as $(u_{k=1}^{(1:500)}, v_{k=1}^{(1:500)}, z_{k=1}^{(1:500)})$.
- In order to introduce uncertainty into the system, the Gaussian noises are added on both sensor measurements independently ($\hat{u}^{(i)} = u^{(i)} + \phi_u, \hat{v}^{(i)} = u^{(i)} + \phi_v, \hat{z}^{(i)} = z^{(i)} + \phi_z$) where $\phi \sim \mathcal{N}(\mu, \sigma)$:
  - The pixel noises are assigned to ($\mu_u = 0, \sigma_u = 8$) and ($\mu_v = 0, \sigma_v = 8$).
  - Whereas, the mean of the depth noise is $\mu_z = 0$ and standard deviation is chosen as $\sigma_z^{(i)}(z, \theta)$ with respect to feature points' distance to the camera. The lateral noise and the surface angle θ is assumed to be 0. This depth's noise model is discussed in 5.3.2.

- For every measurements, convariance matrices ($\mathbf{Q}_{\mathbf{xyz},0}^{(1:500)}$, $\mathbf{Q}_{\mathbf{xyz},1}^{(1:500)}$) are formed with the same standard deviations of the added noises, where $\mathbf{Q}_{\mathbf{xyz},0}^{(i)} = \mathbf{J_{bp}}(\mathbf{u}^{(i,k)})^T \begin{bmatrix} \sigma_u^2(=8^2) & 0 & 0 \\ 0 & \sigma_v^2(=8^2) & 0 \\ 0 & 0 & (\sigma_z^{(i)}(z,\theta))^2 \end{bmatrix} \mathbf{J_{bp}}(\mathbf{u}^{(i,k)})$.
- Perfectly matched noisy 3D point features along with their covariance matrices are given to the optimizer to estimate ${\mathbf{T}^*}_{0,1} = {\mathbf{x}^*}_{0,1}$ camera's transformation and calculate $\mathbf{Q}_{\mathbf{tq},0,1}$ covariance matrix of its estimation as discussed in 5.4.
- This whole process is repeated 1000 times.

In the following simulation figures, we show above scenario in a 3D visual environment. In this environment, we have noisy 3D point features, confidence ellipsoids whose center is around the noisy point features and the real position of the points that are somewhere inside the ellipsoids. It is worth noting that the volume of the confindence ellipsoids are determined by the conic ray model and are scaled with 3σ.

$$\mathbf{x}_0^{\mathcal{C}} = [0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.000]$$

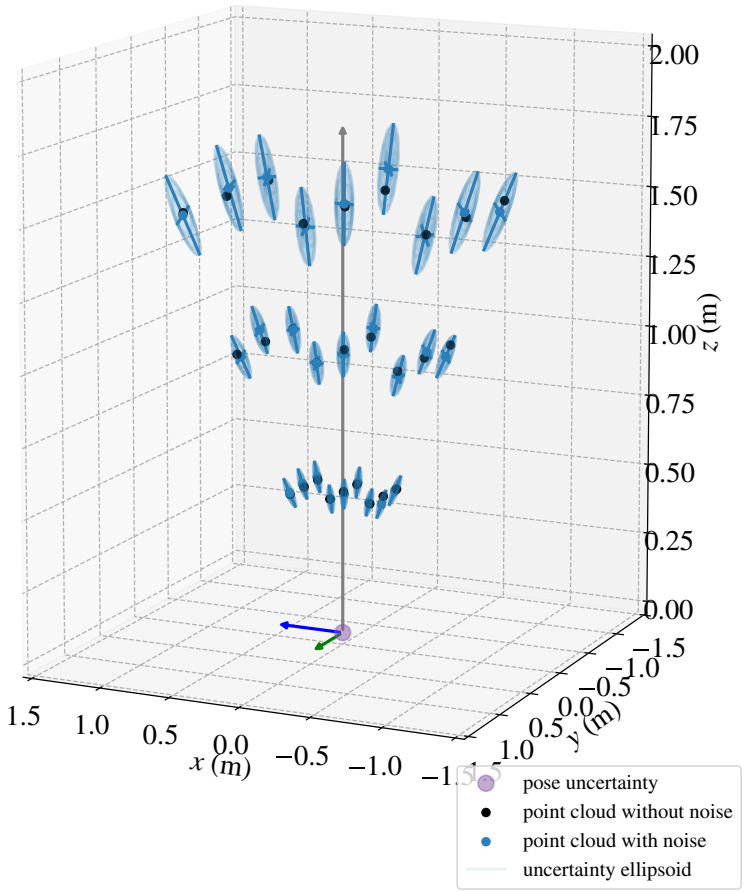


Figure 6.2.1: Simulation Environment At The Initial $\mathbf{x}_0^{\mathcal{C}}$ Pose

The figure 6.2.1 depicts the environment at its initial pose $\mathbf{x}_0^{\mathcal{C}}$. For the sake of visibility, we only draw a small subset of 3D feature points and scaled the confidence ellipses by 10. Notice how volume and orientation of the confidence ellipsoids are situated according to the conic ray model.

$$\mathbf{x}_1^{\mathcal{C}} = [0.600, 0.600, 0.050, -0.183, -0.183, 0.000, 0.966]$$

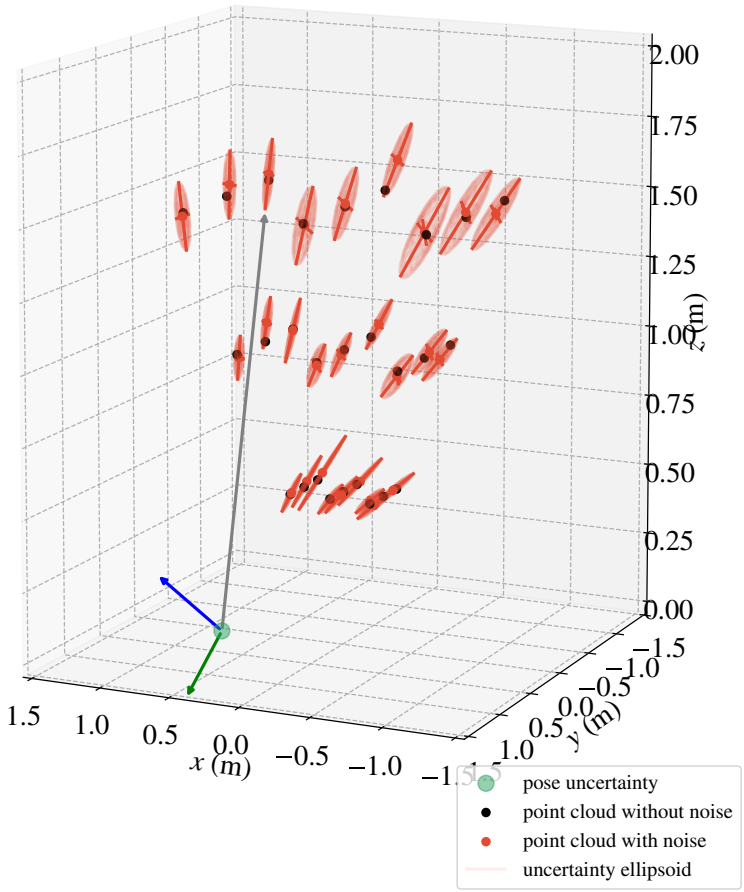


Figure 6.2.2: Simulation Environment At The Next $\mathbf{x}_1^{\mathcal{C}}$ Pose

Then, we rotate and translate the camera to its next pose $\mathbf{x}_0^{\mathcal{C}}$ as shown in figure 6.2.2. Notice how confidence ellipsoids are situated with the new camera pose accordingly.

$\mathbf{x}_0^{\mathcal{C}}$ = [0.000,0.000,0.000,0.000,0.000,0.000,1.000] and
$\mathbf{x}_1^{\mathcal{C}}$ = [0.600,0.600,0.050,-0.183,-0.183,0.000,0.966]

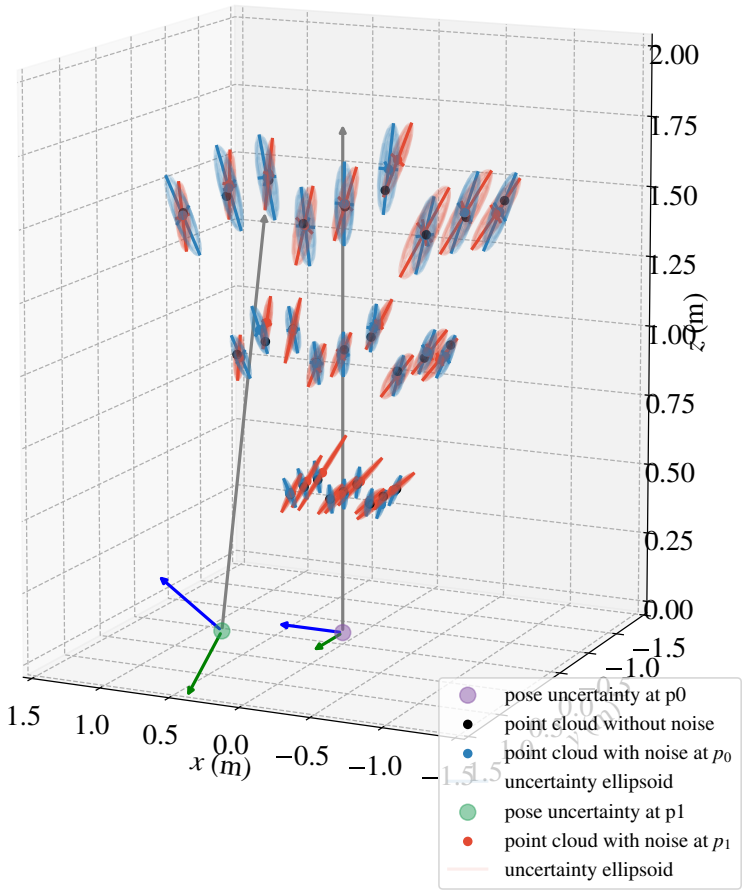


Figure 6.2.3: Simulation Environment At Both $\mathbf{x}_0^{\mathcal{C}}$ and $\mathbf{x}_1^{\mathcal{C}}$ Poses

Finally, the figure 6.2.3 shows how confidence ellipsoids from both views intersect each other and include the real feature points within the intersected space. Now that we now have everything we need, we can estimate camera transformation and its covariance and compare with their original values.

6.2.1 RPE and The Estimated Covariances

With the simulated data, we run the experiment 1000 times and added random noise on point features. Then, we estimate relative poses ${\mathbf{x}^*}_{0,1}^{(1:1000)}$ and its covariances $\mathbf{Q}_{\mathbf{tq},0,1}^{(1:1000)}$. In the end, we calculate RPE with $\Delta = 1$ so that we

validate whether the RPE lies within the 3σ bounds for each parameter of the state vector. In this case, 3σ bounds correspond to a rule of thumb evaluation metric that checks whether the RPE are bounded with 99.7% probability.

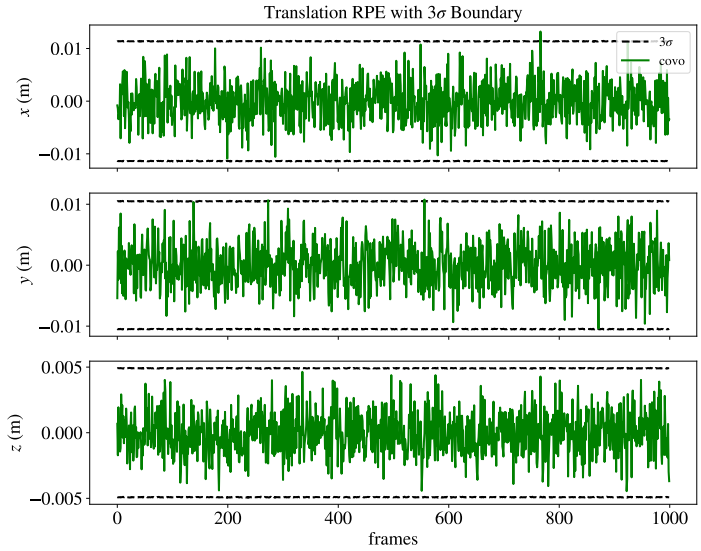


Figure 6.2.4: Translational RPE in Simulation Environment

As seen in figure 6.2.4, the translational RPE are bounded with except 2 predictions which is statistically acceptable.

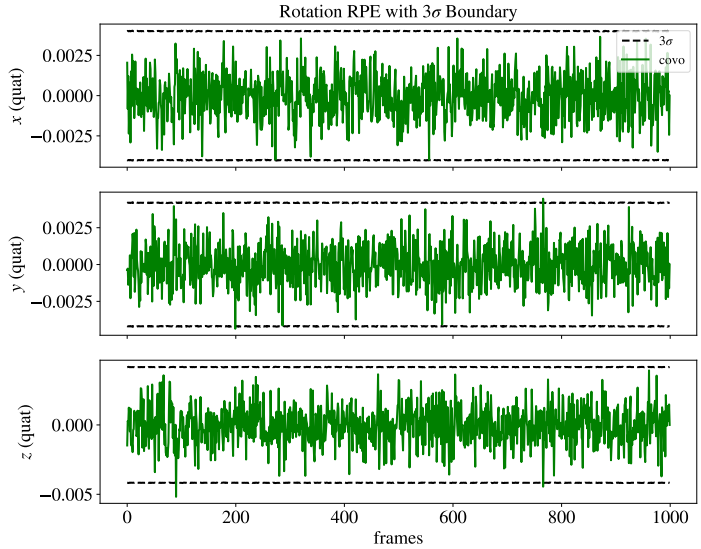
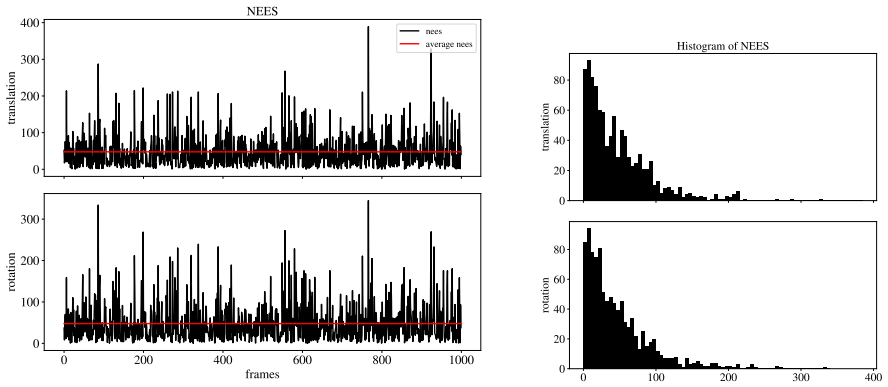


Figure 6.2.5: Rotational RPE in Simulation Environment

The same behavior is observed for the rotation as seen in figure 6.2.5. It indicates that the propagated covariances of the estimated poses based on the proposed model produce statistically acceptable covariances with simulated data if 3σ bounds are considered. What we also noticed that the majority of the RPE are beyond the 1σ bounds, which is not acceptable since 1σ corresponds to 68% probability. In other words, we expected that 68% of the RPE must be bounded with 1σ, which does not hold in our case. This is because we introduce a non-linearity with the projection function. That means the noise is not Gaussian anymore when we propagate point feature covariances $\mathbf{Q_{xyz},0,1}$ from its disparity space $\mathbf{Q_{uvd},0,1}$ with the non-linear error propagation which is the approximated with the first-order Taylor expansion $\mathbf{J_{bp}}$ (see notation 5.22). However, σ bounds evaluations itself is not enough to prove estimator's credibility. Thus, further evaluations are done with NEES.

6.2.2 Evaluation of Estimated Covariances

Even though σ bounds confirm that our algorithm provides consistent pose estimation given RPE is bounded with 3σ, we need to check whether the covariance of the estimated pose is being too optimistic or not since it fails at 1σ. Hence, we run our simulation as described at the beginning of this section again. Then, we calculate ANEES; i.e., 15.64 for translation and 15.72 for rotation. Also, the results are illustrated in figure 6.2.6.

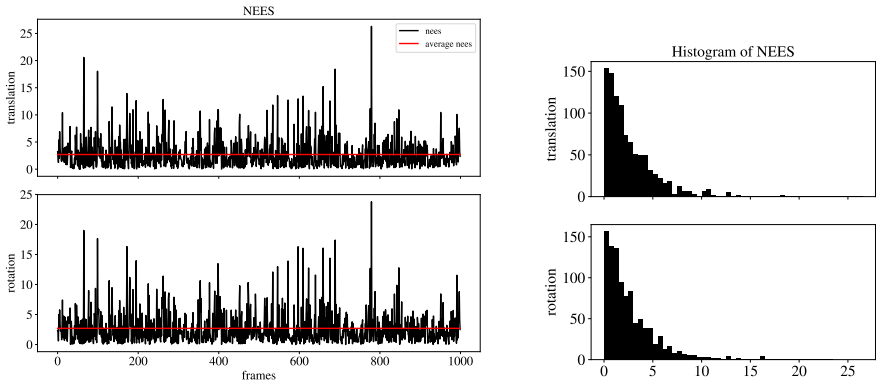


(a) NEES (b) Histogram of NEES

Figure 6.2.6: NEES For Simulated Data Without ϕ Scaling Factor

These results show that the estimator is not consistent according to the ANEES analysis. This is because the projection operation introduces non-linearity to the system as mentioned before. This linearization process does not work well with large noises, i.e., the pixel noise: $\sigma_u = 8, \sigma_v = 8$, and the depth noise: σ_z especially for the features that get larger with the range. Moreover, we run multiples test scenarios where the pixel uncertainty was kept small such as $0 < \sigma_u < 2$ and $0 < \sigma_v < 2$, and the features are located under $z < 1m$ range. In this case, we approximation works and the ANEES decreases below 3 for both translation and rotation, which is the acceptance threshold for a con-

sistent estimator. In this perspective, [Yaakov Bar-Shalom, X. Rong Li 2001, pp.395-397] suggest a couple of heuristic methods to make the estimator consistent. Among them, we apply multiplication of estimated covariance $\mathbf{Q}_{\mathbf{tq},k,k+1}$ by scaling factor ϕ after optimization.



(a) NEES (b) Histogram of NEES

Figure 6.2.7: NEES For Simulated Data With $\phi = 4^2$ Scaling Factor

For our initial test setup which extensively described in section 6.2, we find out that ANEES for translation and rotation becomes 2.7 and 2.69, respectively when taking $\phi = 4^2$ and the results are shown in figure 6.2.7. Therefore, the scaling factor can be accepted as we consider the estimator to be consistent with this setup. The reason we take this setup along with all empiric parameters in the simulation is that we want to create an environment that covers the possible worst case scenarios in the real-world dataset. We will explain why we specifically set $\sigma_u = 8$ and $\sigma_v = 8$ after analyzing the effect of pseudo inliers in the following section 6.3.1.

6.3 TUM RGB-D Dataset

TUM dataset by Sturm, Burgard, and Cremers 2012 provides an extensive benchmark capability for RGB-D based Visual SLAM or VO systems. It consists of datasets that are collected in two different indoor environments; i.e., fr1 refers to datasets collected in an office ($6 \times 6m^2$) and fr2 is collected in an open spaced garage ($10 \times 12m^2$). In our experiments, we choose a subset of datasets that are mainly suitable for VO and has sufficient texture on the images. The selected datasets are given in table 6.1. Besides, calibration parameters of RGB and depth images are validated by the time-synchronized ground truth measurements recorded by a motion capture system.

Before we use these datasets to validate our relative pose estimation and its covariance, we are going to investigate the effect of pseudo inliers that exist in our feature matches even after applying RANSAC.

Dataset Name	Duration [s]	Avg. Trans. Vel. [m/s]	Avg. Rot. Vel. [deg/s]
fr1 xyz	30	0.24	8.92
fr1 rpy	28	0.06	50.15
fr1 room	49	0.33	29.88
fr1 360	29	0.21	41.60
fr1 desk	23	0.41	23.33
fr1 desk2	25	0.43	29.31
fr2 desk	99	0.19	6.34

Table 6.1: List of Chosen TUM RGB-D Datasets

6.3.1 Discovering the Effect of Pseudo Inliers on Pixel Uncertainty

As discussed earlier in section 5.3.1, the pixel uncertainty caused by quantization operation is treated as a half pixel. Nevertheless, this error is itself not sufficient when choosing a standard deviation of pixel noise for feature covariances to be propagated through the equation 5.22. Remember that we still had outliers in feature matching even after applying RANSAC, which we call them as pseudo inliers. We proposed to treat them as inliers in the condition that they will increase the pixel uncertainty. Moreover, we expected that RANSAC would bound these pseudo inliers, which can utilize them as pixel noise. Thus, we are interested in determining the boundaries of the pseudo inliers. Since we have TUM RGB-D dataset and the ground truth measurements, we can calculate the pixel error of the pseudo inliers by aligning feature matches with transformation information from ground truth.

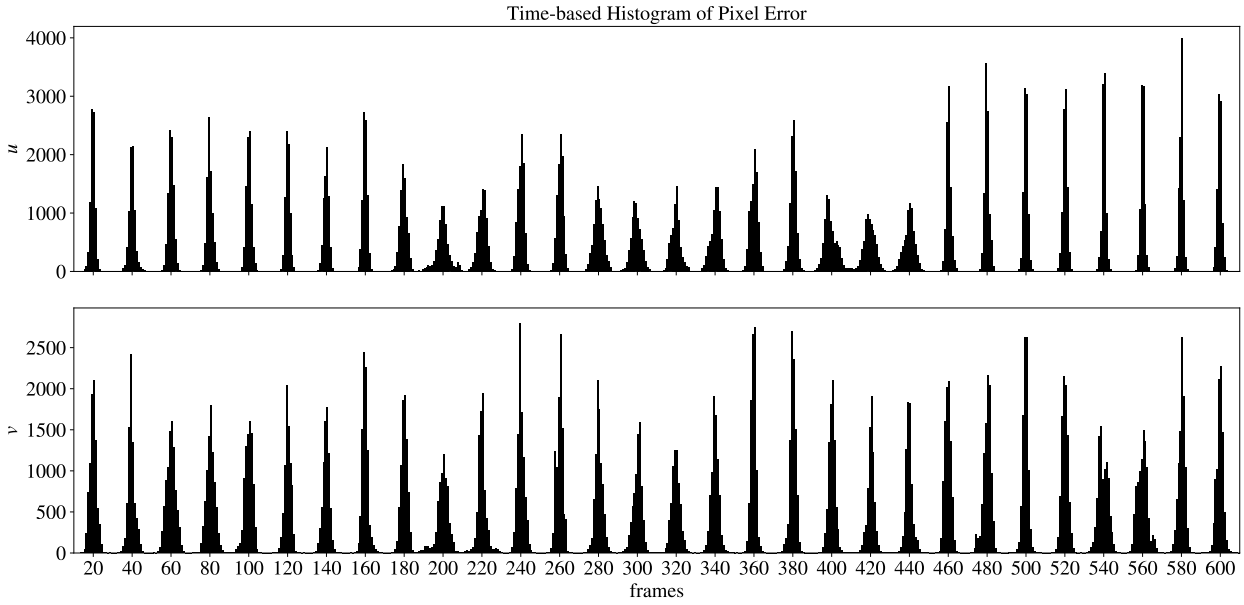


Figure 6.3.1: This illustration shows how the pixel error distances are distibuted along the trajectory. Note that we grouped the errors every 20 frames.

To be more specific, let's remind ourselves with the CoVO pipeline. To find the pixel error caused by pseudo inliers, we run our CoVO algorithm up until the optimization part at which we estimate the $\mathbf{x}^*{}_{k,k+1}(= \mathbf{T}^*{}_{k,k+1})$ transformation information. At this point, we already have feature matches that include pseudo inliers. Now, instead of estimating the transformation with the LM algorithm, we can rotate and translate $\mathbf{u}^{(1:m,k)}$ features from k^{th} frame towards $k+1^{th}$ frame and save them as $\hat{\mathbf{u}}^{(1:m,k+1)}$. In an ideal case, we expect transformed $\hat{\mathbf{u}}^{(1:m,k+1)}$ features and measured $\mathbf{u}^{(1:m,k+1)}$ features to be aligned closely such that the error distances between them are maximum a half pixel. Regardless, the error distance for some matches will be greater than a half pixel due to the pseudo inliers as we are operating on a real-world dataset. As a result, we can use this test case to identify the pixel error distance caused by the pseudo inliers. We can now apply to every consecutive frame of the whole trajectory. Thus, we illustrate the results in a time-based scheme to observe how the pixel error changes throughout the trajectory. The illustration gathered by 'fr1 xyz' dataset is given in figure 6.3.1. As seen, the pixel distance error is distributed as Gaussian and its standard deviation changes over the trajectory. To see how the standard deviation changes clearly, we draw the time-based standard deviation of the pixel errors in figure 6.3.2.

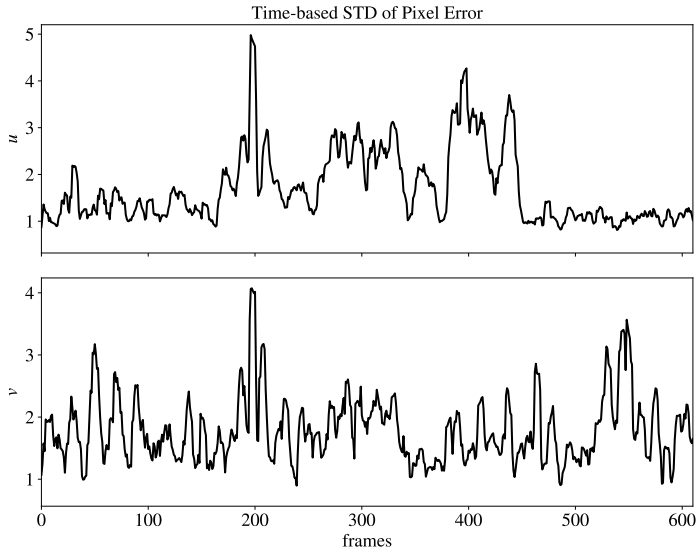
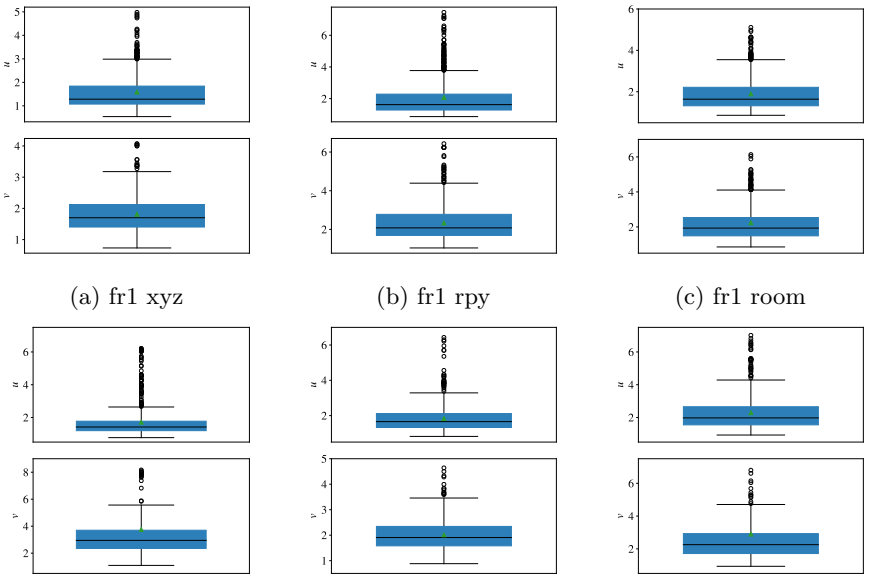


Figure 6.3.2: As seen in the previous figure, the pixel errors were distributed Gaussian but varying with time. Thus, this figure is drawn to see how the standard deviation of these distibutions changes throughout the trajectory more clearly

Because we want to cover as many corner cases as possible, we run the test over many datasets so that we can find the largest standard deviation. To see all datasets' results, we take the time-based standard deviation results which we use to draw figure 6.3.2 and illustrate with a boxplot. All the standard deviations of the pixel error from six different datasets are given in in figure 6.3.3.



(a) fr1 xyz (b) fr1 rpy (c) fr1 room

(d) fr1 360 (e) fr1 desk (f) fr1 desk2

Figure 6.3.3: Boxplots of The Standard Deviations of Pixel Errors in Different Datasets

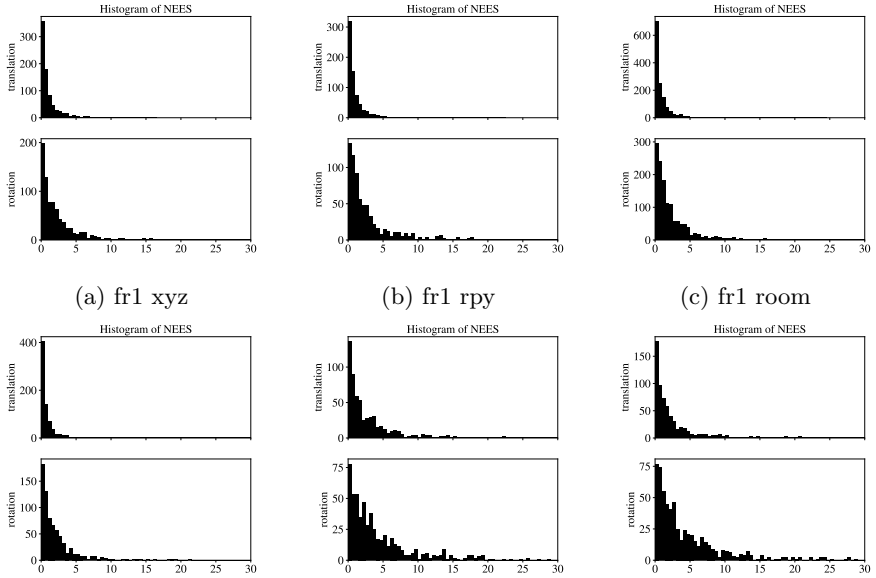
To cover the worst case scenario, we take pixel uncertainty as $\sigma_u = 8$ and $\sigma_v = 8$ from fr1 rpy and fr1 desk2 datasets since the largest error occurs in these two datasets. Finally, we form a covariance matrix $\mathbf{Q_{uvz}}^{(i,k)}$ of a point feature based on the experimental analysis that we conducted in this section.

6.3.2 Evaluation of Estimated Covariance

Similar to simulation data, we calculate NEES for the covariance of the estimated poses after running the CoVO algorithm with 6 different datasets. Note that we set standard deviation of pixel uncertainty according to the largest value $\sigma_u = 8$ and $\sigma_v = 8$ which was determined in the previous section. Also, we scale resulting covariance $\mathbf{Q_{tq}}_{,k,k+1}$ with the $\phi = 4^2$ to compensate the errors caused by the linearization after each optimization for consecutive frames. Under these parameterizations, we run the algorithm and calculate histogram of NEES and ANEES for the selected datasets. The resulting histograms are illustrated in 6.3.4 and their ANEES are given in table 6.2.

	Translation ANEES	Status	Rotation ANEES	Status
fr1 xyz	1.55	conservative	2.48	conservative
fr1 rpy	1.08	conservative	3.10	conservative
fr1 360	1.21	conservative	3.69	overconfident
fr1 desk	2.88	conservative	4.99	overconfident
fr1 desk2	2.72	conservative	5.30	overconfident
fr1 room	0.96	conservative	2.65	conservative

Table 6.2: ANEES in Different Datasets



(a) fr1 xyz (b) fr1 rpy (c) fr1 room

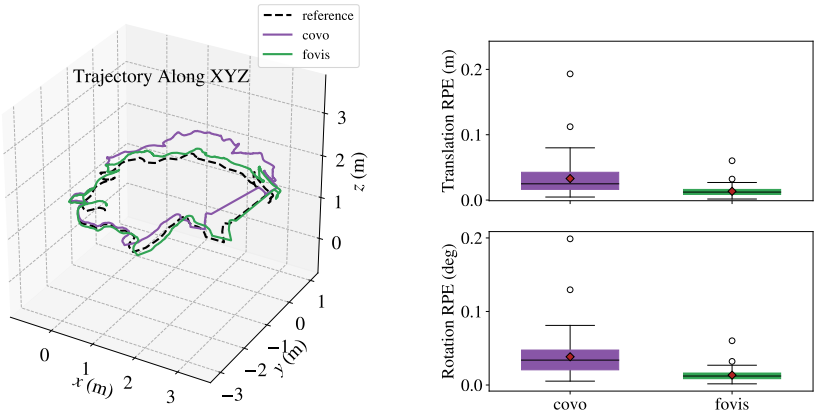
(d) fr1 360 (e) fr1 desk (f) fr1 desk2

Figure 6.3.4: Histogram of NEES in Different Datasets

We observe that covariance of the translation estimations are conversative, which is acceptable in VO systems. However, covariance of the rotation estimation result in being overconfident for 3 datasets; i.e., 'fr1 360', 'fr1 desk' and 'fr1 room'. Thus, we can say the estimator is *inconsistent* for the rotation, as opposed to the translation providing consistent covariance estimations. In literature, the accuracy and consistency of the estimation in VO algorithm are compared with the translation since it already includes the effect of the rotation (see notation 3.17). At the moment, the exact reason why covariances of rotation part are estimated poorly is unknown to me. Thus, it could be a future work where I need to analyze the quaternion algebra and manifold operation in Ceres-Solver more deeply.

6.4 Comparison to FOVIS

As for the accuracy of the estimated poses, we compare the proposed algorithm with FOVIS, which is a defacto VO application since it produces very accurate pose estimation with fast computations. To provide an illustration of the trajectory built from relative pose estimations, we draw estimations results of 'fr2 desk' which is suitable to present drift effects and it is given in figure 6.4.1 along with the boxplot representation of RPEs.



(a) The 3D Trajectories

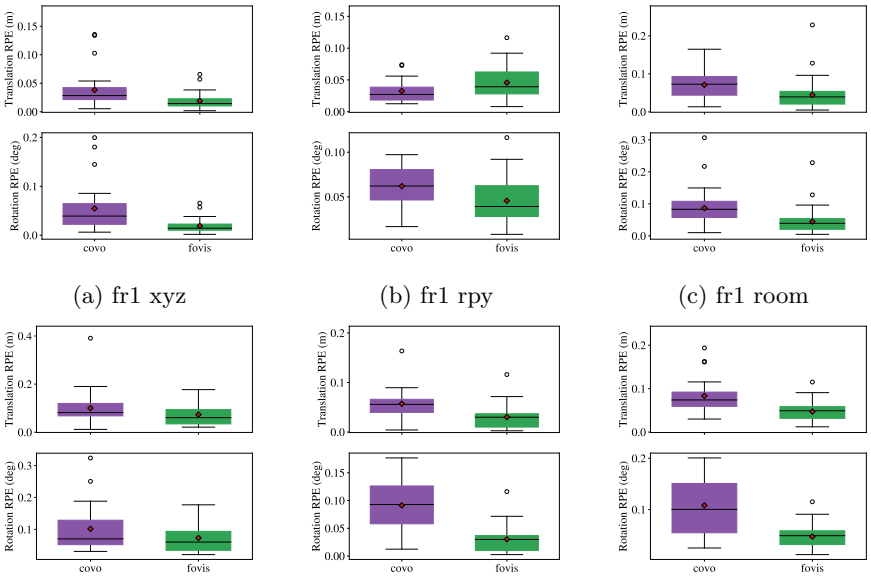
(b) Boxplots of RPE

Figure 6.4.1: FOVIS versus CoVO in The TUM FR2 Desk Dataset

We run both algorithms for the seven datasets and the resulting RMSEs of RPEs are given in table 6.3. Note that we take $\Delta = 30$ frames while calculating RPEs. Additionally, we also provide the boxplot representation of RPE for more explicit comparison in figure 6.4.2.

	FOVIS RMSE	COVO RMSE
fr1 xyz	0.0240	0.0512
fr1 rpy	0.0533	0.0368
fr1 360	0.0867	0.1239
fr1 desk	0.0406	0.0665
fr1 desk2	0.0533	0.0941
fr1 room	0.0587	0.0791
fr2 desk2	0.0156	0.0428

Table 6.3: FOVIS versus CoVO With RMSE RPE



(a) fr1 xyz (b) fr1 rpy (c) fr1 room

(d) fr1 360 (e) fr1 desk (f) fr1 desk2

Figure 6.4.2: FOVIS versus CoVO With RPE Boxplots

We can say that FOVIS is more accurate than CoVO in the context of relative pose estimations for several reasons we can name. FOVIS utilizes the keyframe scheme and different outlier rejection algorithm than RANSAC. Also, it provides a better initial guess to its optimizer after several refinement processes throughout the pipeline. Most importantly, it minimizes the 3D-to-2D correspondences which are less prone to settle for undesired local minimums in the LM algorithm.

This chapter has dealt with the evaluation of the proposed CoVO algorithm regarding accuracy and consistency. Initially, we began by describing the two main error metrics: RPE for accuracy comparisons and NEES for covariance estimator's consistency. Then, we provided a detailed step by step instruction upon which the simulation environment was built. With the use of simulated data, we identified how large noise in feature and depth measurement

contributed to covariance estimator's inconsistency. Besides this important validation, we discovered the effect of the pseudo inliers that still exist after an outlier rejection process. We proposed that the pixel error distance caused by pseudo inliers can be used as the pixel uncertainty of image features. This approach is necessary to estimate metric pose covariances, which is the primary task of this thesis. In the final section of the chapter, we compare the proposed CoVO algorithm with FOVIS in terms of accuracy.

Chapter 7

Conclusion

This thesis was undertaken to design an error-aware RGB-D Visual Odometry system and evaluate its credibility. It is achieved by identifying the errors occuring in both sensor and algorithm level and integrating them into the optimization problem.

After an introduction chapter, the first step was to lay out the foundational elements of Visual Odometry. Then, the geometrical models of a RGB-D camera sensor was given in order to be acquainted with the working principles of such sensors in Chapter 2. The pinhole and triangulation models served as essential tools when modeling error characteristics of the sensors. In Chapter 3, the typical pipeline of a feature based VO was studied. The pipeline comprised of 4 fundamental processes; that is, extracting features, matching features, rejecting outliers and pose estimation. To model uncertainy of the complete system, not only the systematic errors of the sensors, but also the errors introduced by the image processing processes are needed to be investigated.

After stating the motivation behind the thesis in Chapter 4, two main source of errors are discussed in Chapter 5; i.e., feature related errors caused by outliers and depth related errors caused by the IR sensor. All of these errors were combined in the conic ray model to represent the uncertainty of 3D feature points with covariances. Afterwards, how these uncertainties were incorporated into the optimization process and the covariance of the estimated pose was propagated through feature uncertainty were described in detail.

Chapter 6 started with presenting the simulation environment whose purpose is to validate the corectness of the algorithm. Experiments in simulation showed that linearization of projection function led to inconsistency for the covariance of the estimated poses in the case of large noise. This issue was compensated with a heuristic method that scales resulting covariance accordingly. Next, TUM RGB-D dataset was used not only for testing consistency of the estimations with the real-world data, but also for identifying the effect of pseudo inliers on pixel uncertainty. At the end of the chapther, accuracy of the proposed algorithm was compared with FOVIS.

Finally, several limitations to this work need to be acknowledged. The simulation environment was built under the assumption that the pixel and depth uncertainties are gaussian and there are no outliers in feature matches. The fact that RANSAC is non-deterministic causes to remain outliers that are greater than RANSAC's acceptance threshold. Once remaining outliers are treated as

pseudo inliers, pixel uncertainty grows, which then brings out issues in the linearizations. It is also obvious that setting the pixel uncertainty by the largest standard deviation value after running exhaustive tests to determine how pseudo inliers' error are distributed throughout all trajectories will not cover every corner cases. Thus, this problem requires more sophisticated approaches if more adaptive covariance estimation is desired. Additionally, in contrast to our simulation environment, the analysis being done for estimator's consistency with NEES is usually performed in Monte Carlo simulation to introduce greater uncertainty to the system.

TODO: open questions and future work

Chapter 8

Bibliography

Bibliography

Agarwal, Sameer, Keir Mierle, et al. (2019). *Ceres Solver*. http://ceres-solver.org.

Aqel, Mohammad O.A. et al. (2016). "Review of visual odometry: types, approaches, challenges, and applications". In: *SpringerPlus* 5.1. ISSN: 21931801. DOI: 10.1186/s40064-016-3573-7.

Arras, Ko and Ko Arras (1998). "An Introduction to Error Propagation: Derivation, Meaning, and Examples of Equation cy= fx cx fx". In: *Lausanne: Swiss Federal Institute of Technology Lausanne (EPFL)* September, pp. 98–01. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:An+Introduction+To+Error+Propagation:+Derivation,+Meaning+and+Examples+of+Equation%7B%5C#%7D0.

Bay, Herbert et al. (2008). "Speeded-Up Robust Features (SURF)". In: *Computer Vision and Image Understanding* 110.3, pp. 346–359. ISSN: 10773142. DOI: 10.1016/j.cviu.2007.09.014. arXiv: arXiv:1011.1669v3.

Belter, Dominik, Michał Nowicki, and Piotr Skrzypczyński (2018). "Modeling spatial uncertainty of point features in feature-based RGB-D SLAM". In: *Machine Vision and Applications* 29.5, pp. 827–844. ISSN: 14321769. DOI: 10.1007/s00138-018-0936-9.

Besl, Paul J. and Neil D. McKay (1992). "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/34.121791.

C., Daniel Herrera, Juho Kannala, and Janne Heikkila (2016). "Joint depth and color camera calibration with distortion". In: pp. 144–147. DOI: 10.1109/ICVRV.2015.21.

Choo, Benjamin et al. (2014). "Statistical Analysis-Based Error Models for the Microsoft". In: *sensors*, pp. 17430–17450. DOI: 10.3390/s140917430.

Di, Kaichang et al. (2016). "RGB-D SLAM based on extended bundle adjustment with 2D and 3D information". In: *Sensors (Switzerland)* 16.8. ISSN: 14248220. DOI: 10.3390/s16081285.

Endres, Felix et al. (2014). "3D Mapping with an RGB-D Camera". In: *IEEE TRANSACTIONS ON ROBOTICS* 30.1, pp. 1–11.

Fang, Zheng and Yu Zhang (2015). "Experimental evaluation of RGB-D visual odometry methods". In: *International Journal of Advanced Robotic Systems* 12, pp. 1–16. ISSN: 17298814. DOI: 10.5772/59991.

Fischler, Martin a and Robert C Bolles (1981). "Random Sample Paradigm for Model Consensus: A Apphcatlons to Image Fitting with Analysis and Automated Cartography". In: *Communications of the ACM* 24.6, pp. 381–395. ISSN: 00010782. DOI: 10.1145/358669.358692. arXiv: 3629719.

Fraundorfer, Friedrich and Davide Scaramuzza (2011). "Visual odometry: Part I: The First 30 Years and Fundamentals". In: *IEEE Robotics and Automation Magazine* 19.2, pp. 78–90. ISSN: 10709932. DOI: 10.1109/MRA.2012.2182810.

Frese, Udo (2010). "Interview: Is SLAM Solved?" In: *KI - Künstliche Intelligenz* 24.3, pp. 255–257. ISSN: 0933-1875. DOI: 10.1007/s13218-010-0047-x. URL: http://dx.doi.org/10.1007/s13218-010-0047-x.

Geiger, Andreas, Julius Ziegler, and Christoph Stiller (2011). "StereoScan: Dense 3d Reconstruction in Real-time". In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1–9. URL: http://ieeexplore.ieee.org/abstract/document/5940405/.

Harris, C. and M. Stephens (1988). "A Combined Corner and Edge Detector". In: *Procedings of the Alvey Vision Conference 1988*, pp. 23.1–23.6. ISSN: 09639292. DOI: 10.5244/C.2.23. arXiv: 0804.1469. URL: http://www.bmva.org/bmvc/1988/avc-88-023.html.

Huang, Albert S et al. (2011). "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera". In: *International Symposium of Robotics Research*.

Itseez (2019). *Open Source Computer Vision Library*. https://github.com/itseez/opencv.

Karan, Branko (2015). "Calibration of kinect-type RGB-D sensors for robotic applications". In: *FME Transactions* 43.1, pp. 47–54. ISSN: 14512092. DOI: 10.5937/fmet1501047K.

Kerl, Christian and Daniel Cremers (2013). "Robust Odometry Estimation for RGB-D Cameras". In: *IEEE International Conference on Robotics and Automation*, pp. 3748–3754.

Khoshelham, Kourosh and Sander Oude Elberink (2012). "Accuracy and resolution of kinect depth data for indoor mapping applications". In: *Sensors* 12.2, pp. 1437–1454. ISSN: 14248220. DOI: 10.3390/s120201437. arXiv: arXiv:1505.0193.

Kinect, Time-of-flight et al. (2015). "Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect". In: arXiv: arXiv:1505.05459v1.

Klette, Reinhard (2014). *Concise Computer Vision - An Introduction into Theory and Algorithms*. ISBN: 9781447163190. URL: http://www.springer.com/us/book/9781447163190.

Konolige, Kurt and Motilal Agrawal (2008). "FrameSLAM : From Bundle Adjustment to Real-Time Visual Mapping". In: *IEEE TRANSACTIONS ON ROBOTICS* 24.5, pp. 1066–1077.

Leo, Giuseppe Di, Consolatina Liguori, and Alfredo Paolillo (2011). "Covariance Propagation for the Uncertainty Estimation in Stereo Vision". In: *IEEE Transactions on Instrumentation and Measurement* 60.5, pp. 1664–1673. DOI: 10.1109/TIM.2011.2113070.

Lowe, David G (2004). "Distinctive Image Features from Scale-Invariant Keypoints David". In: pp. 1–28. ISSN: 0920-5691. DOI: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94. arXiv: 0112017 [cs].

M., Calonder et al. (2010). "BRIEF: Binary robust independent elementary features". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISSN: 978-3-642-15560-4. DOI: 10.1007/978-3-642-15561-1_56. arXiv: arXiv:1407.5736v1.

Mallick, Tanwi, Partha Pratim Das, and Arun Kumar Majumdar (2014). "Characterizations of noise in Kinect depth images: A review". In: *IEEE Sensors Journal* 14.6, pp. 1731–1740. ISSN: 1530437X. DOI: 10.1109/JSEN.2014.2309987.

Matthies, L and S A Shafer (1987). "Error Modeling in Stereo Vision". In: *IEEE J. Robotics and Automation* 3.3, pp. 239–248.

Miura, Jun and Yoshiaki Shirai (1993). "An Uncertainty Model of Stereo Vision and its Application to Vision-Motion Planning of Robot". In: *Proc. 13th Int. Joint Conf. on Artificial Intelligence* August, pp. 1618–1623.

Moravec, Hans P (1980). "Obstacle Avoidance and Navigatio n in the Real World by a Seeing Robot Rover". PhD thesis.

Nguyen, Chuong V., Shahram Izadi, and David Lovell (2012). "Modeling kinect sensor noise for improved 3D reconstruction and tracking". In: *Proceedings - 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2012*, pp. 524–530. ISSN: 978-1-4673-4470-8. DOI: 10.1109/3DIMPVT.2012.84. arXiv: arXiv:1505.0193.

Nist, David and James Bergen (2004). "Visual Odometry". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Olson, Clark F et al. (2003). "Rover navigation using stereo ego-motion". In: *Robotics and* 43, pp. 215–229. DOI: 10.1016/S0921-8890(03)00004-6.

Park, Jae-Han et al. (2012). "Spatial Uncertainty Model for Visual Features Using a Kinect Sensor". In: *Sensors* 12.12, pp. 8640–8662. ISSN: 1424-8220. DOI: 10.3390/s120708640. URL: http://www.mdpi.com/1424-8220/12/7/8640/.

Richard Hartley, Andrew Zisserman (2003). *Multiple View Geometry*. Vol. 53. 9, pp. 1689–1699. ISBN: 9788578110796. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

Rosten, Edward and Tom Drummond (2006). "Machine learning for high-speed corner detection". In: pp. 1–14. URL: papers://9cdf6d24-b953-4ed9-8d83-e25ff0393b0d/Paper/p3567.

Rublee, Ethan et al. (2011). "ORB: An efficient alternative to SIFT or SURF". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571. ISSN: 1550-5499. DOI: 10.1109/ICCV.2011.6126544.

Rusinkiewicz, Szymon and Marc Levoy (2001). "Efficient Variants of the ICP Algorithm". In: *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*.

Scaramuzza, Davide and Friedrich Fraundorfer (2011). "Visual Odometry Part II: Matching,Robustness, Optimization, andApplications". In: *IEEE Robotics & Automation Magazine* 18.4, pp. 80–92. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943233.

Smisek, Jan, Michal Jancosek, and Tomas Pajdla (2011). "3D with Kinect". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1154–1160. ISSN: 978-1-4673-0063-6. DOI: 10.1109/ICCVW.2011.6130380. arXiv: arXiv:1011.1669v3.

Sol, Joan (2016). "Course on SLAM". In: URL: http://www.iri.upc.edu/people/jsola/JoanSola/objectes/toolbox/courseSLAM.pdf.

Solà, Joan (2007). "Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: A Geometric and Probabilistic Approach". In: p. 199.

Sturm, Jürgen, Wolfram Burgard, and Daniel Cremers (2012). "Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark". In: *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS IROS*. DOI: 10.1.1.364.5940.

Wasenmüller, Oliver and Didier Stricker (2017). "Comparison of kinect v1 and v2 depth images in terms of accuracy and precision". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10117 LNCS, pp. 34–45. ISSN: 16113349. DOI: 10.1007/978-3-319-54427-4_3. arXiv: 1603.06937.

Yaakov Bar-Shalom, X. Rong Li, Thiagalingam Kirubarajan (2001). *Estimation with applications to tracking and navigation.*

Zhang, Zhengyou (2000). "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11, pp. 1330–1334. ISSN: 01628828. DOI: 10.1109/34.888718. arXiv: arXiv:1011.1669v3.

Zhou, Yi, Laurent Kneip, and Hongdong Li (2017). "Semi-Dense Visual Odometry for RGB-D Cameras Using Approximate Nearest Neighbour Fields". In: arXiv: 1702.02512. URL: http://arxiv.org/abs/1702.02512.

Chapter 9

Appendices

9.1 Rigid-Body Transformations

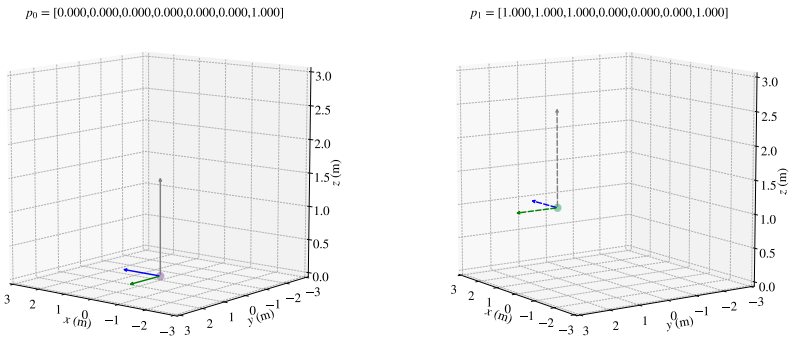
Rigid-body refers to objects made of solid materials so deformation is neglected. Thus, we idealize it by assuming that any given two points of rigid-body remains constant in time regardless of external forces applied on it. In $\mathbb{R}^3$ space, a rigid-body has 6 degree of freedom; i.e, 3 for the position (x, y, z) and 3 for the orientation (α, β, γ). For a rigid-body, the motion is composed of a rotation around an axis and a translation along an axis.

Let $p_0 = [\mathbf{t}, \mathbf{q}]^T = [t_x, t_y, t_z, q_x, q_y, q_z, q_w]^T$ be the initial pose of the frame that is fixed to the rigid-body where $\mathbf{t}$ represents the position and $\mathbf{q}$ rotation.

Translation

One can translate the rigid-body to another position with a simple vector addition operation:

$$\begin{aligned} \mathbf{t}_1 &= \mathbf{t}_0 + \mathbf{t}_{01} \\ \begin{bmatrix} t_x^1 \\ t_y^1 \\ t_z^1 \end{bmatrix} &= \begin{bmatrix} t_x^0 \\ t_y^0 \\ t_z^0 \end{bmatrix} + \begin{bmatrix} t_x^{01} \\ t_y^{01} \\ t_z^{01} \end{bmatrix} \end{aligned} \tag{9.1}$$



(a) p0 (b) p1

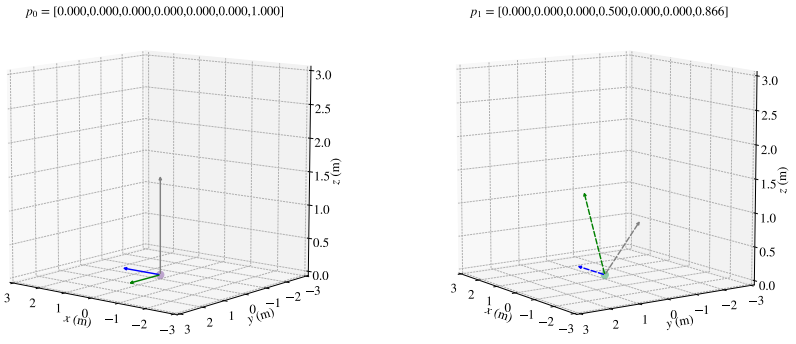
Figure 9.1.1

Rotation with Quaternions

One can rotate the rigid-body to another orientation with the quaternion product:

$$\begin{aligned}
\mathbf{q}_1 &= \mathbf{q}_{01} \otimes \mathbf{q}_0 \\
\mathbf{q}_1 &= [\mathbf{q}_{01}]_L \mathbf{q}_0 \\
\begin{bmatrix} q_w^{01} q_w^0 - q_x^{01} q_x^0 - q_y^{01} q_y^0 - q_z^{01} q_z^0 \\ q_w^{01} q_x^0 + q_x^{01} q_w^0 + q_y^{01} q_z^0 - q_z^{01} q_y^0 \\ q_w^{01} q_y^0 - q_x^{01} q_z^0 + q_y^{01} q_w^0 + q_z^{01} q_x^0 \\ q_w^{01} q_z^0 + q_x^{01} q_y^0 - q_y^{01} q_x^0 + q_z^{01} q_w^0 \end{bmatrix} &= \begin{bmatrix} q_w^{01} - q_x^{01} - q_y^{01} - q_z^{01} \\ q_w^{01} + q_x^{01} + q_y^{01} - q_z^{01} \\ q_w^{01} - q_x^{01} + q_y^{01} + q_z^{01} \\ q_w^{01} + q_x^{01} - q_y^{01} + q_z^{01} \end{bmatrix} \begin{bmatrix} q_w^0 \\ q_x^0 \\ q_y^0 \\ q_z^0 \end{bmatrix}
\end{aligned} \tag{9.2}$$

where $[\mathbf{q}_{01}]_L$ is the left quaternion product in matrix form.



(a) p0 (b) p1

Figure 9.1.2

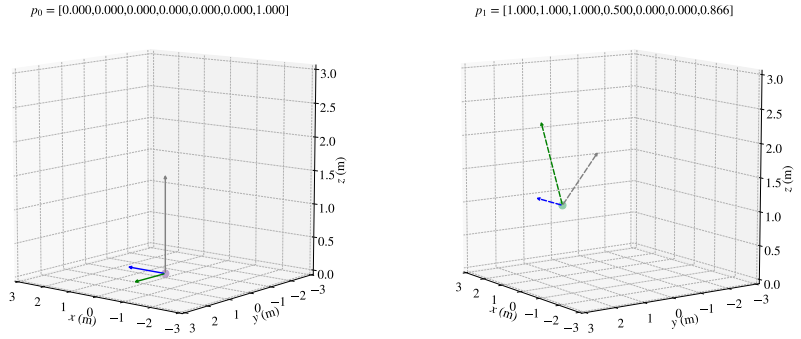
Transformation

The most useful motion is the roto-translation motion, also called transformation, which rotates the rigid-body and then tanslates it:

$$\mathbf{t}_1 = \mathbf{q}_{01} \otimes \mathbf{t'}_0 \otimes \mathbf{q}^*_{01} + \mathbf{t'}_{01} \tag{9.3}$$

$$\mathbf{q}_1 = \mathbf{q}_{01} \otimes \mathbf{q}_0 \tag{9.4}$$

where $\mathbf{t}' = [0, t_x, t_y, t_z]^T$ is the quaternion version of a vector $\mathbf{t}$ with zero scalar part.



(a) p0 (b) p1

Figure 9.1.3

9.2 Least Squares

Throughout this thesis, least squares method empowered many different components of out VO system, such as camera calibration, RANSAC and most importantly motion estimation, Therefore, we will discuss underlying principles of least squares method in this section.

Ultimately, error minimization is an operation which wish to get the maximum likelihood of the function. To do so, we search the most likely state configuration as close as possible to its exact and ideal solution. In the case of any optimization problems, the goal is to find interesting points, such as local/global maximum or local/global minimum, on the *objective function*. However, the exact model $F(\mathbf{x})$ of a system mostly unknown due to the high-degree for non-linearty or lack of knowledge. Additionally, one needs to model the noise characteristics of measurements in real-world. This noise modeling also requires an approximation. Thus, one can only (hopefully) find a good enough solution by iteratively searching. One way to solve effectively such problems is to generate a quadratic model of the objective function around initial guess $\mathbf{x}^0$ and iterate through the function using the *Newton's methods* or its variations. For example, an optimal solution (or an interest point) figure 9.2.1 is at the local minimum of the function that is highlighted as a red point cloud.

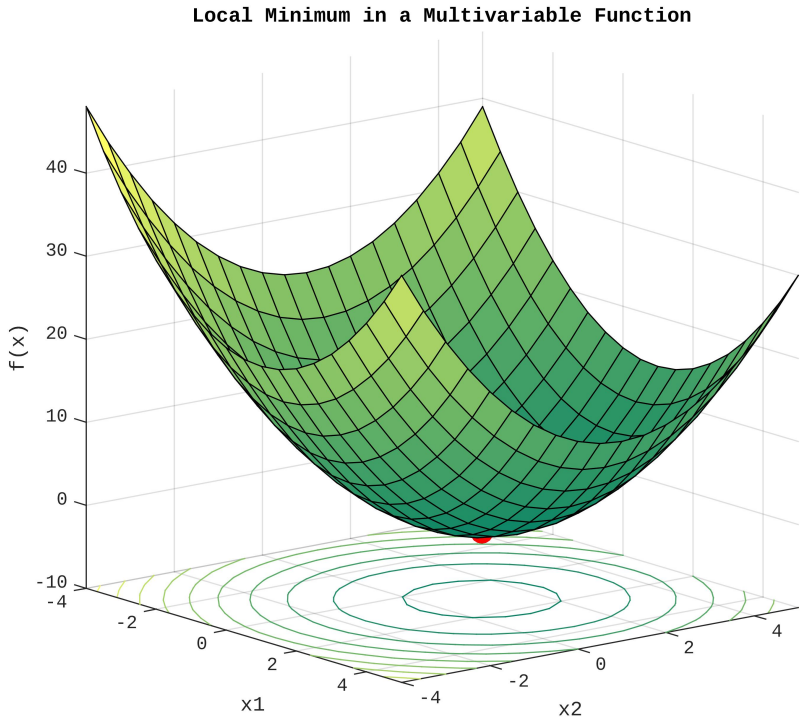


Figure 9.2.1: Local Minimum at a Quadratic Function

To elaborate the problem, we provide a regression example which can be infact solved with linear least squares techniques but it can serve as a simple toy example throughout our explainations.

Suppose that we have a model function $g(\mathbf{x}; a)$. However, we don't know what the $\mathbf{x} = (x_1, x_2)$ coefficients (so-called *optimization parameters*) are and we can only plug a, which is the *independent* variable, into the *model* S to see how the output of the model changes given the independent variable.

$$\eta = g(x; \xi).$$

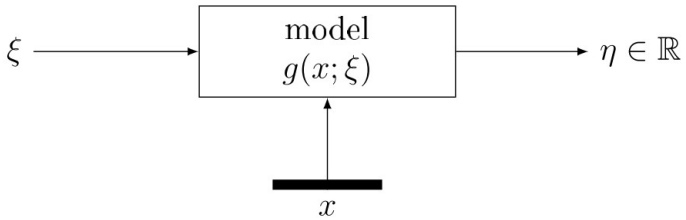


Figure 9.2.2: Least Square Model

$$\xi = a \text{ (independent variable)} \in \mathbb{R} \tag{9.5}$$
$$\eta = S \text{ (dependent variable)} \in \mathbb{R} \tag{9.6}$$

$$S = g(\mathbf{x}; a) := x_1 + x_2 a \quad \text{where} \quad \mathbf{x} \in \mathbb{R}^2 \tag{9.7}$$

To see this effect, we draw a graph that is shown in figure 9.2.3.

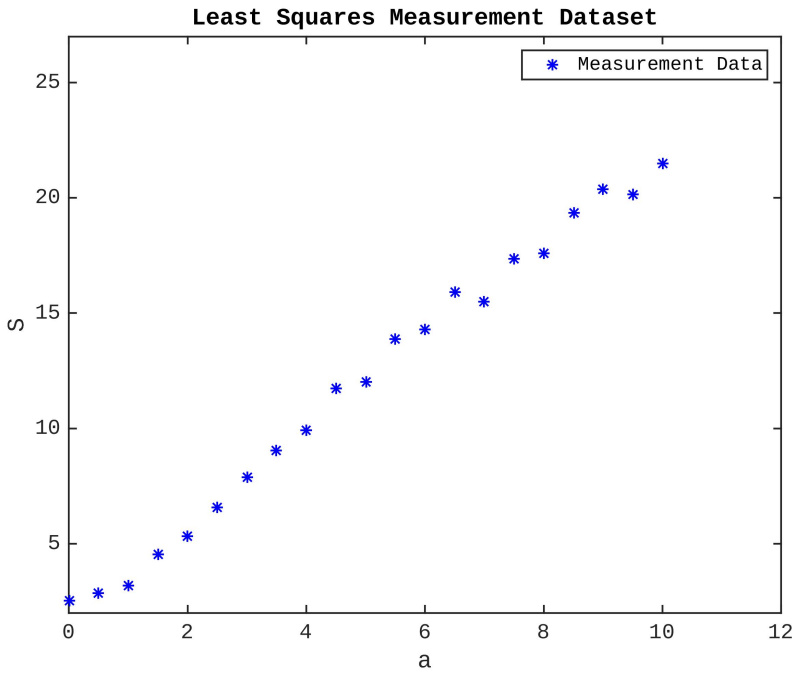


Figure 9.2.3: Least Squares Measurements

Our goal is now to find a function, which will fit this dataset. This is a typical least squares curve fitting problem. In this problem, we construct a *residuals function* $r_i(\mathbf{x})$ by providing error values between model estimation $g(x; a_i)$ and dependent variable S_i. In this case, the dependent variable represent the real world measurements and the residuals function represents the error between the estimated value and measurement value.

$$r_i(\mathbf{x}) := g(\mathbf{x}; a_i) - S_i \qquad \text{for } i = 1, \ldots, m. \tag{9.8}$$

The residuals function is usually squared to magnify larger error effect:

$$\begin{aligned} F(\mathbf{x}) &= \sum_{i=1}^{m} |r_i(\mathbf{x})|^2 = \sum_{i=1}^{m} |g(\mathbf{x}; a_i) - S_i|^2 = \sum_{i=1}^{m} (g(\mathbf{x}; a_i) - S_i)^2 \\ &= \left\| \begin{pmatrix} g(\mathbf{x}; a_1) - S_1 \\ \vdots \\ g(\mathbf{x}; a_m) - S_m \end{pmatrix} \right\|_2^2 \end{aligned} \tag{9.9}$$

Now, one can use the sum of squared error function to calculate the most likely configuration that can minimize the errors.

$$\mathbf{X}^* = \underset{x}{\operatorname{argmin}} F(\mathbf{x}) = \sum_{i=1}^{m} (g(\mathbf{x}; a_i) - S_i)^2, \quad \mathbf{x} \in \mathbb{R}^2 \tag{9.10}$$

At this point, the objective function is ready to be handed over to to any gradient decent based least squares solver. The method will try to find the *optimal* solution $\mathbf{X}^*$ by minimizing the objective function.

$$\text{Minimize} \quad \sum_{i=1}^{m} (g(\mathbf{x}; a_i) - S_i)^2, \quad \mathbf{x} \in \mathbb{R}^2 \tag{9.11}$$

To help our understanding, the objective function is drawn in figure 9.2.4. As we can see, the based on the given $\mathbf{x}$ values, the objective function $F(\mathbf{x})$ behave as follows:

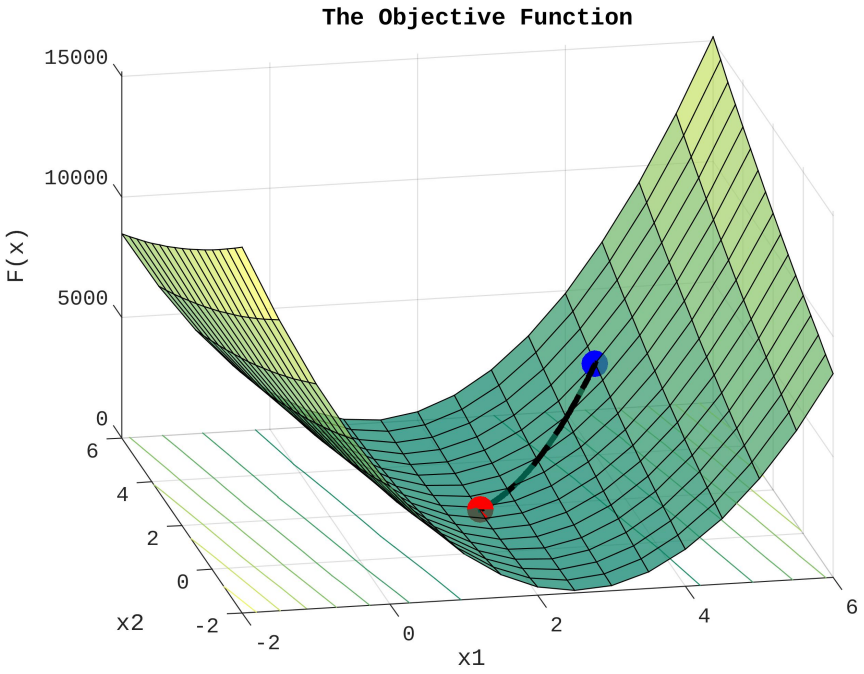


Figure 9.2.4: Local Minimum at Sum of Squared Error Function

The initial guess for the optimization parameters at $x = (4, 4)$, which is highlighted as a blue color point cloud and the local minimum point, which is the interest point of ours, is highlighted with the red cloud. What the gradient descent method essentially does is to travel from the initial guess point to the nearest local minimum on the objective function. As we can see that the descent is successfully performed and the optimization operation results at $\mathbf{x} = (1.9, 2.1)$.

If we now place the optimization variables into our model function, we get the $g(\mathbf{x}; a) = 1.9 + 2.1a$ and a fitted line based on the given dataset in figure 9.2.5.

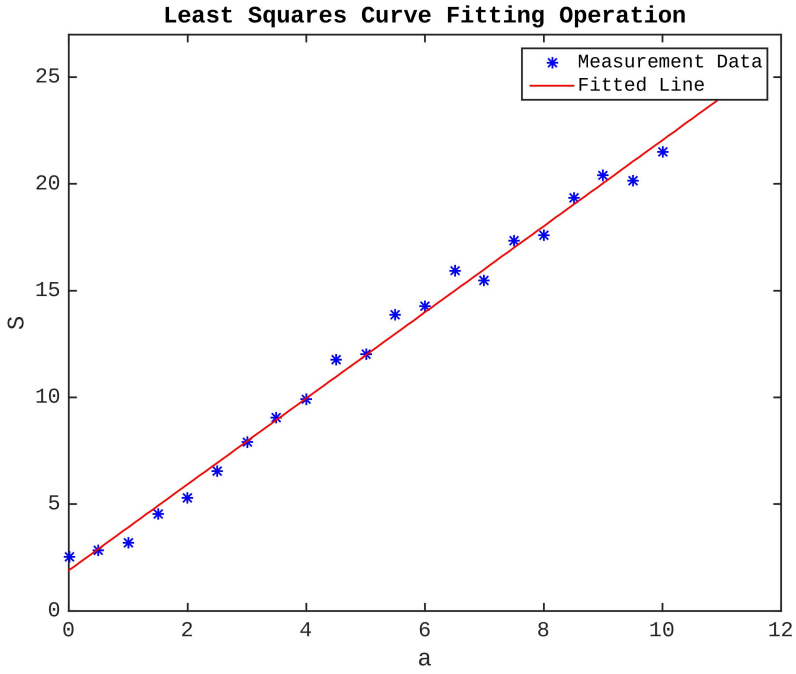


Figure 9.2.5: Least Squares Curve Fitting Operation

Now that we have an intiution how least squares problems are solved by gradient descent methods, let's further discuss one of them.

9.2.1 The Newton's Method

In order to understand non-linear least square algorithms, we first need to look at how the Newton's method works. It provides a guideline for finding roots of a function by taking differentiations of the function. The interesting phenomena is that the roots of function corresponds to the interest points of the function such as maximum, mimimum or saddle points. In least squares problem, we are only interested in minimum points since we want to minimize the error. Keeping this in mind, let's consider a non-linear objective function $F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^{\mathbf{T}}\mathbf{f}(\mathbf{x}) = \frac{1}{2}\sum_i f_i(\mathbf{x})^2$ where it has $\mathbf{x} = [x_1, \ldots, x_n] \in R^n$ multiple unknown variables. We also assume that the function $F(\mathbf{x})$ is differentiable with respect to each x_i unknown variable and its first-order derivatives has the following form:

$$\nabla F(\mathbf{x}) = \mathbf{J}_F = \mathbf{J}_{\mathbf{f}}^{\mathbf{T}}\mathbf{f} \tag{9.12}$$

where $\mathbf{J}_F$ is the *Jacobian* matrix that has a special matrix:

$$\mathbf{J}_F = \left[\frac{\partial}{\partial x_j} f_i(\mathbf{x})\right]_{ij} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{ixj} \quad (9.13)$$

And the second-order derivatives has another special matrix for called *Hessian*

$$\nabla^2 F(x) = \mathbf{H}_F = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f_1(\mathbf{x}) & \frac{\partial^2}{\partial x_1 x_2} f_1(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_1 x_n} f_1(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 x_1} f_2(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} f_2(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_2 x_n} f_2(\mathbf{x}) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2}{\partial x_n x_1} f_m(\mathbf{x}) & \frac{\partial^2}{\partial x_n x_2} f_m(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_n^2} f_m(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{mxn} \quad (9.14)$$

Having first- and second-order derivatives helps us to form a so-called *Newton* step:

$$\Delta \mathbf{x}^n = -\frac{\nabla F(\mathbf{x}^n)}{\nabla^2 F(\mathbf{x}^n)} = -\mathbf{H}_F^{-1} \mathbf{J}_F \quad (9.15)$$

If we calculate the Netwon step, we know that $F(\mathbf{x})$ will decrease in the direction of its negative derivative if the step is added: Then, one can claim that if we iteratively travel through the function in the direction of its negative derivative, we would eventually reach to a nearest local minima with respect to the starting point.

We can briefly describe the algorithm if assuming that we have an educated initial guess $\mathbf{x}^0$ from which we start to search for a local minimum, we can describe the Newton algorithm as follows:

1. take the first- and second-order derivatives of $F(\mathbf{x})$ at the current $\mathbf{x}^n$
2. iterate $\mathbf{x}^{n+1} := \mathbf{x}^n - \Delta \mathbf{x}^n$ until it converges

As we see, the Newton's method is a simple but an effective algorithm if we can calculate the first- and second-order derivatives accurately. However, it is either computationally expensive to calculate Hessian or is unknown beforehand. Therefore, there are several variations of Newton's method, one of which will be discussed in next section.

9.2.2 Levenberg-Marquardt

Levenberg-Marquardt (LM) is one of most well-known algoritm to solve least squares problem. It is the modified version of the Netwon's method. In this section, we describe the idea behind the LM algorithm. Assume that we have a $\mathbf{r}(\mathbf{x}^n)$ residuals function which we wish to model:

$$\mathbf{r}(\mathbf{x}^n) = \begin{pmatrix} r_1(\mathbf{x}^n) \\ \vdots \\ r_m(\mathbf{x}^n) \end{pmatrix} \in \mathbb{R}^m \quad (9.16)$$

To find a local maximum/minimum of the residuals function, we need to determine the first derivative and it $\mathbf{r}'(\mathbf{x}^n)$ appears again in Jacobian form:

$$\mathbf{J} = \frac{\partial \mathbf{r}(\mathbf{x}^n)}{\partial \mathbf{x}^n}\bigg|_{\mathbf{x}^n} = \begin{bmatrix} | & & | \\ \frac{\partial}{\partial x_1}\mathbf{r}(\mathbf{x}^n) & \dots & \frac{\partial}{\partial x_n}\mathbf{r}(\mathbf{x}^n) \\ | & & | \end{bmatrix} = \begin{bmatrix} — & \nabla r_1(\mathbf{x}^n)^T & — \\ & \vdots & \\ — & \nabla r_m(\mathbf{x}^n)^T & — \end{bmatrix} \in \mathbb{R}^{mxn} \tag{9.17}$$

Least squares problem has special forms which can be exploitted. Here, the residuals function function and its derivatives is given:

$$F(\mathbf{x}) = \frac{1}{2}||\mathbf{r}(\mathbf{x}^n)||^2 = \frac{1}{2}\mathbf{r}(\mathbf{x}^n)^T\mathbf{r}(\mathbf{x}^n) \text{ (Objective function)} \tag{9.18}$$

$$\nabla F(\mathbf{x}^n) = \mathbf{J}(\mathbf{x}^n)^T\mathbf{r}(\mathbf{x}^n) = \sum_{i=1}^{m} r_i(\mathbf{x}^n)\nabla r_i(\mathbf{x}^n) \text{ (First-order derivative)} \tag{9.19}$$

$$\nabla^2 F(\mathbf{x}^n) = \mathbf{J}(\mathbf{x}^n)^T\mathbf{J}(\mathbf{x}^n) + \sum_{i=1}^{m} r_i(\mathbf{x}^n)\nabla^2 r_i(\mathbf{x}^n) \text{ (Second-order derivative)} \tag{9.20}$$

Given 9.2.2 and 9.2.2, we elaborate the quadratic model:

$$F(\mathbf{x}^n+\Delta\mathbf{x}) \approx q^n(\Delta\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x}^n)^T\mathbf{r}(\mathbf{x}^n)+\mathbf{r}(\mathbf{x}^n)^T J(\mathbf{x}^n)\Delta\mathbf{x}+\frac{1}{2}\Delta\mathbf{x}^T(\mathbf{J}^T\mathbf{J}+\mathbf{r}(\mathbf{x}^n)^T\mathbf{H})^n\Delta\mathbf{x} \tag{9.21}$$

$$q^n_{LM}(\Delta\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x}^n)^T\mathbf{r}(\mathbf{x}^n) + \mathbf{r}(\mathbf{x}^n)^T J(\mathbf{x}^n)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T{\mathbf{B}_{\mathbf{LM}}}^n\Delta\mathbf{x} \tag{9.22}$$

$$0 = \nabla q^n(\Delta\mathbf{x}^*) = \mathbf{r}(\mathbf{x}^n)^T J(\mathbf{x}^n) + (\mathbf{J}^T\mathbf{J} + \mathbf{r}(\mathbf{x}^n)^T\mathbf{H})^n\Delta\mathbf{x}^* \tag{9.23}$$

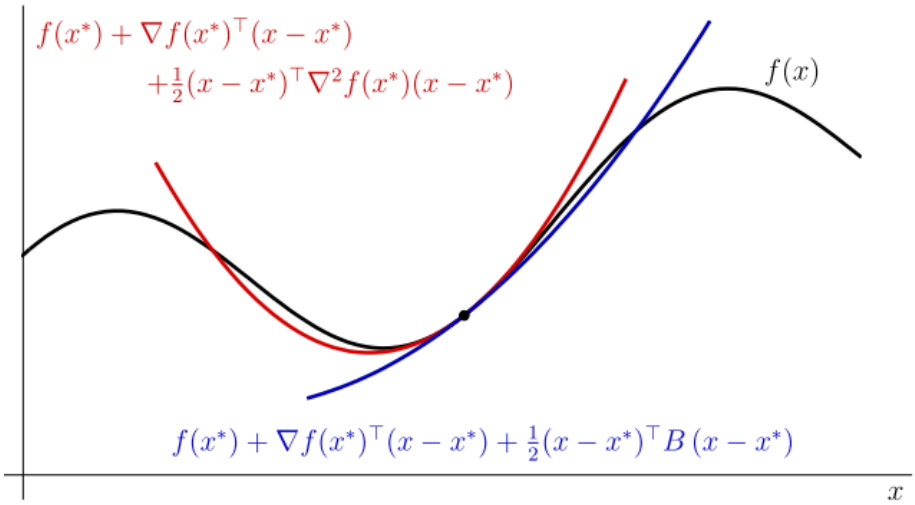


FIGURE 21.2. A function $f : \mathbb{R} \to \mathbb{R}$ and its quadratic Taylor polynomial $T_2(x; x^*)$ (in red) about the point x^*. We also show another quadratic model of the function (blue), whose second derivative (Hessian) B does not coincide with $\nabla f(x^*)$. It can be seen that the Taylor polynomial $T_2(x; x^*)$ approximates the function $f(x)$ better than the simplified model (which makes do without calculating the second derivative $\nabla f(x^*)$) does.

Figure 9.2.6: Second-order Derivative of Taylor Expansion

It is critical to note that LM does not use $\nabla^2 F(\mathbf{x})$ the exact second-order derivative (9.2.2) (also appears in the special form called *Hessian*) in its quadratic model. This is because it is computationaly expensive. Instead, we use an approximated Hessian model by removing $\sum_{i=1}^{m} r_i(\mathbf{x})\nabla^2 r_i(\mathbf{x})$ and replacing with term $\lambda^n \mathbf{I}$. Finally, the final approximated Hessian function would be:

$$\mathbf{B_{LM}}^n = \mathbf{J}(\mathbf{x}^n)^T \mathbf{J}(\mathbf{x}^n) + \lambda^n \mathbf{I} \tag{9.24}$$

where $\lambda^n > 0$ is a positive number and $\mathbf{I} \in \mathbb{R}^{kxk}$ is the identity matrix.

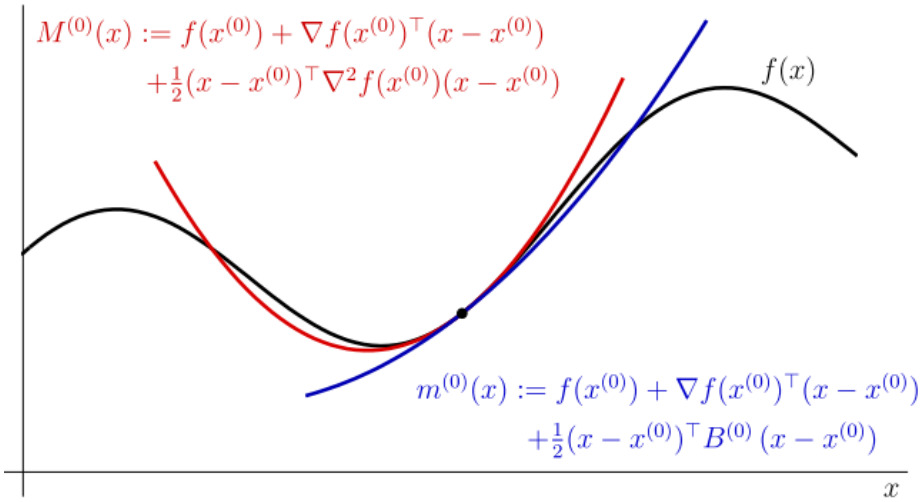


FIGURE 3.1. A function $f : \mathbb{R} \to \mathbb{R}$ and its quadratic Taylor polynomial $M^{(0)}(x)$ (in red) about the point $x^{(0)}$. We also show a simplified quadratic model $m^{(0)}(x)$ of the function (in blue) whose second-order derivative $B^{(0)}$ does not coincide with $\nabla^2 f(x^{(0)})$. It can be seen that the Taylor polynomial $M^{(0)}(x)$ approximates $f(x)$ better than the simplified model $m^{(0)}(x)$ (which makes do without calculating the second derivative $\nabla^2 f(x^{(0)})$) does.

Figure 9.2.7: Approximated Second-order Derivative of Taylor Expansion

Similar to 9.2.1, one can calculate the descent direction in the context of LM method as follows:

$$\mathbf{B_{LM}}^n \Delta \mathbf{x}^n = -\nabla F(\mathbf{x}^n) \text{ or } \Delta \mathbf{x}^n = -(\mathbf{B_{LM}}^n)^{-1} \nabla F(\mathbf{x}^n) \tag{9.25}$$

If we can write above equation using the content of Hessian model more explicitly:

$$[\mathbf{J}(\mathbf{x}^n)^T \mathbf{J}(\mathbf{x}^n) + \lambda^n \mathbf{I}] \Delta \mathbf{x}^n = -\mathbf{J}(\mathbf{x}^n)^T \mathbf{r}(\mathbf{x}^n) \tag{9.26}$$

it gives a clearer picture why $\lambda^n I$ term is used. In other gradient-descent based algorithm performs line search to determine the step length of the current iteration. In LM, this is done by tuning λ parameter, also known as *damping parameter*. For example, suppose that we assign λ a significantly small value. Then, 9.2.2 becomes:

$$\lambda^n \Delta \mathbf{x}^n \approx -\mathbf{J}(\mathbf{x}^n)^T \mathbf{r}(\mathbf{x}^n) \text{ or } \Delta \mathbf{x}^n \approx -\frac{1}{\lambda^n} \mathbf{J}(\mathbf{x}^n)^T \mathbf{r}(\mathbf{x}^n) \text{ or } \Delta \mathbf{x}^n \approx -\frac{1}{\lambda^n} \nabla F(\mathbf{x}^n) \tag{9.27}$$

Whereas, if we assign λ a significantly large value, then it becomes:

$$\mathbf{J}(\mathbf{x}^n)^T\mathbf{J}(\mathbf{x}^n)\Delta\mathbf{x}^n = -\mathbf{J}(\mathbf{x}^n)^T\mathbf{r}(\mathbf{x}^n) \tag{9.28}$$

which is a regular *Newton step* (corresponds to $\alpha = 1$ in **??**).

On the other hand, another question arises about damping parameter about how to tune the parameter so that it will allow the algorithm to converge to a local minimum efficiently and accurately. This is done by the *progress ratio* test:

$$\rho^n = \frac{F(\mathbf{x}^n) - F(\mathbf{x}^n + \Delta\mathbf{x})}{q^n_{LM}(\mathbf{0}) - q^n_{LM}(\Delta\mathbf{x})} = \frac{\text{actual decrease in objective } F(\mathbf{x})}{\text{predicted decrease by model } q^n_{LM}(\mathbf{\Delta x})} \tag{9.29}$$

Based on the p^n, we can create an empiric strategy:

1. If $p^n \geq t_2$, then it is considered as a very successful step; therefore, we can even choose smaller value for damping facor in the next iteration so that we increase the descent speed.

2. If $t_1 \leq p^n < t_2$, then it is still a successfull step but we can keep the damping factor same in the next iteration so that we don't miss out the local minimum.

3. If $p^n < t_1$, then it is a bad step; therefore, we can reject this damping factor choice and choose a larger value.

Fundemantally, this is how LM algoritm works. One must keep in mind that even the sophisticated LM algorithm might fail to converge a desired interest point on the objective function. There are two crucial factors on which any gradient descent based algorithm depends:

- *outliers* in measurement dataset,

- good *initial guess*.

It is important that we provide a good initial guess and remove outliers from dataset. If these two criteria do not meet, LM might converge to the another local minimum or might not even converge to an optimal solution. That being said, we can now summarize the algorithm into five steps:

1. build the quadratic model $q^n_{LM}(\Delta\mathbf{x}^n)$ of the objective function,

2. compute the descent direction $\Delta\mathbf{x}^n$ by solving the linear system of equations in 9.2.2,

3. calculate the progress ratio ρ^n in 9.2.2.

4. choose the next damping factor λ^{n+1} according to progress ratio test,

5. set the next iteration based on progress ratio test:
   if $\rho^n > t_1 \rightarrow \mathbf{x}^{n+1} := \mathbf{x}^n + \Delta\mathbf{x}^n$ (step accepted)
   if $\rho^n > t_1 \rightarrow \mathbf{x}^{n+1} := \mathbf{x}^n$ (step rejected)

9.3 Least Squares on a Manifold

In this section, we will explain how least squares optimization on a manifold work in the context of VO. The necessary theory and derivations are taken from Sol 2016. However, the original work was implemented for the graph SLAM problem. Thus, we change the residuals function for the VO problem accordingly. The ultimate goal in VO is to find the relative pose of a camera from k^{th} frame to $k+1^{th}$ frame. We define a relative pose as a state vector $\mathbf{x}_{k,k+1}$. Through LM algorithm, we hope to find a optimal solution $\mathbf{x}^*{}_{k,k+1}$ where the residuals are minimum. Remember that we iterately descent to the minimum by performing an addition operator $\Delta\mathbf{x}$ to each parameters in the state vector. However, one important point to note that our state vector is comprised of translation $\mathbf{t}_{k,k+1}$ and rotation $\mathbf{q}_{k,k+1}$.

$$\mathbf{x}^n_{k,k+1} = \begin{bmatrix} \mathbf{t}^n_{k,k+1} \\ \mathbf{q}^n_{k,k+1} \end{bmatrix} \in \mathbb{R}^7 \ , \ \Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{t} \\ \Delta\mathbf{q} \end{bmatrix} \in \mathbb{R}^6 \tag{9.30}$$

One can perform a regular + addition operation with translation to travel on the objective function $F(\mathbf{x}_{k,k+1})$ since it is in Euclidean space $\mathbb{R}^3$ where one can add vectors to each other. Conversely, this does not apply for rotation since it is $SO(3)$ lie group in which the elements $\phi \in \mathbb{R}^3$ of rotation are in the tangent space $\mathcal{R} \in SO(3)$. A solution to this issue would be optimizing on a manifold. Hence, we need to introduce *box-plus* operator $\boxplus : \mathcal{S}x\mathbb{R}^n \rightarrow \mathcal{S}$ where $\mathcal{S}$ is an arbitrary manifold and $\mathbb{R}^n$ is a N-dimensional real value vector space. The goal is to perform small changes that are mapped to a local neighborhood in its own state space:

$$\mathbf{x}^{n+1} = \mathbf{x}^n \boxplus \Delta\mathbf{x} \tag{9.31}$$

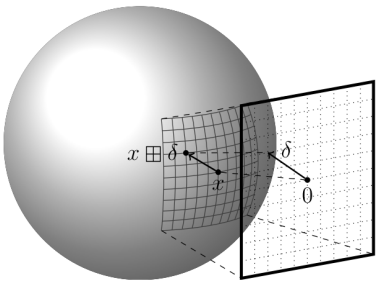


Figure 9.3.1: Mapping a local neighborhood in the state space

For translating the camera with a small euclidean vector, one can perform regular addition since $\mathbb{R}^3 x \mathbb{R}^3 \rightarrow \mathbb{R}^3$:

$$\mathbf{t}^n \boxplus \Delta\mathbf{t} = \mathbf{t}^n + \Delta\mathbf{t} = \begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{9.32}$$

TODO: CHANGE $\mathbb{R}x\mathbb{R}$ to $\mathbb{R} \times \mathbb{R}$

For rotating, however; the box-plus operation refers to $SO(3) \times \mathbb{R}^3 \rightarrow SO(3)$ and one can rotate in its local space with a small unit quaternion as follows:

$$\mathbf{q}^n \boxplus \Delta\mathbf{q} = \mathbf{q}^n \otimes \Delta\mathbf{q} = \mathbf{q}^n \otimes \begin{bmatrix} \sqrt{1 - ||\Delta\phi||^2} \\ \Delta\phi \end{bmatrix} \tag{9.33}$$

How does our new state vector with a manifold effect the LM algoritm? Remember that we form $q_{LM}(\Delta\mathbf{x})$ quadratic functions for each iteration around $\mathbf{x}^n$ and $\mathbf{J_s}$ Jacobian and $\mathbf{B_{LM,s}}$ approximated Hessian matrix are calculated by taking derivative of residuals function with respect to state vector $\mathbf{x}_{k,k+1}$. Since we modify our state vector representation and its corresponding updating operation, we need to modify the way we calculate derivatives. Here is the quadratic function with modified elements:

$$F(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \approx q_{LM}(\Delta\mathbf{x}) = \frac{1}{2}\mathbf{r_s}(\mathbf{x}^n)^T\mathbf{r_s}(\mathbf{x}^n) + \mathbf{r_s}(\mathbf{x}^n)^T\mathbf{J'_s}(\mathbf{x}^n)\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{B'_{LM,s}}(\mathbf{x}^n)\Delta\mathbf{x} \tag{9.34}$$

One can apply chain rule to form the new Jacobian matrix:

$$\begin{aligned} \mathbf{J'_s}(\mathbf{x}^n) = \mathbf{L}\frac{\partial \mathbf{r_s}(\mathbf{x}^n)}{\partial \Delta\mathbf{x}}\bigg|_{\mathbf{x}^n} &= \mathbf{L}\frac{\partial \mathbf{r_s}(\mathbf{x}^n)}{\partial (\mathbf{x}^n \boxplus \Delta\mathbf{x})}\bigg|_{\mathbf{x}^n} \frac{\partial (\mathbf{x}^n \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\bigg|_{\mathbf{x}^n, \Delta\mathbf{x}=0} \\ &= \mathbf{L}\mathbf{J}(\mathbf{x}^n)\mathbf{M}(\mathbf{x^n} \boxplus \mathbf{\Delta x}) = \mathbf{L}\mathbf{J'}(\mathbf{x}^n) = \mathbf{J'_s}(\mathbf{x}^n) \end{aligned} \tag{9.35}$$

where $\mathbf{L}$ is the matrix from the Cholesky factorization of information matrix, $\mathbf{J}(\mathbf{x}^n)$ is the older Jacobian matrix with the respect to older state vector where we assumed that all elements are in Euclidean space, $\mathbf{M}(\mathbf{x^n} \boxplus \mathbf{\Delta x})$ is the matrix that we form by taking partial derivative with respect to new state vector. Let's investigate further by breaking the new Jacobian matrix into smaller matrices to understand better. For weighting the optimization process, we assign weights with the corresponding cofindence ellipsoid of matched features from both consecutive frames by means of back- and forward-projection.

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_k^{(1)} \\ \mathbf{L}_{k+1}^{(1)} \\ \vdots \\ \mathbf{L}_k^{(m)} \\ \mathbf{L}_{k+1}^{(m)} \end{bmatrix} \tag{9.36}$$

where $\mathbf{L}_k^{(i)}$, $\mathbf{L}_{k+1}^{(i)} \in \mathbb{R}^{3x3}$ are calculated from $\mathbf{\Omega_{xyz}}_{,k}^{(i)} = {\mathbf{Q_{xyz}}_{,k}^{(i)}}^{-1}$ by factorization $\mathbf{\Omega} = \mathbf{L}\mathbf{L}^\mathbf{T}$. For each matched feature, the calculation of older Jacobian with back- and forward-projection is the following:

$$\mathbf{J}(\mathbf{x}^n) = \frac{\partial \mathbf{r}(\mathbf{x}^n)}{\partial \mathbf{x}^n}\bigg|_{\mathbf{x}^n} = \begin{bmatrix} — & \nabla\mathbf{r}^{(1)}(\mathbf{x}^n)^T & — \\ & \vdots & \\ — & \nabla\mathbf{r}^{(m)}(\mathbf{x}^n)^T & — \end{bmatrix} = \begin{bmatrix} {\mathbf{J_b}}^{(1)}(\mathbf{x}^n) \\ {\mathbf{J_f}}^{(1)}(\mathbf{x}^n) \\ \vdots \\ {\mathbf{J_b}}^{(m)}(\mathbf{x}^n) \\ {\mathbf{J_f}}^{(m)}(\mathbf{x}^n) \end{bmatrix}^T \tag{9.37}$$

where $\mathbf{J_b}^{(i)}(\mathbf{x}^n)$, $\mathbf{J_f}^{(i)}(\mathbf{x}^n) \in \mathbb{R}^{3x7}$. Notice that in older state vector, we have 3 elements from translation and 4 elements from rotation of the quaternion. Also, here is the second partial derivative of the chain rule:

$$\mathbf{M}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) = \begin{bmatrix} \mathbf{M_b}^{(1)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \\ \mathbf{M_f}^{(1)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \\ \vdots \\ \mathbf{M_b}^{(m)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \\ \mathbf{M_f}^{(m)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \end{bmatrix} \tag{9.38}$$

where $\mathbf{M_b}^{(i)}(\mathbf{x}^n \boxplus \Delta\mathbf{x})$, $\mathbf{M_f}^{(i)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \in \mathbb{R}^{7x6}$. Whereas, when taking partial derivative with the respect to new state vector that has same 3 elements from translation and 3 elements from rotation as we choose the quaternion to be a unit (so-called *local parameterization*). To take the derivative with respect to new state vector, we need to consider Euclidean space for translation and tangent space for rotation. For translation part, we don't have any further effect on the new Jacobian since we stay in the same space:

$$\begin{aligned} \mathbf{M}_{\mathbf{b}}^{(i)}(\mathbf{t}^n \boxplus \Delta\mathbf{t}) = \frac{\partial(\mathbf{t}^n \boxplus \Delta\mathbf{t})}{\partial\Delta\mathbf{t}}\bigg|_{\Delta\mathbf{t}=0} &= \frac{\partial(\mathbf{t}^n + \Delta\mathbf{t})}{\partial\Delta\mathbf{t}}\bigg|_{\Delta\mathbf{t}=0} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I}_3 \end{aligned} \tag{9.39}$$

For rotation part, however; we apply chain rule one more time to take its derivative:

$$\begin{aligned} \mathbf{M}_{\mathbf{b}}^{(i)}(\mathbf{q}^n \boxplus \Delta\mathbf{q}) = \frac{\partial(\mathbf{q}^n \boxplus \Delta\phi)}{\partial\Delta\phi}\bigg|_{\Delta\phi=0} &= \frac{\partial(\mathbf{q}^n \otimes \Delta\mathbf{q})}{\partial\Delta\mathbf{q}}\bigg|_{\Delta\phi=0} \frac{\partial\Delta\mathbf{q}}{\partial\Delta\phi} \\ &= \frac{\partial(\mathbf{Q}^+(\mathbf{q}^n)\Delta\mathbf{q})}{\partial\Delta\mathbf{q}}\bigg|_{\Delta\phi=0} \frac{\partial\begin{bmatrix}\sqrt{1-||\Delta\phi||^2} \\ \Delta\phi\end{bmatrix}}{\partial\Delta\phi}\bigg|_{\Delta\phi=0} \\ &= \mathbf{Q}^+(\mathbf{q}^n)\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} q_w^n & -q_x^n & -q_y^n & -q_z^n \\ q_x^n & q_w^n & -q_z^n & q_y^n \\ q_y^n & q_z^n & q_w^n & -q_x^n \\ q_z^n & -q_y^n & q_x^n & q_w^n \end{bmatrix}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -q_x^n & -q_y^n & -q_z^n \\ q_w^n & -q_z^n & q_y^n \\ q_z^n & q_w^n & -q_x^n \\ -q_y^n & q_x^n & q_w^n \end{bmatrix} \in \mathbb{R}^{4x3} \end{aligned} \tag{9.40}$$

Note that while rotating $\mathbf{q}^n$ with $\boldsymbol{\Delta}\mathbf{q}$, we utilize $\mathbf{Q}^+$ matrix multiplication of a quaternion rather Hamilton product for convenience. If we combine both parts into a single matrix, it will be formed as follows:

$$\mathbf{M}_{\mathbf{b}}^{(i)}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) = \left.\frac{\partial(\mathbf{x}^n \boxplus \Delta\mathbf{x})}{\partial\Delta\mathbf{x}}\right|_{\mathbf{x}^n, \Delta\mathbf{x}=0} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3x3} \\ \mathbf{0}_{4x4} & \mathbf{M}_{\Delta\phi} \end{bmatrix} \in \mathbb{R}^{7x6} \tag{9.41}$$

As explained, we now have the new Jacobian matrix based on a manifold fashion. Thus, we can solve the following linear system of equation as for the LM algorithm to calculate step length:

$$(\mathbf{J}'_{\mathbf{s}}(\mathbf{x}^n_{k,k+1})^T \mathbf{J}'_{\mathbf{s}}(\mathbf{x}^n_{k,k+1}) + \lambda^n \mathbf{I})\Delta\mathbf{x} = -\mathbf{J}'_{\mathbf{s}}(\mathbf{x}^n_{k,k+1})\mathbf{r}_{\mathbf{s}}(\mathbf{x}^n_{k,k+1}) \tag{9.42}$$

Then, we can add corresponding small changes to the new state vector:

$$\mathbf{x}^{n+1} = \mathbf{x}^n \boxplus \Delta\mathbf{x} \tag{9.43}$$

After this point, we iterate through an optimal solution with the LM algorithm as discussed in appendices 9.2.

9.4 Error Propagation Law

In state estimation applications, sensor measurements and their uncertainty are usually fused in order to keep the uncertainy of the estimated state vector bounded. The uncertainty is represented with a probability distribution, which is desired to be distributed as Gaussian. To estimate uncertainty of estimated state vector, we map the probabilty distribution function of X, $p(X)$, to the probabilty distribution function of $Y = f(X)$, $p(Y)$. This is called *error propagation law*. If $f(X)$ is a non-linear function, linearization is required.

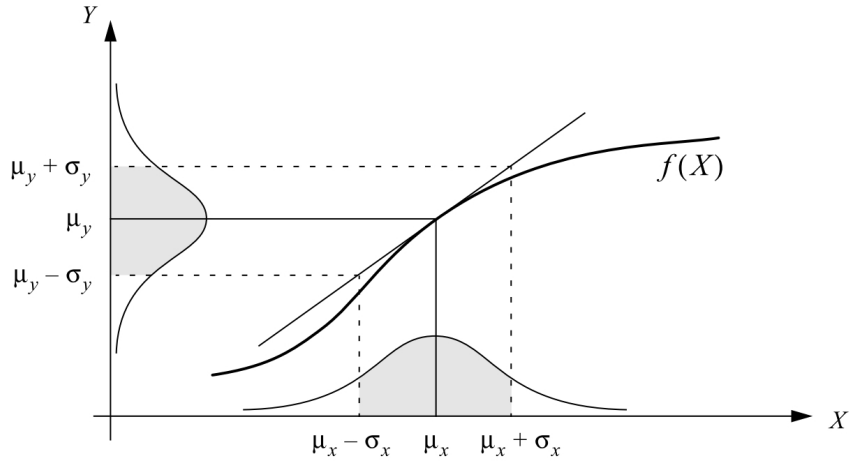


Figure 9.4.1: Error Propagation Arras and Arras 1998

Suppose that $X \sim \mathcal{N}(\mu_x, \sigma_x)$ is distributed Gaussian and we want to know how σ probability bound $[\mu_x - \sigma_x, \mu_x + \sigma_x]$ is propagated through $f(X)$. Approximation of $f(X)$ at $X = \mu_x$ can be represented with a first-order Taylor expansion:

$$Y \approx f(\mu_x) + \left.\frac{\partial f}{\partial X}\right|_{X=\mu_x} (X - \mu_x) \tag{9.44}$$

By means of linearization, we are now able to determine $\mathcal{N}(\mu_y, \sigma_y)$ with linear mapping:

$$\mu_y = f(\mu_x) \tag{9.45}$$

The interesting part is the standard deviation mapping since it is used to describe the uncertainty:

$$\sigma_y = \left.\frac{\partial f}{\partial X}\right|_{X=\mu_x} \sigma_x \tag{9.46}$$

It is important to note that propagated (μ_x, σ_x) are only approximation to real mapping $f(X)$ function. To eliminate errors caused by linearization, standard deviation σ_x should be small at known μ_x mean.

If $\mathbf{X} = (X_1, \ldots, X_n) \in \mathbb{R}^n$ has multiple inputs with multiple $f(\mathbf{X}) = \mathbf{Y} = (Y_1, \ldots, Y_m) \in \mathbb{R}^m$ outputs, we approximate with a first-order partial derivative, which is a Jacobian matrix:

$$\begin{aligned} \mathbf{Y} &\approx f(\mu_1, \ldots, \mu_n) + \sum_{i=1}^{n} [\frac{\partial f}{\partial X_i}(\mu_1, \ldots, \mu_n)][X_i - \mu_i] \\ \mathbf{Y} &\approx f(\mathbf{X}_{\mu_x}) + \mathbf{J}(\mathbf{X}_{\mu_x})(\mathbf{X} - \mathbf{X}_{\mu_x}) \end{aligned} \tag{9.47}$$

Let's propagate errors for multivariant function $f(\mathbf{X})$ with its Jacobian and covariance:

$$\mathbf{Q}_y = \mathbf{J}(\mathbf{X}_{\mu_x})^T \mathbf{Q}_x \mathbf{J}(\mathbf{X}_{\mu_x}) \tag{9.48}$$

where $\mathbf{Q}$ is the covariance matrix that corresponds to uncertainty.

9.5 Calibration Parameters of TUM RGB-D

Dataset	Parameter Name	Value	Explanation
RGB TUM FR1	f_x	517.3	focal length along x direction
	f_y	516.5	focal length along y direction
	c_x	318.6	principle offset point along x direction
	c_y	255.3	principle offset point along y direction
	k_1	0.2624	the coefficients of distortion
	k_2	-0.9531	
	k_3	1.1633	
	p_1	-0.0054	
	p_2	0.0026	
RGB TUM FR2	f_x	520.9	focal length along x direction
	f_y	521.0	focal length along y direction
	c_x	325.1	principle offset point along x direction
	c_y	249.7	principle offset point along y direction
	k_1	0.2312	the coefficients of distortion
	k_2	-0.7849	
	k_3	0.9172	
	p_1	-0.0033	
	p_2	-0.0001	

9.6 Tuning Parameters of CoVO

VO Pipeline	Parameter Name	Value	Explanation
ORB	scale_factor	1.2	Scale factor smoothing imag
	n_features	1000	Number of features
	n_levels	8	Number of pyramid levels
	edge_thres	20	FAST edge threshold (in pix
	filter_score_perc	75	Top N corners for matching (in
	wta_k	2	Number of random points to form
	patch_size	31	Number of pixels to form BRI
3D Point	depth_near_thres	0.5	Nearest depth threshold (in
	depth_far_thres	5	Farthest depth threshold (in
Operability	insuff_n_features	30	Insufficient number of matched featu
Uncertainty	var_u	8^2	Pixel noise variance along u dire
	var_v	8^2	Pixel noise variance along v dire
	var_d	$\sigma_z(z, \theta)$ (see notation 5.7)	Depth noise variance model
Pose Covariance	cov_scale	4^2	Scaling factor to keep pose covariance
Ceres	lin_solver_type	DENSE_SCHUR	Linear solver type for calculating s
	cov_solver_type	SPARSE_QR	Linear solver type for calculating covar