

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	ii
List of Tables	iii
List of Abbreviations	iv
List of Notations	v
Acknowledgments	vi
1 Introduction	1
2 Camera Models	3
2.1 The Pinhole Model	3
2.2 The Triangulation Model	8
2.3 RGB-D Camera Calibration	10
3 Fundamentals of Visual Odometry	15
3.1 Related Works	15
3.2 Feature Extraction	16
3.3 Feature Matching	20
3.4 Outlier Rejection	21
3.5 Pose Estimation	24
4 Motivation	29
5 An Error-Aware RGB-D Visual Odometry	30
5.1 Related Works	30
5.2 Algorithm Description	31
5.3 Modeling Uncertainty of RGB-D Camera	32
5.3.1 Feature Related Uncertainty	32
5.3.2 Depth Related Uncertainty	35
5.4 Pose Estimation with Uncertainties	40
5.5 Covariance of the Estimated Pose	46
6 Evaluation	49
6.1 Error Metrics	49
6.2 Simulation Environment	51
6.2.1 RPE and The Estimated Covariances	54

6.2.2	Evaluation of Estimated Covariances	56
6.3	TUM RGB-D Dataset	57
6.3.1	Discovering Effect of Pseudo Inliers on Pixel Uncertainty	58
6.3.2	Evaluation of Estimated Covariance	60
6.4	Comparison to FOVIS	61
7	Conclusion	64
8	Bibliography	66
9	Appendices	71
9.1	Rigid-Body Transformations	71
9.2	Least Squares	73
9.2.1	The Newton's Method	77
9.2.2	Levenberg-Marquardt	78
9.3	Error Propagation Law	83

List of Figures

2.0.1 Microsoft Kinect V1: Kinect is a RGB-D camera that has 640×480 pixels spatial resolution, $0.8\text{-}4.0\text{ m}$ depth range, $2\text{-}40\text{ mm}$ depth resolution and 30 fps.	3
2.1.1 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 camera coordinate system is drawn according to right-hand rule.	4
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.	5
2.1.3 Skews 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 $s = 0$	5
2.1.4 The Z-axis of the camera coordinate system aligns with the principal axis that is <i>local</i> to camera frame. In fact, we have measured 3D points that we know their position in the <i>world coordinate system</i> which is also refer to the <i>global frame</i> , as opposed to the camera coordinate system referring to <i>local frame</i>	6
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.	7
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 [KE12].	8
2.2.2 The Depth Measurement Model	9
2.2.3 Non-linear Relationship Between Inverse Depth and Disparity [SJP11]	10
2.3.1 When calibrating RGB and IR camera, one should produce many measurement points by placing checker board at different orientation and different tilted posture [Kar15].	12
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 [Kar15].	13
3.2.1 FAST corners	18

3.2.2 BRIEF pairs	19
3.3.1 Feature Mathces	20
3.4.1 Outlier Mathces	21
3.5.1 GIVE EX FOR TRANSFORMATION TWO CAM POSES Transformation between Two Images	24
3.5.2 Camera Trajectory	25
3.5.3 Minimize Reprojection Error	27
3.5.4 Minimize Euclidean Error	28
5.3.1 Conic Ray 3D Error Model with 3σ Values	33
5.3.2 Conic Ray 3D Error Model with 3σ Values	34
5.3.3 Depth Noise	36
5.3.4 Kinect Depth Noise Experiment	37
5.3.5 Kinect Depth Error Histogram at Different Distances	38
5.3.6 Kinect Noise Fitted Model	39
5.4.1	41
5.4.2 Mapping a local neighborhood in the state space	44
5.5.1 DRAW A CONFIDENCE ELLIPSOID FOR THE ESTIMATED RELATIVE POSE	47
6.2.1 asda	52
6.2.2 asda	53
6.2.3 asda	54
6.2.4 asda	55
6.2.5 asda	55
6.2.6 Remove Y axes title and Y axes thick from graphs!!!	56
6.2.7 Remove Y axes title and Y axes thick from graphs!!!	57
6.3.1 explain the window	58
6.3.2 asda	59
6.3.3 Remove Y axes title and Y axes thick from graphs!!!	60
6.3.4 4 x 4	61
6.4.1 Remove Y axes title and Y axes thick from graphs!!!	62
6.4.2 4 x 4	62
9.1.1	72
9.1.2	72
9.1.3	73
9.2.1 Local Minimum at a Quadratic Function	74
9.2.2 Least Square Model	74
9.2.3 Least Squares Measurements	75
9.2.4 Local Minimum at Sum of Squared Error Function	76
9.2.5 Least Squares Curve Fitting Operation	77
9.2.6 Second-order Derivative of Taylor Expansion	80
9.2.7 Approximated Second-order Derivative of Taylor Expansion	81
9.3.1 Error Propagation [AA98]	83

List of Tables

List of Abbreviations

DLT Direct Linear Transformation.

IMU Inertial Measurement Unit.

IR Infrared.

RGB-D RGB Color Camera with Depth Sensor.

SLAM Simultaneous Localization and Mapping.

VO Visual Odometry.

List of Notations

$\mathbf{u}^{(i,k)}, \mathbf{u}_{\text{RGB}}^{(i,k)}$ i^{th} 2D feature point's pixel coordinates on k^{th} RGB frame.

$\mathbf{u}_{\text{IR}}^{(i,k)}$ i^{th} 2D feature point's pixel coordinates on k^{th} IR frame.

$\mathbf{X}_c^{(i,k)}$ i^{th} 3D feature point's position coordinates on k^{th} frame in camera coordinate system.

$\mathbf{X}_w^{(i)}$ i^{th} 3D feature point's position coordinates in world coordinate system.

\mathbf{K}_{RGB} intrinsic matrix for RGB camera.

\mathbf{P}_{RGB} projection matrix for RGB camera.

\mathbf{T}_{RGB} transformation matrix for RGB camera.

\mathbf{P}_{IR} projection matrix for IR camera.

\mathbf{K}_{IR} intrinsic matrix for IR camera.

\mathbf{T}_{IR} transformation matrix for IR camera.

\mathbf{F}_p projection function.

\mathbf{F}_d distortion function.

\mathbf{F}_{dp} projection function combined with distortion effect.

F_d function that converts disparity value to metric depth.

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 Simultaneous Localization and Mapping (SLAM). SLAM problem has almost 30 years history [Mor80] and vision systems always had great importance. Even though some researchers [Fre10] 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 [GZS11] and FOVIS [HBH⁺11], 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.

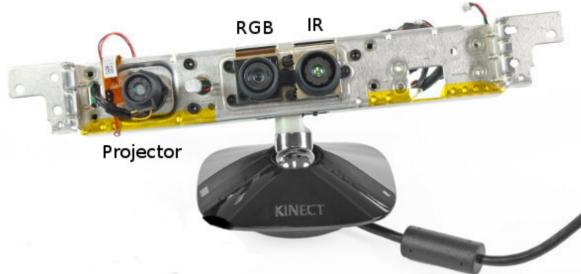


Figure 2.0.1: Microsoft Kinect V1: Kinect is a RGB-D camera that has 640×480 pixels spatial resolution, 0.8-4.0 m depth range, 2-40 mm depth resolution and 30 fps.

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*.

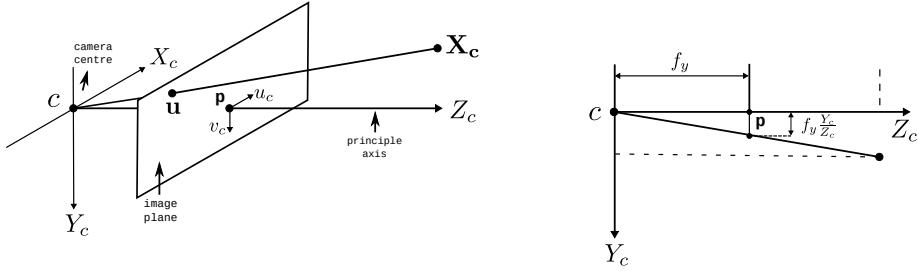


Figure 2.1.1: The Pinhole Model: C is the camera centre and \mathbf{p} is the principle point. A 3D point \mathbf{X}_c in the camera coordinate system is projected as a 2D point \mathbf{u} onto the image plane. Note that the camere coordinate system is drawn according to right-hand rule.

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:

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

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, instead of the centre.

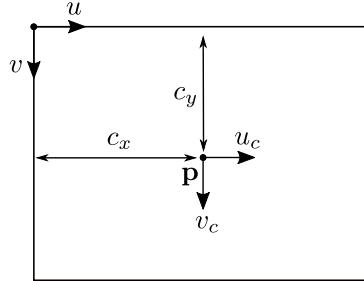


Figure 2.1.2: We take OpenCV's pixel coordinate convention where the origin is located at the upper left corner. In order to shift the origin from camera centre to the corner, c_x, c_y principle point offsets are added.

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

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

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.

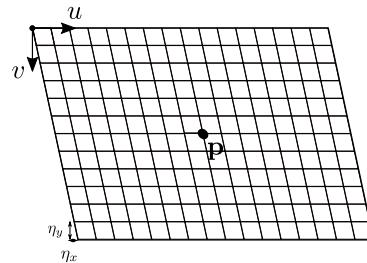


Figure 2.1.3: Skews pixels occur in earlier versions of CCD cameras because the camera's detector array is not orthogonal to the principle axis. This design issue is mostly fixed in modern cameras and thus it is usually neglected by taking $\eta_x = 1$, $\eta_y = 1$ and $s = 0$.

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

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

At this point, we can extract the following matrix:

$$\mathbf{K}_{\text{RGB}} = \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

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:

$$\mathbf{u} = \mathbf{K}_{\text{RGB}}[\mathbf{I}|0]\mathbf{X}_c \quad (2.5)$$

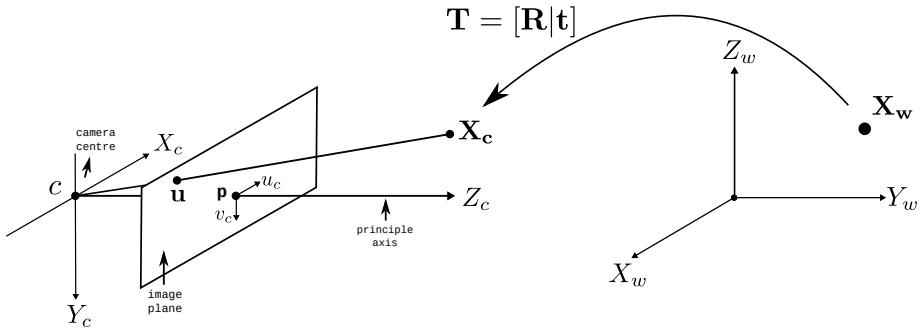


Figure 2.1.4: The Z-axis of the camera coordinate system aligns with the principal axis that is *local* to camera frame. In fact, we have measured 3D points that we know their position in the *world coordinate system* which is also refer to the *global frame*, as opposed to the camera coordinate system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

Remember that we measure 3D points in the real-world with respect to the camera centre. Thus, these two coordinate 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:

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

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

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

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

$$\mathbf{R} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.9)$$

Then, we add a translation $\mathbf{t} \in \mathbb{R}^{3 \times 1}$:

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad (2.10)$$

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

$$\mathbf{T}_{\text{RGB}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \quad (2.11)$$

The $\mathbf{T}_{\text{RGB}} \in \mathbb{R}^{4 \times 3}$ 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:

$$\mathbf{u} = \mathbf{P}_{\text{RGB}}\mathbf{X}_w = \mathbf{K}_{\text{RGB}}\mathbf{T}_{\text{RGB}}\mathbf{X}_w = \mathbf{K}_{\text{RGB}}[\mathbf{R}|\mathbf{t}]\mathbf{X}_w = \mathbf{K}_{\text{RGB}}\mathbf{X}_c \quad (2.12)$$

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

$$\mathbf{u} = \mathbf{F}_p(\mathbf{X}_w) \quad (2.13)$$

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.

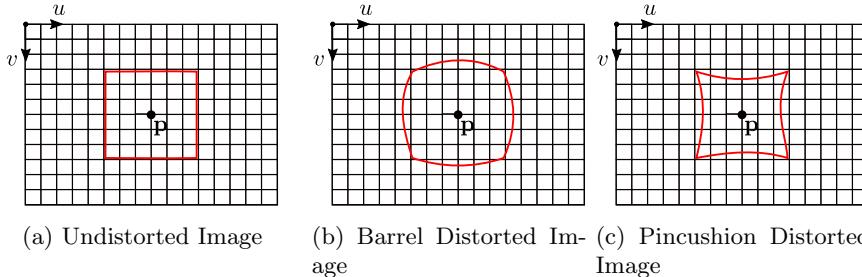


Figure 2.1.5: All emitted lights through the camera lens is expected to intersect at the same focus point and the radial distortion is caused by having a defected lens that causes the emitted lights focusing different points. This makes the straight lines appear to be curved.

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:

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

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

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

Let's combine projection and distortion function into one:

$$\mathbf{u} = \mathbf{F}_{dp}(\mathbf{X}_w) \quad (2.16)$$

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.

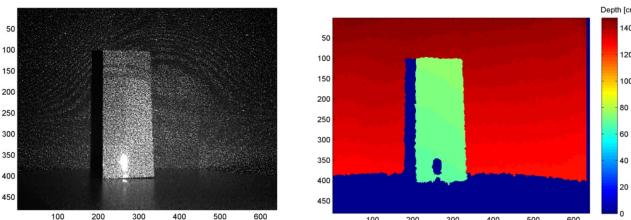


Figure 2.2.1: On the left, the IR camera capture the patterns of speckles projected on an object. On the right, we see the resulting depth image if disparity data is converted accordingly [KE12].

Since we use Kinect 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.

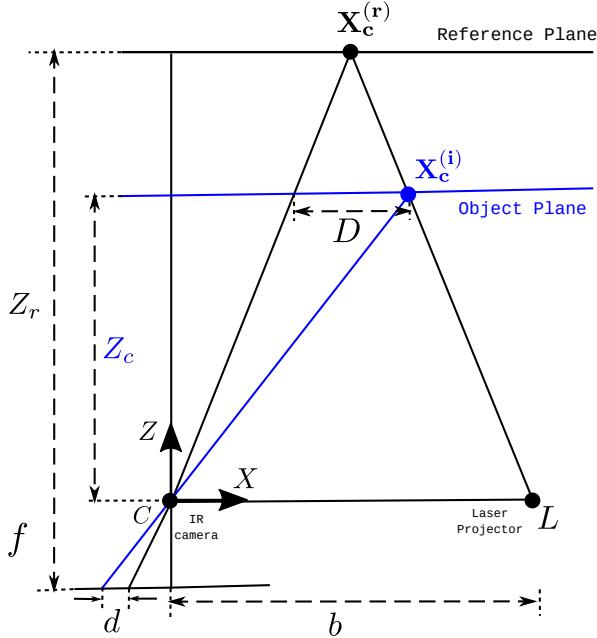


Figure 2.2.2: The Depth Measurement Model

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 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:

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

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:

$$Z_c = \frac{Z_r}{1 + \frac{Z_r}{fb}d} \quad (2.18)$$

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.

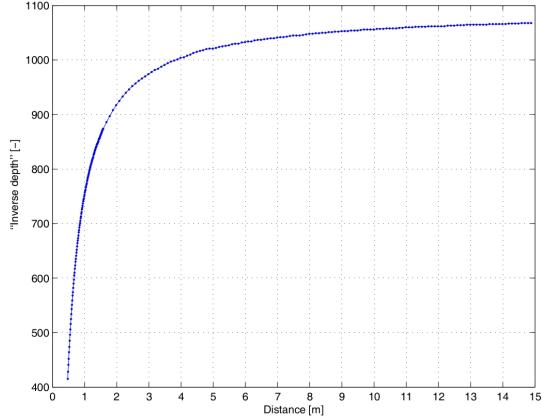


Figure 2.2.3: Non-linear Relationship Between Inverse Depth and Disparity [SJP11]

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

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

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

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

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

$$Z_c = F_d(d') \quad (2.21)$$

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.

$$\mathbf{u} = \mathbf{K}_{IR}\mathbf{F}_d(\mathbf{T}_{IR}\mathbf{X}_w) \quad (2.22)$$

Keep in mind that we first need to calibrate the IR camera's intrinsic and extrinsic parameters 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 [Zha00], [SJP11], [Kar15] and [CKH16] 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 ${}^{\text{IR}}\mathbf{T}_{\text{RGB}}$ 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:

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

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

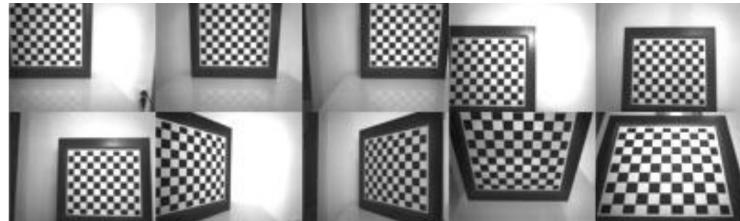
$$u^{(i)} = \frac{p_{00}X_w^{(i)} + p_{01}Y_w^{(i)} + p_{02}Z_w^{(i)} + p_{03}}{p_{20}X_w^{(i)} + p_{21}Y_w^{(i)} + p_{22}Z_w^{(i)} + p_{23}} \quad (2.24)$$

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

Since \mathbf{u} and \mathbf{X}_w are known, we can find elements of the $\mathbf{p} = (p_{00}, p_{01}, \dots, p_{23})^T$ 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, one should produce many measurement points by placing checker board at different orientation and different tilted posture [Kar15].

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:

$$\operatorname{argmin}_{\mathbf{F}_{dp} \rightarrow \mathbf{K}_{RGB}, \kappa_{RGB}, \mathbf{R}_{RGB}^{(i)}, \mathbf{t}_{RGB}^{(i)}} \sum_{i=1:n} \|\mathbf{u}_{RGB}^{(i)} - \mathbf{F}_{dp}(\mathbf{X}_w^{(i)})\|^2 \quad (2.26)$$

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 $\mathbf{X}_w^{(i)}$ 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,

- $\mathbf{R}_{\text{RGB}}^{(i)}$ is the corresponding orientation for RGB camera and
- $\mathbf{t}_{\text{RGB}}^{(i)}$ 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, $\mathbf{R}_{\text{IR}}^{(i)}$ corresponding orientation for IR camera and $\mathbf{t}_{\text{IR}}^{(i)}$ 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.

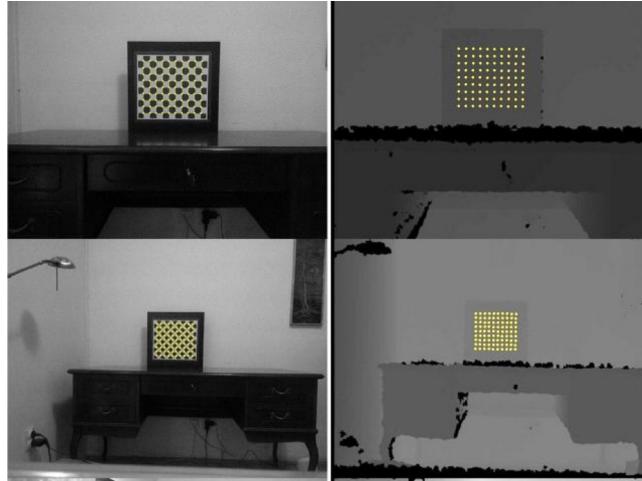


Figure 2.3.2: When calibrating the depth parameters, checkerboard should be placed at the different distances to the camera so that we get enough measurement points, especially for fitting m, n parameters [Kar15].

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

$$d'^{(i)} = M(\mathbf{u}_{\text{IR}}^{(i)}) \quad (2.27)$$

where M is a function that return the disparity value at pixel coordinates $\mathbf{u}_{\text{IR}}^{(i)} = [u_{IR}^{(i)}, u_{IR}^{(i)}]^T$ 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 :

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

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 Works

First research in estimating camera ego-motion was done in a PhD Thesis [Mor80] in 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 [MS87], formulized the ego-motion estimation for stereo vison. 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 robots since it was a critical application [OMSM03].

For the first time, Visual Odometry term was introduced in [NB04] by the computer vision researchers. 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. In computer vision, Structure from Motion (SFM), that tackles 3D reconstruction of an environment with moving camera, is 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 because VO systems started to empower other applications such SLAM and are required to operate in real-time, all of which introduces a great challenge. With the help of early robotics applications in NASA and computer vision community, the VO became quite popular field. That being said, two of the most influential papers were [FS11] and [SF11]. By publishing these tutorials, the authors gave researchers and engineers a design recipe for building the right VO system 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 [AMSI16].

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 phenomenas such as camera sensor types and solver algorithms choice. One can utilize monocular or stereo cameras in order to capture images. The stereo cameras are further grouped into two category; i.e., active and passive

cameras. For active camera, besides color or monochrome camera, there is another camera such IR camera that measures the depth information. For passive camera, the depth is calculated from two color or monochrome camera. In the case of solver algorithms, one can group VO into two category; 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. Before estimating camera poses, an important step for the feature-based algorithms is to remove outliers in feature matches with RANSAC [FB81]. There are two popular open source feature-based VO tools; i.e., LIBVISO2 [GZS11] and FOVIS [HBH⁺11]. To explain LIBVISO2 briefly, both simple blob and corner filters are used 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 passive stereo camera. Finally, they minimize 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. On the other hand, FOVIS is more accurate and faster than LIBVISO2 according to [FZ15]. FOVIS uses only FAST corner detectors to extract features. For the matchings, FOVIS uses keyframe fashion 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 biggest disadvantage of feature-based VO is that 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) [BM92] was used to minimize the geometrical error between 3D surfaces. There are also various kinds of ICP algorithms that offers to compute efficiently [RL01]. Even though ICP is useful for creating 3D shapes with point clouds, it is slower and less accurate comparing to feature-based methods. Then, another type of appearance-based approach, so-called Dense Visual Odometry (DVO) [KC13], was emerged by minimizing the photometrical error based on the pixel intensity between consecutive frames.

TODO: OUTLINE THE FOLLOWING SECTIONS. DESCRIBE VO PIPELINE BRIEFLY.

3.2 Feature Extraction

Image features are a collection of regions of interest or of points of interest that describe the image. In this way, we compress the necessary information from images so that we can achieve computationally expensive task more efficiently. Points of interest, also called *keypoints*, 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

a good feature from others is that it is distinct, repeatable, computationally cheap and invariant to geometrical changes. Obviously, one has many options to produce such image feautes but two common methods that are widely used 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 [1], SURF [2] or CENSURE [3] are popular choices for detecting blob featurus. Corners are the meeting points where two or many edges intersect. Corners, on the other hand, take advantage of geometrical structure of an image. FAST [4], Shi-Tomasi or Haris [5] 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 have greater response than your previously defined threshold as Keypoints. This might a Gaussian filter for blobs or a corner detector filter for corners. In the second step (optional), you perform non-maxima suppression on the resulting keypoints to find local minima of the function. This step will help to remove similar keypoints and to choose the ones having maximum confidence so that distinctiveness of the features are ensured.

Inherity, 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 envorinment involve more texture oriented objects like floors, walls etc. or geometrical shapes like urban areas where many lines exist? However, rule of thumb when choosing feature detector is that blobs are distinct but slow to compute and corners are fast to compute but less distict.

Extracting features is the very first step of VO. The next step is to encode the detected keypoints into a format that we can perform comparison or search operation among them. This is done by taking the neighboring pixels around the keypoints and convert into a more compact form. For example, SIFT , most well-known feature decsriptor, creates a patch around a keypoint, divides this patch into smaller grids, calculate the gradient of each grid and saves them as a histogram. This procedure makes feature descriptor robust agaist 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 in our VO and will discuss its properties.

ORB

One of the most strict requirements of VO is the real-time contraints 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, corners based feature detector are more prevalant in VO. *ORB (oriented FAST and rotated BRIEF)* studied known issues of FAST detector and BRIEF descriptor. Then, it combines them to compansate each other's drawbacks. In the end, it perform 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 discrete circular radius of $r = 9$.

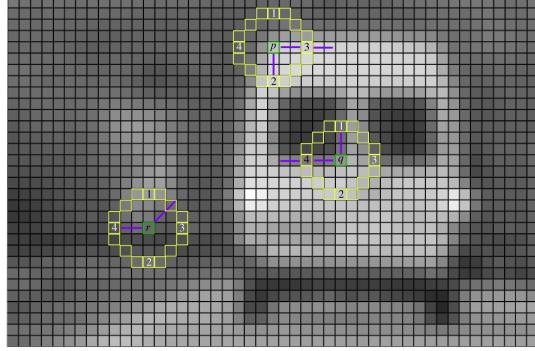


Figure 3.2.1: FAST corners

If the selected pixel is $\pm t$ darker or brighter than adjacent pixels, we call it a corner, where t is our empiric threshold.

$$M_c(p) = \sum_{r \in S_{bright}} |I_{p \rightarrow r} - I_p| - t \text{ or } M_c(p) = \sum_{r \in S_{dark}} |I_p - I_{p \rightarrow r}| - t \quad (3.1)$$

We retrieve $M_{1:n}^c$ the set of corner candidates from comparing $I_{p \rightarrow x}$ the adjacent pixels around I_p the target pixel.

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

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

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

$$R^c = \det(\mathbf{A}) - k(\text{trace}(\mathbf{A}))^2 \quad (3.3)$$

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

3. **Calculate orientation of corners with image moments:** ORB uses BRIEF to create feature descriptors; however, 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 :

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

where $a + b$ defines the order of the moment and we need the moments of order one:

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

Then, we get the orientation of the patch \mathbf{S}_n :

$$\theta(\mathbf{S}_n) = \text{atan2}(m_{01}, m_{10}) \quad (3.6)$$

4. Form BRIEF descriptors with their corresponding orientation:

Once the top N corners and their orientations are detected, descriptions can be formed with BRIEF.

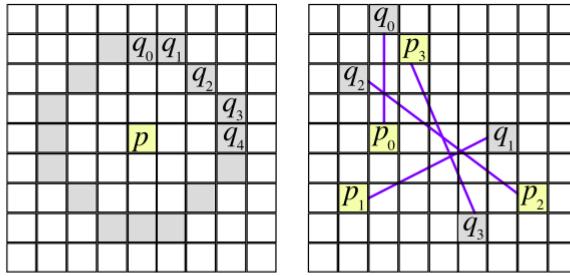


Fig. 9.12 *Left:* The figure shows one pixel location p in an image I and 16 pixel locations q on a discrete circle around p . Let $s(p, q) = 1$ if $I(p) - I(q) > 0$ and 0 otherwise. Then $s(p, q_0) \cdot 2^0 + s(p, q_1) \cdot 2^1 + \dots + s(p, q_{15}) \cdot 2^{15}$ defines the LBP code at pixel location p , i.e. a binary code of 16 bits. *Right:* BRIEF suggests the use of an order defined by random pairs of pixels within the chosen square neighbourhood, illustrated here by four pairs (p_i, q_i) , defining $s(p_0, q_0) \cdot 2^0 + s(p_1, q_1) \cdot 2^1 + s(p_2, q_2) \cdot 2^2 + s(p_3, q_3) \cdot 2^3$

Figure 3.2.2: BRIEF pairs

To do so, we randomly (normal distribution) selected 256 pairs (p_i, q_i) inside the patch \mathbf{S}_n :

$$\mathbf{S}_n = \begin{pmatrix} p_0, \dots, p_n \\ q_0, \dots, q_n \end{pmatrix} \quad (3.7)$$

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

$$\mathbf{p}_{i,\theta} = \mathbf{R}_\theta \mathbf{p}_i \text{ and } \mathbf{q}_{i,\theta} = \mathbf{R}_\theta \mathbf{q}_i \quad (3.8)$$

where $p_i = (x_i, y_i)$ and $q_i = (x_i, y_i)$ are the pixel coordinates of the points. It is important to note that authors [] 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 perform pixel density comparison: between randomly selected pair points:

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

Finally, we sum comparison results with binary form to get the descriptor of the patch \mathbf{S}_n :

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

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 first task is to form a group of image pairs and to compute translation and rotation information between each image pair, continuously. In literature, there are two ways to select image pairs: frame to frame or key frame to frame. In the former case, one pairs 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 sufficient amount of feature matchings. The latter has certain advantages over the former; however, we choose the former as we wish to model uncertainty of our motion estimation algorithm as accurate as possible.

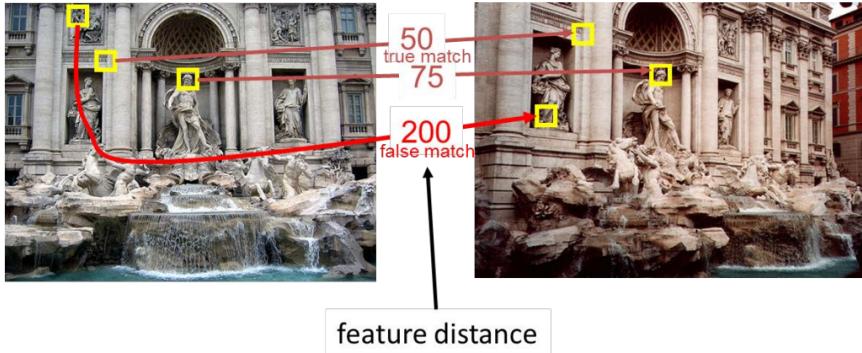


Figure 3.3.1: Feature Mathces

All being said, let's assume we pair consecutive images $(\mathbf{I}_k, \mathbf{I}_{k+1})$. In each image, we gather feature descriptors $(\mathbf{D}_k^{1:n}, \mathbf{D}_{k+1}^{1:m})$ 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 Brute-Force matching algorithm, which produces less 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 $\mathbf{D}_k^{1:n}$ descriptors in k^{th} image with $\mathbf{D}_{k+1}^{1:m}$ descriptors in $k + 1^{th}$ image by calculating *Hamming* distance:

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

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 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, which is an abbreviation to Random Sample Consensus. RANSAC is an iterative algorithm which fits desired model with presence of outliers by selecting 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.

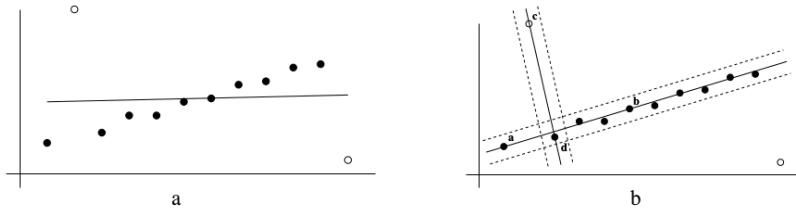


Fig. 4.7. **Robust line estimation.** The solid points are inliers, the open points outliers. (a) A least-squares (orthogonal regression) fit to the point data is severely affected by the outliers. (b) In the RANSAC algorithm the support for lines through randomly selected point pairs is measured by the number of points within a threshold distance of the lines. The dotted lines indicate the threshold distance. For the lines shown the support is 10 for line $\langle a, b \rangle$ (where both of the points a and b are inliers); and 2 for line $\langle c, d \rangle$ where the point c is an outlier.

Figure 3.4.1: Outlier Mathces

In ??, we discussed how to model rotation and translation motion along with the intrinsic matrix. Parameteres that we aim to identify are elements of projection matrix in 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:

$$k \begin{bmatrix} u'_i \\ v'_i \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \quad (3.11)$$

The goal is to fit parameters of \mathbf{H} with the selected subset of the matching and our assumption is majority of selected point matchings should be inliers. In this way, we can easily detect ourliers by testing whether they fit to model

parameters or not. To make 3.4 more obvious, we form linear system of equations:

$$\underbrace{\begin{bmatrix} u_i & v_i & 1 & 0 & 0 & 0 & -u'_i u_i & -u'_i v_i - u'_i \\ 0 & 0 & 0 & u_i & v_i & 1 & -v'_i u_i & -v'_i v_i - v'_i \end{bmatrix}}_{\mathbf{A}} = \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (3.12)$$

First step of RANSAC is to select a subset that contains minimum number of matching points to determine parameters of the model. In homography \mathbf{H} case, we need at least 4 point pairs ($\mathbf{p} = (u_i, v_i)$, $\mathbf{q} = (u'_i, v'_i)$) 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 least squares problem: With all that being said, we minimize so-called *algebraic distance error*:

$$\operatorname{argmin}_h \|\mathbf{Ah} - \mathbf{0}\|^2. \quad (3.13)$$

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:

$$\|\mathbf{q} - \mathbf{Hp}\|_2 > \mathbf{d} \quad (3.14)$$

In the third step, we include the points that passed our test procedure in the second step into our subset. In the forth step, we have another test. In this test, we check whether the number of matching points in our subset is large enough to prove that we include majority of the inliers. If not, we go back to first step and repeat the whole process again until we fulfill the fourth step. Here is the pseudo code that summarize 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 3.4, there are number of empiric parameters that we need to define; **S**, **N**, **d**, **T**. 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 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:

$$N = \log(1 - p) / \log(1 - (1 - \epsilon)^s) \quad (3.15)$$

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

For **d**, it is chosen empirically if distribution of outliers is unknown. If it is known, i.e. Gaussian with mean μ and σ , 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 that we reach expected ratio of inliers; thus we don't have to iterate through N number of times. We can terminate it earlier if the following condition is satisfied:

$$T = (1 - \epsilon)n \quad (3.16)$$

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 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

TODO: SWITCH K AND C INDICES ON TRANSLATION AND ROTATION
RESERVE!

Pose estimation is the core part of the VO system. After extracting and matching features, we finally are ready to compute $\mathbf{T}_{k,k+1}$ transformation matrix. The *transformation matrix* corresponds to relative camera motion information between two images that are recorded in different poses (see figure 3.5.1). Let's assume; we have consecutive camera poses $\mathbf{x}_k^c = [\mathbf{p}_k^c, \mathbf{q}_k^c]$ and $\mathbf{x}_{k+1}^c = [\mathbf{p}_{k+1}^c, \mathbf{q}_{k+1}^c]$ where $\mathbf{p}^c = [p_x, p_y, p_z]^T$ is the position of the camera in \mathbb{R}^3 and $\mathbf{q}^c = [q_w, q_x, q_y, q_z]^T$ is the orientation of the camera in quaternion form in $SO(3)$. Notice that, in camera model chapter 2, we used rotation matrix to represent orientations, which made it convenient to combine intrinsic matrix and extrinsic matrix (rotation and translation) into a single projection matrix (refer to 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. Additionally, 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 .

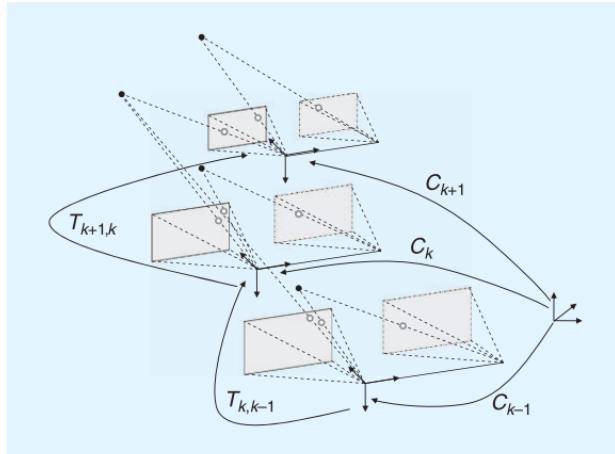


Figure 1. An illustration of the visual odometry problem. The relative poses $T_{k,k-1}$ of adjacent camera positions (or positions of a camera system) are computed from visual features and concatenated to get the absolute poses C_k with respect to the initial coordinate frame at $k = 0$.

Figure 3.5.1: GIVE EX FOR TRANSFORMATION TWO
CAM POSES Transformation between Two Images

That being said, one can formulate the transformation \mathbf{T} between two camera pose. Let's first find the camera position:

$$\mathbf{p}_{k+1} = \mathbf{q}_{k,k+1} \otimes \mathbf{p}_k^c \otimes \mathbf{q}_{k,k+1}^* + \mathbf{t}_{k,k+1} \quad (3.17)$$

where $\mathbf{q}_{k,k+1} \otimes \mathbf{p}_k \otimes \mathbf{q}_{k,k+1}^*$ is the *hamilton product* that is used to rotate

the camera position at the k^{th} pose and $\mathbf{t}_{k,k+1}$ is the simple vector addition that is used to translate the camera position. Next, the camera orientation can be found as follows:

$$\mathbf{q}_{k+1}^c = \mathbf{q}_{k,k+1} \otimes \mathbf{q}_k^c \quad (3.18)$$

where $\mathbf{q}_{k,k+1}^c \otimes \mathbf{q}_k^c$ 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.

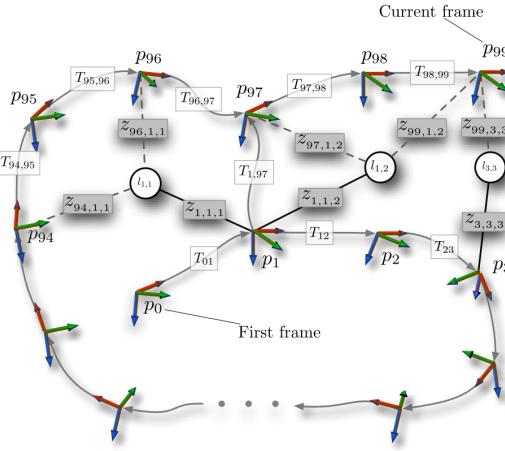


Figure 3.5.2: Camera Trajectory

The ultimate goal in VO is to compute transformation $\mathbf{T}_{k,k+1} = [\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}]$ 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:

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

$$\mathbf{q}_n^c = \mathbf{q}_{n,n-1} \otimes \dots \otimes \mathbf{q}_{2,1} \otimes \mathbf{q}_{1,0} \otimes \mathbf{q}_0^c \quad (3.20)$$

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 the best approximate transformation information due to the noise. Several methods available in literature to create relationship between feature points that we detect across different image frames.

So far, we discussed how we can process images so that we have adequate information to compute camera pose. However, we only mentioned 2D image features. To estimate the pose in 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 a metric depth information rather than relative scale in this thesis. Therefore, we have two common choices in terms of camera types: Stereo Camera or RGBD Camera. In our experiments, we experimented on RGBD 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 2 different cost functions for modeling the least problem:

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

The 2D term refers to 2D points that are in the image plane and that we call them keypoints in section-?. Whereas the 3D term refers to 3D points that are in the Camera Coordinate System and that we call them point clouds in section-?.

TODO: DEFINE ONE TERM FOR POINT CLOUD, FEATURE POINT OR LANDMARK. IT IS CONFUSING!

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. In the motivation chapter 4, I state the reasons why I chose 3D-to-3D even though it is not a common choice in VO literature.

3D-to-2D Correspondences

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

1. back-project 2D keypoints from $k + 1^{th}$ frame to 3D feature points
2. back-transform the back-projected 3D feature point from $k + 1^{th}$ frame towards k^{th} frame
3. reproject the back-transformed 3D feature point onto the k^{th} image plane
4. minimize 2D *reprojection error* along with all feature matches

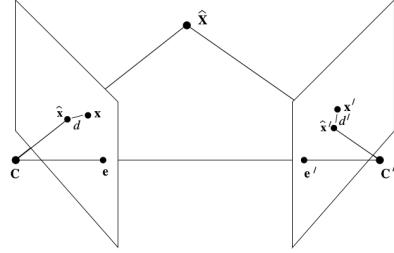


Fig. 12.2. **Minimization of geometric error.** The estimated 3-space point \hat{x} projects to the two images at \hat{x}, \hat{x}' . The corresponding image points \hat{x}, \hat{x}' satisfy the epipolar constraint, unlike the measured points x and x' . The point \hat{x} is chosen so that the reprojection error $d^2 + d'^2$ is minimized.

Figure 3.5.3: Minimize Reprojection Error

To illustrate the situation more clearly, let's assume we have a 3D feature point $\mathbf{X}_i = [x, y, z]^T$ in Camera Coordinate System and we measure the projections of this exact feature point as 2D keypoint features, $\mathbf{x}_i^k = [u_i^k, v_i^k]^T$ and $\mathbf{x}_i^{k+1} = [u_i^{k+1}, v_i^{k+1}]^T$ on subsequent camera poses k^{th} and $k+1^{th}$ respectively. What we also know is that we can back-project measured 2D keypoints to 3D feature points using projection matrix 2.12 that we estimated in calibration process. Note that, since we measure 3D feature points with respect to Camera coordinate system, we are not interested in extrinsic matrix but only intrinsic. Now, let's write again projection function that converts 3D feature points to 2D image keypoints: $\mathbf{x}_i = \mathbf{K}\mathbf{X}_i$. One can also back-project 2D image keypoints to 3D feature points: $\mathbf{K}^T\mathbf{x}_i = \mathbf{X}_i$.

Now, we can formulate the four steps of 3D-to-2D correspondences as follows:

1. $\mathbf{X}_i^{k+1} = \mathbf{K}^T\mathbf{x}_i^{k+1}$
2. $\mathbf{X}_i^{k'} = f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_i^{k+1}) = \mathbf{q}_{k,k+1} \otimes \mathbf{X}_i^{k+1} \otimes \mathbf{q}_{k,k+1}^* + \mathbf{t}_{k,k+1}$
3. $\mathbf{x}_i^{k'} = \mathbf{K}\mathbf{X}_i^{k'}$
4. minimize $\sum_i \|\mathbf{x}_i^k - \mathbf{x}_i^{k'}\|^2$ where $\mathbf{x}_i^k, \mathbf{x}_i^{k'} \in \mathbb{R}^2$

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

$$\underset{\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}}{\operatorname{argmin}} \sum_i \|\mathbf{x}_i^k - f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_i^{k+1})\|^2 \quad (3.21)$$

This method works considerably well in practice. In VO literature [], it is also reported that it performs better than 3D-to-3D correspondences.

3D-to-3D Correspondences

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

1. back-project both 2D keypoints from k^{th} and $k+1^{th}$ frames to 3D feature points

2. back-transform the back-projected 3D feature point from $k + 1^{th}$ frame
3. minimize 3D *euclidean distance* along with all feature matches

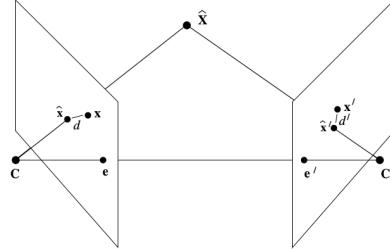


Fig. 12.2. **Minimization of geometric error.** The estimated 3-space point \hat{X} projects to the two images at \hat{x}, \hat{x}' . The corresponding image points \hat{x}, \hat{x}' satisfy the epipolar constraint, unlike the measured points x and x' . The point \hat{X} is chosen so that the reprojection error $d^2 + d'^2$ is minimized.

Figure 3.5.4: Minimize Euclidean Error

Similar illustrations 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 keypoint features, \mathbf{x}_i^k and \mathbf{x}_i^{k+1} . However, rather minimizing error on 2D image plane, we want to minimize them on 3D Camera coordinate system. Therefore, we need to back-project both keypoint features. With that in mind, we can formulate the three steps of 3D-to-3D correspondences as follows:

1. $\mathbf{X}_i^{k+1} = \mathbf{K}^T \mathbf{x}_i^{k+1}$ and $\mathbf{X}_i^k = \mathbf{K}^T \mathbf{x}_i^k$
2. $\mathbf{X}_i^{k'} = f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_i^{k+1}) = \mathbf{q}_{k,k+1} \otimes \mathbf{X}_i^{k+1} \otimes \mathbf{q}_{k,k+1}^* + \mathbf{t}_{k,k+1}$
3. minimize $\sum_i \|\mathbf{X}_i^k - \mathbf{X}_i^{k'}\|^2$ where $\mathbf{X}_i^k, \mathbf{X}_i^{k'} \in \mathbb{R}^3$

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

$$\underset{\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}}{\operatorname{argmin}} \sum_i \|\mathbf{X}_i^k - f(\mathbf{t}_{k,k+1}, \mathbf{q}_{k,k+1}, \mathbf{X}_i^{k+1})\|^2 \quad (3.22)$$

In VO literature, this method is usually discarded since it performs poorly comparing to 3D-to-2D correspondences. However, we prove 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 measurements 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 [EHS⁺14], [KA08], [DZW⁺16], [BNS18] 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 Works

To build an error-aware VO with a RGB-D camera, one needs to be able to model 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 [LLP11] and [MS93]. 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. Beacuse our VO algorithm is a feature-based, quantization, introducing a systematic error in the location of detected features, of projected 2D points onto the image plane from 3D world using RGB camera is needed to be known. This quantization error refers also as a pixel noise [Ric03] which is assumed to be maximum half pixel and normally distributed in literature. To represent the pixel uncertainty of features in 3D world, the conic ray model [Sol07] is used by taking advantage of the pinhole model of a RGB camera.

One needs to add depth uncertainty as well. For doing so, the depth camera needs to be investigated. In [MDM14], authors gathered existing studies about the depth camera's noise of Kinect. Since Kinect has two version that uses different technolog for measuring depth information, researchers compared the performance of both versions in [WS17] and [KSLK15]. Note that we use Kinect V1 which uses the Structured-Light technique. Another extensive paper is [KE12], outlining both a theoretical and experimental background for accuracy and resolution of the Kinect's depth camera based on the geometrical model. Conversely, [CLDB14] conducts a comprehensive statistical analysis to build a high-order polynomial mathematical model for the standard deviation of depth error.

Even though the studies [PSBB12], [NIL12] that deals with Kinect model the uncertainty, they don't utilize them on a VO system. In [PSBB12], the author contributed an important methodology on how to model confidence ellipsoids of 3D point clouds measured by Kinect. In this paper, standard deviation of the depth noise is assumed to constant; in fact, it increases with distance from camera quadratically. [NIL12], however, addresses this issue by defining a

quadratic function whose parameters are identified with optimization process. They even evaluate performance of their proposed model for pose estimation, which is quite relevant to our task, but settings of their experimentation is unclear since they focus mostly on 3D reconstruction. For applications such as SLAM and 3D Reconstruction, there are other papers [EHS⁺14], [KA08], [DZW⁺16] that incorporate uncertainty models to , but they assume one-pixel Gaussian noise for features. Obviously, this increases their localization accuracy. On the other hand, it is 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 [BNS18]. i They achieved it after implementing so-called reserve-SLAM simulation tool to identify the pixel noise caused by feature descriptor. The difference to to this thesis, they found out that feature detectors that uses edge filters causes error and thus they assign a confidence ellipsoid rotated along the edge. They too perform the optimization process in the presence of outliers. 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 small outliers after applying RANSAC.

5.2 Algorithm Description

Before modeling uncertainties of the RGB-D camera and integrating into the VO pipeline, we will outline the proposed error-aware VO pipeline. As many VO algorithm follow a standard architecture, parameter tuning process and algorithmic variations are performed to improve accuracy and efficiency of the system. To build the VO, we take advantage of two commonly used open source libraries; i.e, OpenCV [] for handling image feature manipulations and Ceres [] for optimization. Our proposed VO algorithm can be described in 7 steps as follows:

REVISE THESE STEPS AND REFINE THEM:

1. **Extracting Feature:** From grayscaled consecutive RGB frame, we use ORB that provides both feature extraction with FAST corners and feature descriptor with rotated BRIEF. The details of ORB can be found in the previous sections 3.2. As to parameters of ORB in OpenCV, we choose number of feature as 1000, scale factor of pyramid decimation ratio as 1.2, number of pyramid level as 8, patch size of BRIEF as 31 and FAST corner threshold as 20 pixels. These are infact default parameters except number of features.
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, cross check is performed to make sure both features are identified as a match in their own comparison set. Before applying RANSAC, an pre-filtering is applied by removing worst 25% matches based on distance calculated by Hamming window.
3. **Rejecting Outliers:** With the remaining matches, we apply RANSAC to reject outliers. In doing so, we choose RANSAC threshold to be 10 pixels.

4. **Register 2D Features With Depth:** We simply combine inlier 2D features with their corresponding depth information. It is critical to note that Kinect has invalid depth measurement which are 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 has 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 $\mathbf{J}_{bp}(\mathbf{u})$ (see notation ??).
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 ??). 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 procees, we calculate the covariance of the estimated relative camera pose by propagating it from the feature covariances \mathbf{Q}_{xyz} with the Jacobian of residuals function $\mathbf{J}'(\mathbf{x}^*)$ at the optimal solution (see notation ??).

5.3 Modeling Uncertainty of RGB-D Camera

The main reason why conventional VO applications do not provide any uncertainty information (namely covariance matrix) is that it is hard to model error characteristics of the whole VO pipeline as we perform many preprocessing, each of which eventually introduces different types of error. What we aim , in this chapter, is to define potential error source of the VO and to model them. To do so, we will investigate noise characteristic of sensors in RGB-D camera. Kinect, having three sensors: RGB camera, IR camera and Ifrared (IR) laser projector as we discussed in chapter 2. In our experiments and evaluations, we assume that these sensors are calibrated such that there are 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 source of errors into two categories: feature related and depth related uncertainties. The former is caused by feature extraction and matching algorithms. The latter is caused by depth camera sensor. In the following sections, we will discuss how we can model these two error sources and how to form an uncertainty model for the Kinect so that we estimate meaningful covariance matrices for each relative camera pose.

5.3.1 Feature Related Uncertainty

Our VO pipeline heavily relies on the detected features (also called landmarks). When building such a VO system, 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 feature, we expect not to have high-degree rotation or large amount of scaling on image features so that matching algorithm would not suffer from high number of outliers. Under this 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 half pixel due to the quantization process of the RGB camera. The ideal scenario breaks when we have outliers. What happens is that we back-project features from image plane to Camera coordinate system, transform them towards another image and project them onto the image plane to see if how these matched interest points are situated in the same image plane. If we had perfect matches, all the matches would situate closely and the error distance between matches would be half pixel. However, in reality, we still have outliers even after applying RANSAC. Thus, the error distance for those outliers would be greater than one pixel. In this thesis, we call those outliers that appears after RANSAC *pseudo inliers*. Moreover, instead of taking pixel errors half pixel, we investigate the pixel error caused by pseudo inliers in section ?? and use them as pixel uncertainty.

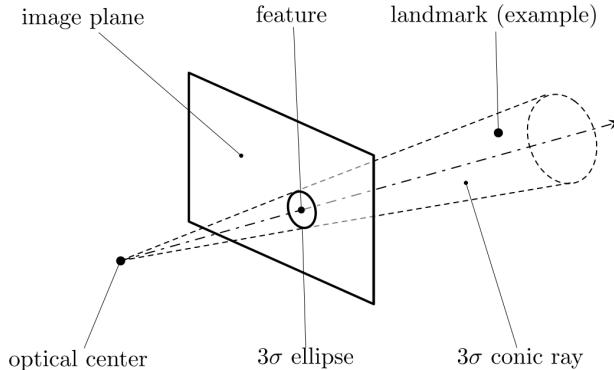


Figure 5.3.1: Conic Ray 3D Error Model with 3σ Values

Having all these in mind, let's model uncertainty related to features. First of all, remember that 3D points in world are projected to 2D points on an image plane with Pinhole Model from section-?.?. 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. That is an over-simplified definition of a color camera. With the help of pinhole model, one can build a model for light ray coming from a 3D point. Since we have aforementioned errors, we can't measure the exact location of the point. However, it is likely that the point is within the dash line region (see figure 5.4.2) if the pinhole model is considered. In this way, we form an uncertainty region which we call *conic ray*. Within this conic ray, we can represent the uncertainty

of a point with an *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. Let's assume, it is in Camera coordinate system:

$$g_{xyz}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^3 |\mathbf{Q}_{xyz}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{m}_x)^\top \mathbf{Q}_{xyz}^{-1} (\mathbf{x} - \mathbf{m}_x)\right) \quad (5.1)$$

where $\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ is the real position of the point, $\mathbf{m}_x = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix}$ 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 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 as we measure features on image plane in u and v direction. As to measurement error in z direction, we have direct knowledge (we assume that disparity data is already converted to depth information). To get errors in x and y directions, we need to propagate pixel uncertainties from image plane to Camera coordinate system. To do so, we remind ourselves with the back-projection function:

$$\mathbf{x} = \mathbf{F}_{bp}(\mathbf{u}) \quad (5.2)$$

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

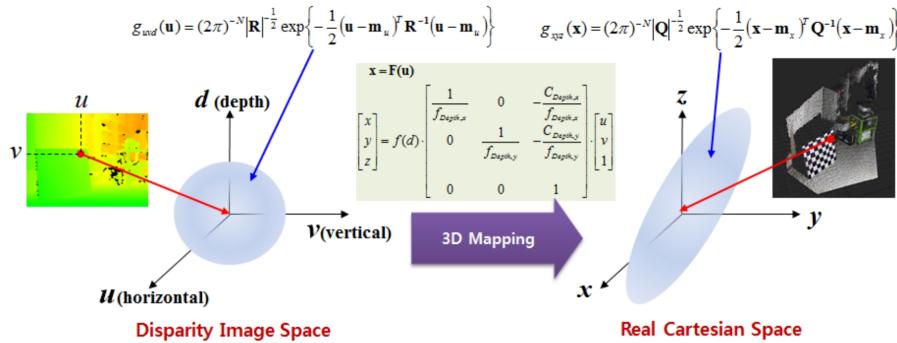


Figure 5.3.2: Conic Ray 3D Error Model with 3σ Values

One can represent the probability distribution of the pixel error with another multivariant Gaussian distribution formed by pixel uncertainties:

$$g_{uvz}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^3 |\mathbf{Q}_{uvz}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{m}_u)^\top \mathbf{Q}_{uvz}^{-1} (\mathbf{x} - \mathbf{m}_u)\right) \quad (5.4)$$

where $\mathbf{x} = \begin{bmatrix} u \\ v \\ z \end{bmatrix}$ is the real pixel coordinate along with the real depth, $\mathbf{m}_u = \begin{bmatrix} m_u \\ m_v \\ m_z \end{bmatrix}$ 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 measurement errors as pixel and depth measurements are considered independent.

To convert pixel error from image plane to Camera coordinate system, we utilize the error propagation law. In doing so, we need the partial derivative of the back-projection function:

$$\mathbf{J}_{bp}(\mathbf{u}) = \frac{\partial \mathbf{F}_{bp}(\mathbf{u})}{\partial \mathbf{u}} = \begin{bmatrix} \frac{z}{f_x} & 0 & \left(\frac{u}{f_x} - \frac{c_x}{f_x}\right) \\ 0 & \frac{z}{f_y} & \left(\frac{v}{f_y} - \frac{c_y}{f_y}\right) \\ 0 & 0 & 1 \end{bmatrix} \quad (5.5)$$

Then, we propagate the pixel covariances as follows:

$$\mathbf{Q}_{xyz} = \mathbf{J}_{bp}(\mathbf{u})^T \mathbf{Q}_{uvz} \mathbf{J}_{bp}(\mathbf{u}) \quad (5.6)$$

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

5.3.2 Depth Related Uncertainty

Modeling noise in depth measurements is more complicated than RGB camera. We explained how structured IR light speckles are projected onto an object so that IR camera can capture its deformed patterns in section 2.2. During this process, many things can go wrong. For example, (1) certain 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 in order to avoid these events as much as possible. What we aim to model in this section is mostly systematic errors in Kinect. For doing so, we rely on the experiments that is done by THIS GUY [].

Kinect's Systematic Depth Noise

An experimental analysis is conducted to measure Kinect's depth noise by THIS GUY []. To do so, they calculated the difference between ground truth and Kinect measurements. They found out that there are two types of systematic noise occurring in Kinect's depth measurements: *axial* noise and *lateral* noise.

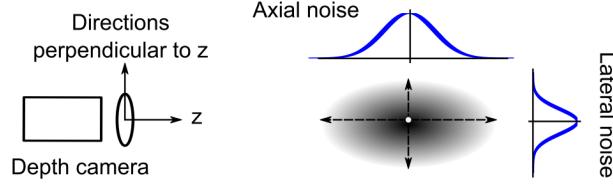


Figure 1. We propose a 3D noise distribution of Kinect depth measurement in terms of *axial* (z-direction) and *lateral* (directions perpendicular to z) noise.

Figure 5.3.3: Depth Noise

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 (see figure 5.3.3). Then, they collect depth measurements at different distance to the planar surface positioning at different angles. For calculating the axial noise, they (1) remove the lateral noise cropping edges, (2) then remaining depth region is fitted a plane that has minimum error to the ground truth and (3) they finally calculated distance difference between measured depth and ground truth. On the other hand, the lateral noise is simply calculated by taking pixel difference between fitted straight edge passing through center of distribution and measured (zigzag-like shape) pixels.

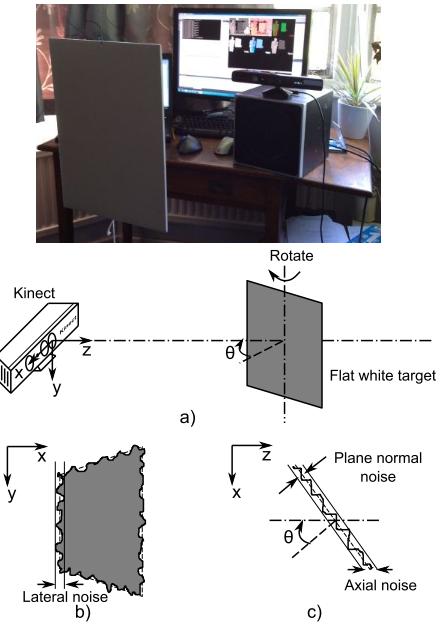


Figure 2. Experimental setup to measure noise distributions of Kinect sensor. a) Photo and schematic representation of experimental setup, with Kinect capturing depth maps of a rotating target. b) The 2D projected depth map of the planar target with lateral noise along one edge highlighted. c) A top-down cross-section of the depth map revealing noise distribution along z-axis and normal to the target plane.

Figure 5.3.4: Kinect Depth Noise Experiment

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 as seen in figure 5.3.5. Whereas, the axial noise distribution gets wider with the increased distance.

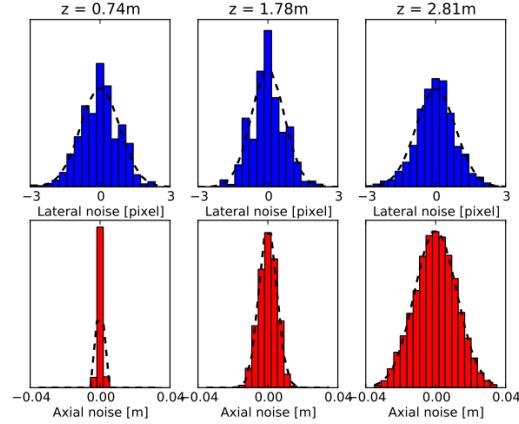


Figure 5. Measured lateral noise (top row) and axial noise (bottom row) measured for three z-distances. Dashed lines show fitted Gaussian distributions. The spread of lateral noise (in pixels) does not vary significantly with increasing z-distance, while the spread of axial noise (in meters) does.

Figure 5.3.5: Kinect Depth Error Histogram at Different Distances

However, the axial noise has another property, which is the response to the different angles. Notice the following figure that shows this issue. The standard deviation of the axial noise are increased drastically after 60 degrees.

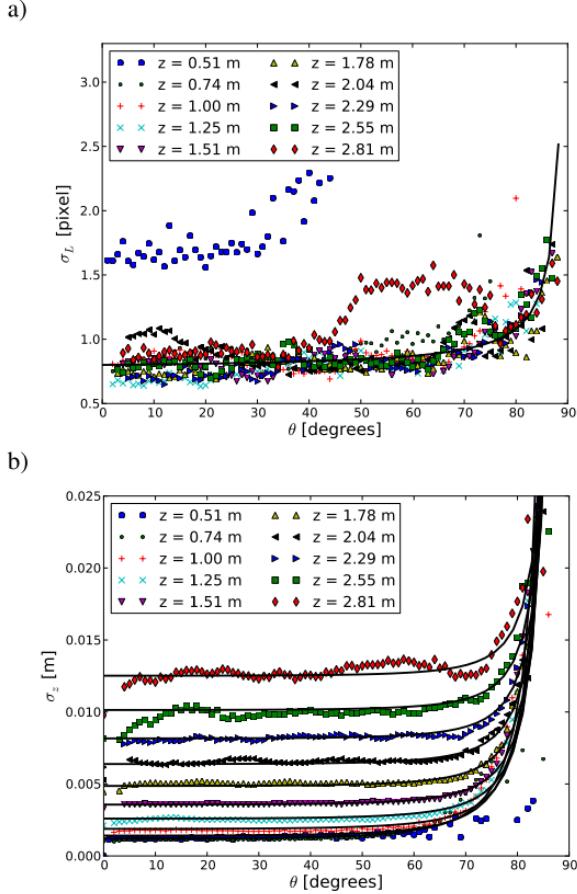


Figure 7. Fitted models plotted over a) lateral and b) z-axial noise.

Figure 5.3.6: Kinect Noise Fitted Model

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 standard deviation of the error and distance along z-axis.

$$\sigma_z(z, \theta) = 0.0012 + 0.0019 \cdot (z - 0.4)^2, 10^\circ \leq \theta \leq 60^\circ \quad (5.7)$$

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

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

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

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

This is how Kinect's depth noise is characterized experimentally by THIS GUY []. To validate these noise models' correctness, they implemented a 3D reconstruction and a camera pose tracking scenario and the noise models improved the overall accuracy of both application. 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 distinct feature in each frame. Therefore, we need to find a way to integrate the depth noise model into our feature-based VO pipeline.

Now, let's update our confidence ellipse (see 5.3.1) formulation:

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

While the axial noise can be embedded as the third dimension along the z-axis, the lateral noise, on the other hand, has an indirect relationship to overall uncertainty of a point cloud. This indirect relationship occurs when associating depth pixel coordinates with the color pixel coordinates, namely *registration*. Therefore, one can avoid this registration error by applying a smoothing filter on depth images. Remember that lateral noise of the Kinect was mostly around one pixel. Thus, a 3x3 smoothing filter can be used on extracted features or edges to compensate the error.

To summarize, we measure (1) pixel coordinates $(u^{(i)}, v^{(i)})$ and (2) depth $z^{(i)}$ from disparity data, get regarding (3) intrinsic camera parameters after calibrating and (4) estimate pixels noise and depth axial noise by conducting experimental analysis before hand. If one also knows $(X_{cam}^{(i)}, Y_{cam}^{(i)}, Z_{cam}^{(i)})$ exact coordinates of the 3D point cloud in Camera Coordinate system , one can simply calculate the probability of a measured point cloud being the real position. On the other hand, the exact location of the point clouds are infact unknown in reality. However, the probabilistic model, we built with the conic ray and confidence ellipsoids, based normally distributed noise characteristic can allows us to estimate relative camera poses by building a least squares problem. 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 covarince matrices for the measurements not only improves the convergences of the optimization algorithm but also enables us to estimate a covariance matrix for the estimated relative pose. In the following sections, we will describe how to estimate relative pose of a camera and calculate a covariance of the estimated relative pose parameters.

5.4 Pose Estimation with Uncertainties

Before diving into the formulation, it is good idea to refresh our knowledge about how to estimate relative pose of a camera using 3D-to-3D correspondences in section ???. Remember that after preprocessing extracted features, we would

have m number of measured feature matches for k^{th} and $k+1^{th}$ consecutive camera frames stored as $(\mathbf{X}_{1:m}^k, \mathbf{X}_{1:m}^{k+1})$. The 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 also wish to calculate since they refer to the relative pose of the cameras as well. As discussed earlier, after back-projecting features, the main idea was to back-transform the $\mathbf{X}_{1:m}^{k+1}$ features onto $\mathbf{X}_{1:m}^k$ features to minimize the error while optimizing the translation and rotation. Here we define the back-transform function as follows:

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

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

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

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 propagate it through uncertainty of estimated relative pose. If we utilized our conic ray model for the measured feature, we would have different covariance matrices for each feature. Let's include covariance matrices into the optimization problem:

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

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

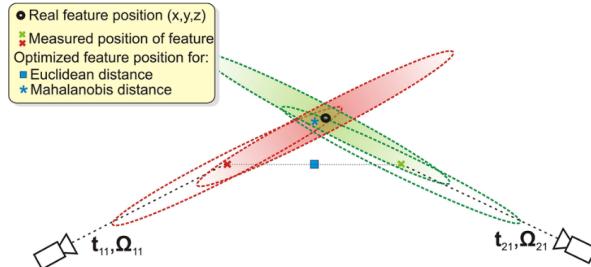


Fig. 3 2-D example which shows the results of graph optimization with an anisotropic uncertainty model of the measurements. Optimization with anisotropic uncertainty model allows to minimize Mahalanobis distance and reduce the error between estimated and real position of the feature

Figure 5.4.1

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:

$$\mathbf{r}_f^{(i)}(\mathbf{x}_{k,k+1}) = \mathbf{X}_i^{k+1} - f_f(\mathbf{x}_{k,k+1}, \mathbf{X}_i^k) \quad (5.14)$$

where the forward-transform function is defined as:

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

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

$$\mathbf{x}_{k,k+1}^* = \operatorname{argmin}_{\mathbf{x}_{k,k+1}} \sum_i \|\mathbf{r}_b^{(i)}(\mathbf{x}_{k,k+1})\|_{\mathbf{Q}_i^{k+1}}^2 + \|\mathbf{r}_f^{(i)}(\mathbf{x}_{k,k+1})\|_{\mathbf{Q}_i^k}^2 \quad (5.16)$$

With the new auxilary function, we are able to minimize the error on both forward- and back-transformation. To take advantage of the sparsity of the matrix manipulation, we further modify the residuals function. Let's define, 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. Then, let's reorganize the residual blocks by stacking residual blocks by columns:

$$\mathbf{x}_{k,k+1}^* := \operatorname{argmin}_{\mathbf{x}_{k,k+1}} \frac{1}{2} \left\| \begin{array}{c} \mathbf{r}_b^{(1)}(\mathbf{x}_{k,k+1}) \\ \mathbf{r}_f^{(1)}(\mathbf{x}_{k,k+1}) \\ \vdots \\ \mathbf{r}_b^{(m)}(\mathbf{x}_{k,k+1}) \\ \mathbf{r}_f^{(m)}(\mathbf{x}_{k,k+1}) \end{array} \right\|_{\mathbf{Q}^k, \mathbf{Q}^{k+1}}^2 \quad (5.17)$$

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 further details of 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:

$$\begin{aligned} \mathbf{x}^* &= \operatorname{argmin}_{\mathbf{x}} F(\mathbf{x}) = \operatorname{argmin}_{\mathbf{x}} \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|_{\mathbf{Q}}^2 \\ &= \frac{1}{2} \mathbf{r}(\mathbf{x})^T \mathbf{Q}_{\mathbf{uvz}}^{-1} \mathbf{r}(\mathbf{x}) \\ &= \frac{1}{2} \mathbf{r}(\mathbf{x})^T \boldsymbol{\Omega}_{\mathbf{uvz}} \mathbf{r}(\mathbf{x}) \end{aligned} \quad (5.18)$$

where $\boldsymbol{\Omega}_{\mathbf{uvz}} = \mathbf{Q}_{\mathbf{uvz}}^{-1}$ is the *information matrix*, repesent 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.

That being said, remember the main principle of the LM algorithm that is to build quadratic models around the initial guess and to descent to the nearest

local minimum iteratively. Thus, the new quadratic model for the weighted least squares can be written as follows:

$$F(\mathbf{x} + \Delta\mathbf{x}) \approx q_{LM}(\Delta\mathbf{x}) = \frac{1}{2}\mathbf{r}_s(\mathbf{x})^T\mathbf{r}_s(\mathbf{x}) + \mathbf{r}_s(\mathbf{x})^T\mathbf{J}_s(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{B}_{LM,s}(\mathbf{x})\Delta\mathbf{x} \quad (5.19)$$

where $\mathbf{r}_s(\mathbf{x}) = \mathbf{L}^T\mathbf{r}(\mathbf{x})$ is the residuals blocks, $\mathbf{J}_s(\mathbf{x}) = \mathbf{L}^T\mathbf{J}(\mathbf{x})$ is the Jacobian matrix and $\mathbf{B}_{LM,s}(\mathbf{x}) = \mathbf{J}_s(\mathbf{x})^T\mathbf{J}_s(\mathbf{x}) + \lambda\mathbf{I}$ is the approximated Hessian matrix. We can get the \mathbf{L} matrix by the Cholesky factorization of $\boldsymbol{\Omega} = \mathbf{LL}^T$. Afterwards, we can proceed with the same calculation of the regular LM algorithm that we defined in the appendices. In the end, we expect to converge to a local minimum where the translation and the rotation results are sufficient.

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta\mathbf{x} \quad (5.20)$$

Least Squares on a Manifold

Our 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 $\mathbf{q}_{k,k+1}$.

$$\mathbf{x}_{k,k+1}^n = \begin{bmatrix} \mathbf{t}_{k,k+1}^n \\ \mathbf{q}_{k,k+1}^n \end{bmatrix} \in \mathbb{R}^7, \Delta\mathbf{x} = \begin{bmatrix} \Delta\mathbf{t} \\ \Delta\mathbf{q} \end{bmatrix} \in \mathbb{R}^6 \quad (5.21)$$

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. To do so, we 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:

$$\mathbf{x}^{n+1} = \mathbf{x}^n \boxplus \Delta\mathbf{x} \quad (5.22)$$

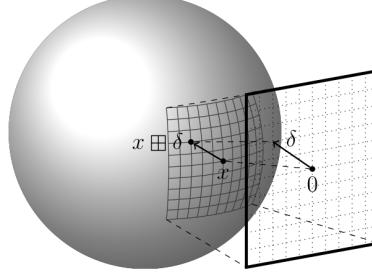


Figure 5.4.2: Mapping a local neighborhood in the state space

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

$$\mathbf{t}^n \boxplus \Delta\mathbf{t} = \mathbf{t}^n + \Delta\mathbf{t} = \begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \quad (5.23)$$

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

$$\mathbf{q}^n \boxplus \Delta\mathbf{q} = \mathbf{q}^n \otimes \Delta\mathbf{q} = \mathbf{q}^n \otimes \begin{bmatrix} \sqrt{1 - \|\Delta\phi\|^2} \\ \Delta\phi \end{bmatrix} \quad (5.24)$$

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:

$$F(\mathbf{x}^n \boxplus \Delta\mathbf{x}) \approx q_{LM}(\Delta\mathbf{x}) = \frac{1}{2} \mathbf{r}_s(\mathbf{x}^n)^T \mathbf{r}_s(\mathbf{x}^n) + \mathbf{r}_s(\mathbf{x}^n)^T \mathbf{J}'_s(\mathbf{x}^n) \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{B}'_{LM,s}(\mathbf{x}^n) \Delta\mathbf{x} \quad (5.25)$$

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

$$\begin{aligned} \mathbf{J}'_s(\mathbf{x}^n) &= \mathbf{L} \frac{\partial \mathbf{r}_s(\mathbf{x}^n)}{\partial \Delta\mathbf{x}} \Big|_{\mathbf{x}^n} = \mathbf{L} \frac{\partial \mathbf{r}_s(\mathbf{x}^n)}{\partial (\mathbf{x}^n \boxplus \Delta\mathbf{x})} \Big|_{\mathbf{x}^n} \frac{\partial (\mathbf{x}^n \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \Big|_{\mathbf{x}^n, \Delta\mathbf{x}=0} \\ &= \mathbf{L} \mathbf{J}(\mathbf{x}^n) \mathbf{M}(\mathbf{x}^n \boxplus \Delta\mathbf{x}) = \mathbf{L} \mathbf{J}'(\mathbf{x}^n) = \mathbf{J}'_s(\mathbf{x}^n) \end{aligned} \quad (5.26)$$

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, $\mathbf{M}(\mathbf{x}^n \boxplus \Delta\mathbf{x})$ 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 confidence ellipsoid of matched features from both consecutive frames by means of back- and forward-projection.

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}^{(k,1)} \\ \mathbf{L}^{(k+1,1)} \\ \vdots \\ \mathbf{L}^{(k,m)} \\ \mathbf{L}^{(k+1,m)} \end{bmatrix} \quad (5.27)$$

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

$$\mathbf{J}(\mathbf{x}^n) = \frac{\partial \mathbf{r}(\mathbf{x}^n)}{\partial \mathbf{x}^n} \Big|_{\mathbf{x}^n} = \begin{bmatrix} \nabla \mathbf{r}^{(1)}(\mathbf{x}^n)^T & \cdots \\ \vdots & \cdots \\ \nabla \mathbf{r}^{(m)}(\mathbf{x}^n)^T & \cdots \end{bmatrix} = \begin{bmatrix} \mathbf{J}_b^{(1)}(\mathbf{x}^n) \\ \mathbf{J}_f^{(1)}(\mathbf{x}^n) \\ \vdots \\ \mathbf{J}_b^{(m)}(\mathbf{x}^n) \\ \mathbf{J}_f^{(m)}(\mathbf{x}^n) \end{bmatrix}^T \quad (5.28)$$

where $\mathbf{J}_b^{(i)}(\mathbf{x}^n), \mathbf{J}_f^{(i)}(\mathbf{x}^n) \in \mathbb{R}^{3 \times 7}$. 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:

$$\mathbf{M}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) = \begin{bmatrix} \mathbf{M}_b^{(1)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) \\ \mathbf{M}_f^{(1)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) \\ \vdots \\ \mathbf{M}_b^{(m)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) \\ \mathbf{M}_f^{(m)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) \end{bmatrix} \quad (5.29)$$

where $\mathbf{M}_b^{(i)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}), \mathbf{M}_f^{(i)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) \in \mathbb{R}^{7 \times 6}$. Whereas, when taking partial derivative with 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:

$$\begin{aligned} \mathbf{M}_b^{(i)}(\mathbf{t}^n \boxplus \Delta \mathbf{t}) &= \frac{\partial(\mathbf{t}^n \boxplus \Delta \mathbf{t})}{\partial \Delta \mathbf{t}} \Big|_{\Delta \mathbf{t}=0} = \frac{\partial(\mathbf{t}^n + \Delta \mathbf{t})}{\partial \Delta \mathbf{t}} \Big|_{\Delta \mathbf{t}=0} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I}_3 \end{aligned} \quad (5.30)$$

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

$$\begin{aligned}
\mathbf{M}_{\mathbf{b}}^{(i)}(\mathbf{q}^n \boxplus \Delta \mathbf{q}) &= \frac{\partial(\mathbf{q}^n \boxplus \Delta \phi)}{\partial \Delta \phi} \Big|_{\Delta \phi=0} = \frac{\partial(\mathbf{q}^n \otimes \Delta \mathbf{q})}{\partial \Delta \mathbf{q}} \Big|_{\Delta \phi=0} \frac{\partial \Delta \mathbf{q}}{\partial \Delta \phi} \\
&= \frac{\partial(\mathbf{Q}^+(\mathbf{q}^n) \Delta \mathbf{q})}{\partial \Delta \mathbf{q}} \Big|_{\Delta \phi=0} \frac{\partial \left[\begin{array}{c} \sqrt{1 - ||\Delta \phi||^2} \\ \Delta \phi \end{array} \right]}{\partial \Delta \phi} \Big|_{\Delta \phi=0} \\
&= \mathbf{Q}^+(\mathbf{q}^n) \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} q_w^n & -q_x^n & -q_y^n & -q_z^n \\ q_x^n & q_w^n & -q_z^n & q_y^n \\ q_y^n & q_z^n & q_w^n & -q_x^n \\ q_z^n & -q_y^n & q_x^n & q_w^n \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} -q_x^n & -q_y^n & -q_z^n \\ q_w^n & -q_z^n & q_y^n \\ q_z^n & q_w^n & -q_x^n \\ -q_y^n & q_x^n & q_w^n \end{bmatrix} \in \mathbb{R}^{4 \times 3}
\end{aligned} \tag{5.31}$$

Note that while rotating \mathbf{q}^n with $\Delta \mathbf{q}$, 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:

$$\mathbf{M}_{\mathbf{b}}^{(i)}(\mathbf{x}^n \boxplus \Delta \mathbf{x}) = \frac{\partial(\mathbf{x}^n \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}} \Big|_{\mathbf{x}^n, \Delta \mathbf{x}=0} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{4 \times 4} & \mathbf{M}_{\Delta \phi} \end{bmatrix} \in \mathbb{R}^{7 \times 6} \tag{5.32}$$

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:

$$(\mathbf{J}'_{\mathbf{s}}(\mathbf{x}_{k,k+1}^n)^T \mathbf{J}'_{\mathbf{s}}(\mathbf{x}_{k,k+1}^n) + \lambda^n \mathbf{I}) \Delta \mathbf{x} = -\mathbf{J}'_{\mathbf{s}}(\mathbf{x}_{k,k+1}^n) \mathbf{r}_{\mathbf{s}}(\mathbf{x}_{k,k+1}^n) \tag{5.33}$$

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

$$\mathbf{x}^{n+1} = \mathbf{x}^n \boxplus \Delta \mathbf{x} \tag{5.34}$$

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

5.5 Covariance of the Estimated Pose

Another important question one can ask in any kind of odometry applications is that what is the uncertainty, namely covariance, of the odometry measurements? Generally, traditional VO does not provide an answer to this question since it does not take uncertainty of the features into account. However, we are now able to estimate a covariance matrix of the estimated translation and rotation parameter since we use the conic ray model to estimate feature uncertainties.

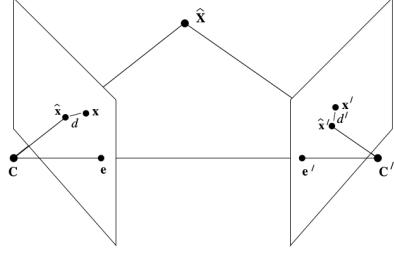


Fig. 12.2. **Minimization of geometric error.** The estimated 3-space point \hat{x} projects to the two images at \hat{x}, \hat{x}' . The corresponding image points \hat{x}, \hat{x}' satisfy the epipolar constraint, unlike the measured points x and x' . The point \hat{x} is chosen so that the reprojection error $d^2 + d'^2$ is minimized.

Figure 5.5.1: DRAW A CONFIDENCE ELLIPSOID FOR THE ESTIMATED RELATIVE POSE

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 previous section. We can simply utilize the *error propagation law* (I refer readers for to appendices 9.3 for further details of it). For doing so, we require the Jacobian matrix at the optimal solution and corresponding covariance matrices for each matched feature. Then, we can propagate them to get the covariance for the estimated parameters by applying the following equation:

$$\mathbf{Q}_{\mathbf{t}, \mathbf{q}}^{k, k+1} = \left(\frac{1}{2}\right)^2 \cdot \mathbf{J}'(\mathbf{x}^*_{k, k+1})^T \mathbf{Q}_{\mathbf{xyz}}^k \mathbf{J}'(\mathbf{x}^*_{k, k+1}) \quad (5.35)$$

where $\mathbf{J}'(\mathbf{x}^*_{k, k+1}) \in \mathbb{R}^{6m \times 6}$ is the Jacobian of extended residuals function with back-,forward-transformation and manifold in 5.4,

$$\mathbf{Q}_{\mathbf{xyz}}^{k, k+1} = \begin{bmatrix} \mathbf{Q}_{\mathbf{xyz}}^{(k,1)} & \dots & & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{\mathbf{xyz}}^{(k+1,1)} & \dots & \vdots \\ \vdots & & \vdots & \\ \mathbf{0} & & \mathbf{Q}_{\mathbf{xyz}}^{(k,m)} & \mathbf{0} \\ & & & \mathbf{Q}_{\mathbf{xyz}}^{(k+1,m)} \end{bmatrix} \in \mathbb{R}^{6m \times 6m} \quad (5.36)$$

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

$$\mathbf{Q}_{\mathbf{xyz}}^{(k,i)} = \mathbf{J}_{\mathbf{bp}}^T(\mathbf{u}) \begin{bmatrix} \sigma_u^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & (\sigma_z^{(i)}(z, \theta))^2 \end{bmatrix} \mathbf{J}_{\mathbf{bp}}(\mathbf{u}) \in \mathbb{R}^{3 \times 3} \quad (5.37)$$

is the covarince matrix for the i^{th} matched feature from k^{th} frame,

$$\mathbf{Q}_{\mathbf{t}, \mathbf{q}}^{k, k+1} = \begin{bmatrix} \sigma_{t_x}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{t_y}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{t_z}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{q_x}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{q_y}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{q_z}^2 \end{bmatrix} \in \mathbb{R}^{6 \times 6} \quad (5.38)$$

is the resulting covariance matrix for the estimated state vector parameters (in other other words, it is the uncertainty of the estiamted pose of the camera) for k^{th} and $k + 1^{th}$ consecutive frames, m is the the number of measured feature matches. Notice $(\frac{1}{2})^2$ in front of 5.5. This multiplication corresponds to the back- and forward-transformation of residuals function. To be more clear, remember that we represented a relative camera motion with both back and forward transformation in order 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 5.4). Therefore, we divide by 2. The reason we square it is because of the covariance.

Chapter 6

Evaluation

Computer vision applicatons like VO require many approximations and linearization techniques in order 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. Considering that the proposed error-aware VO algorithm outputs two information: relative pose estimation and its covariance estimation, this chapter will be mainly built 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 with these questions, simulation environment will used to validate the model at first. Then, RGB-D datasets will be used to test the algorithm in real-worl environment.

6.1 Error Metrics

There are already well-established error evaluation methods in literature so I am going to follow them in order to better understand 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 asumme 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 are hold so we need to be aware of the error caused by these issues.

Relative Pose Error

For evaluating the accuracy of a VO algorithm, it is better calculate the relative pose error rather than comparing with the absolute (whole) estimated trajectory. This is also refered as Relative Pose Error (RPE). On the other hand, some VO estimate a relative pose by frame-to-frame and some VO by keyframes. To able to compare all kinds of VO with each other, a fixed time interval Δ is chosen to calculate the local accuracy of the pose estimation. In this way, we take local

drifts into account to compare both types of VO algorithms. Thus, we defined RPE at time instance i as follows:

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

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

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

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

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

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 experiments for the same algorithm. Conversely, we take $\Delta = 30$ when comparing the proposed algorithm other VO. This 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. We do this by propagating the uncertainty of the 3D features points through the error model. In this case, it is critical to assess consistency of estimated covariance values in order to use it safely in filter-based or graph-based 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 :

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

We also take an average of NEES (ANEES) over whole trajectory:

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

TODO: find a synonym for 'it is important to note' phrase!

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, another metric when deciding on whether the estimator is consistent or not is to look at the distribution of NEES over trajectory, which is distributed as a

chi-square χ_d^2 with d degrees of freedom. For an estimator with 3 degree of freedom, acceptance region is $\hat{\epsilon} \in [2.5, 3.5]$ when significance level α of χ_d^2 is chosen 2.5% and 50 Monte Carlo runs according to [[Yaa01], pp.234-235]. It is important to note that we did not implement Manto Carlo simulation for our simulation environment. We only treat each estimation as an independent event by adding random gaussian noise to the measurements. Thus, when evaluating consistency of the estimator in simulation, we aim to get $\hat{\epsilon}_t = 3$ for translation and $\hat{\epsilon}_q = 3$ since both have 3 degree 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. To do so, we create a 3D simulation environment that consists of a camera pose, 3D feature points and their confidence ellipsoids. Our test scenario will be comprised of the following steps:

- 500 3D feature points are created within camera's observable space.
- By utilizing the pinhole model, all 3D feature points are projected onto the camera whose initial pose p_0 is known.
- Projected feature points are stored in the form of $(u_{1:500}^0, v_{1:500}^0, z_{1:500}^0)$ sensor measurements as you would usually get it from a regular RGB-D camera.
- The camera is transformed (translated and rotated) with a known distance and rotation $\mathbf{x}_{0,1} = [0.6, 0.6, 0.05, -0.183, -0.183, 0, 0.966]$ to its next pose p_1 .
- The same 3D feature points are projected with respect to the new pose p_1 and stored as $(u_{1:500}^1, v_{1:500}^1, z_{1:500}^1)$.
- 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, mean of depth noise is $\mu_z = 0$ and standard deviation is chosen with respect to feature points' distance to the camera. $\sigma_z^i(z, \theta)$. The lateral noise and the surface angle θ is assumed to be 0. The depth's noise model is discussed in 5.3.2.
- For every measurements, covariance matrices $(\mathbf{Q}_{xyz}^{0,1:500}, \mathbf{Q}_{xyz}^{1,1:500})$ are formed with the same standard deviations of the added noises, where $\mathbf{Q}_{xyz}^{(i)} = \mathbf{J}_{bp}(\mathbf{u})^T \begin{bmatrix} \sigma_u^2 (= 8^2) & 0 & 0 \\ 0 & \sigma_v^2 (= 8^2) & 0 \\ 0 & 0 & (\sigma_z^{(i)}(z, \theta))^2 \end{bmatrix} \mathbf{J}_{bp}(\mathbf{u})$.

- Perfectly matched noisy 3D feature point pairs along with their covariances are given to the optimizer that is discussed in 5.4 to estimate $\mathbf{x}^*_{0,1}$ and $\mathbf{Q}_{\mathbf{t},\mathbf{q}}^{0,1}$.
- This whole process is repeated 1000 times.

In the following simulation figures, we implement discussed scenario into a 3D space. In this environment, we have noisy 3D feature points, confidence ellipsoids whose center is around the noisy feature points and real position of the 3D feature points that are somewhere inside the ellipsoids.

$$p_0 = [0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.000]$$

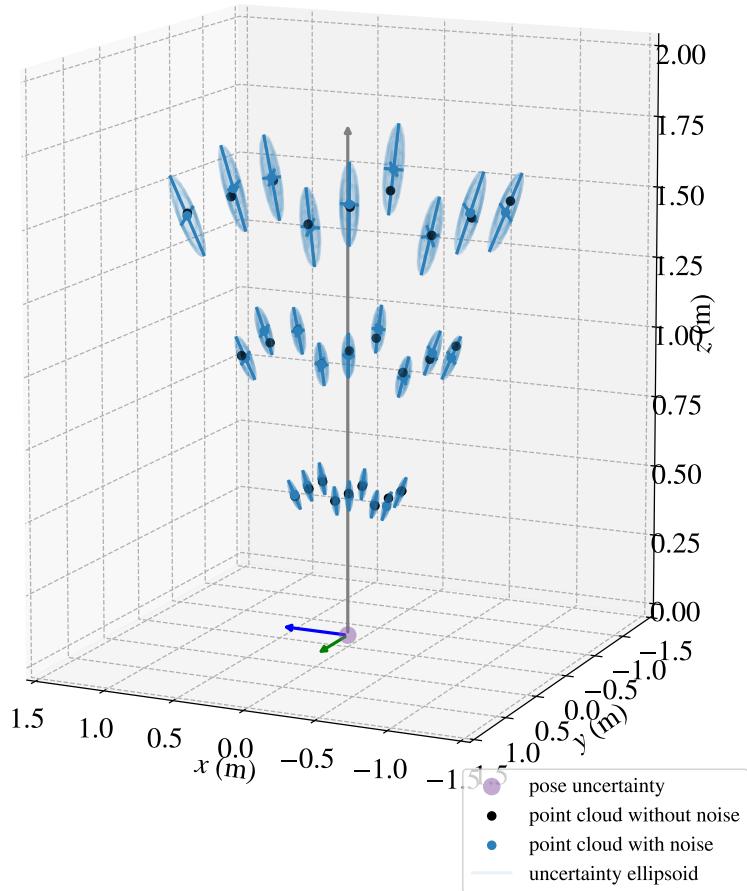


Figure 6.2.1: asda

The figure 6.2.1 depicts the environment at its initial pose p_0 . Note that we

only draw small subset of 3D feature points and scaled the confidence ellipses by 10 to gain visibility. Notice how size and orientation of the confidence ellipsoids are situated according to the conic ray model.

$$p_1 = [0.600, 0.600, 0.050, -0.183, -0.183, 0.000, 0.966]$$

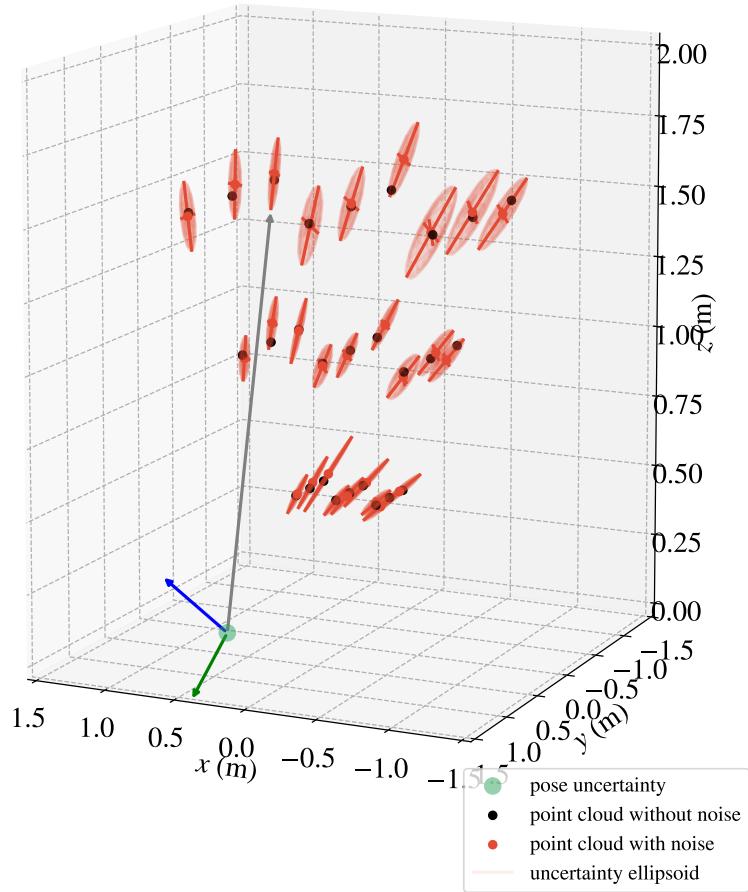


Figure 6.2.2: asda

Then, we move and rotate the camera to its next pose p_1 in the figure 6.2.2. Notice how confidence ellipsoids are changed with the new camera pose accordingly.

$$p_0 = [0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.000] \text{ and}$$

$$p_1 = [0.600, 0.600, 0.050, -0.183, -0.183, 0.000, 0.966]$$

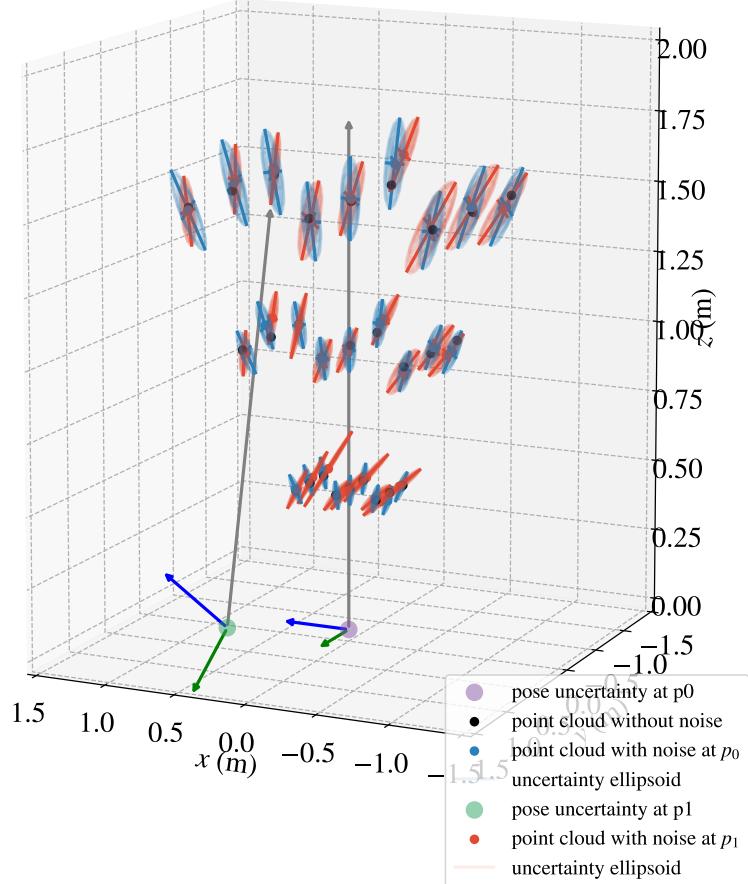


Figure 6.2.3: asda

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.

6.2.1 RPE and The Estimated Covariances

With the simulated data, we estimate relative poses $\mathbf{x}^*_{1:1000}$ and its covariances $\mathbf{x}^*_{1:1000}$. Then, we calculate RPE with $\Delta = 1$ so that we validate whether the error lies within the 3 sigma bounds for each parameters of the state vector. 3 sigma bounds corresponds to a rule of thumb that expresses 99.7% probability as near certainty.

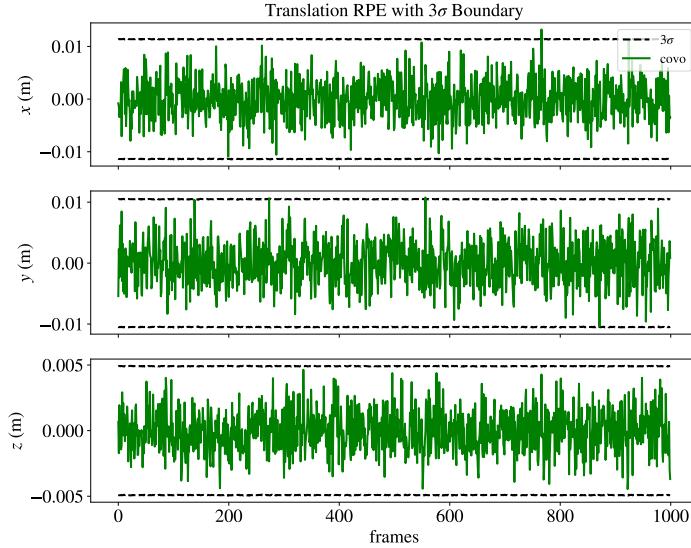


Figure 6.2.4: asda

As seen in figure 6.2.4, parameters of the translation are bounded with 3 sigma with maximum 2 or 3 predictions which is statistically acceptable.

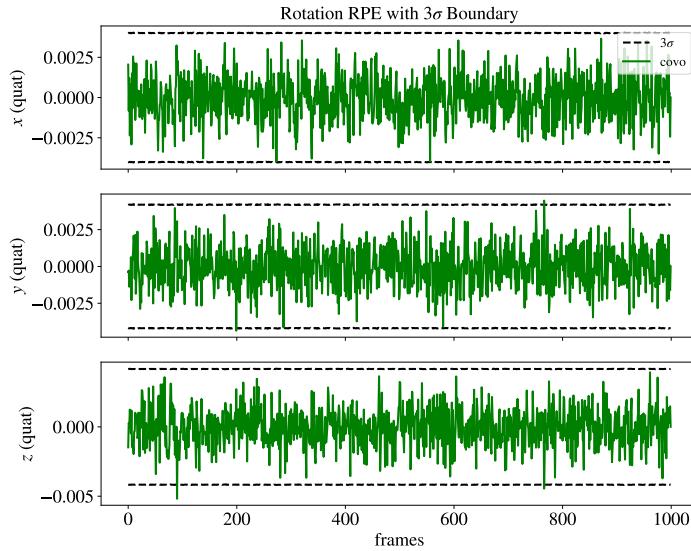


Figure 6.2.5: asda

The same behavior is observed for the rotation as well. This indicates that the propagated covariances of the estimated poses based on the proposed model produce statistically meaningful uncertainty with simulated data if 3σ bounds are considered. However, 3σ 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 3σ bounds confirm that our algorithm provides consistent pose estimation given RPE is bounded, we need to check whether the covariance of the estimated pose are being too optimistic or not. Hence, we run our simulation as described at the beginning of this section. We get ANEES; i.e, 15.64 for translation and 15.72 for rotation. We also illustrate the results in figure 6.2.2.

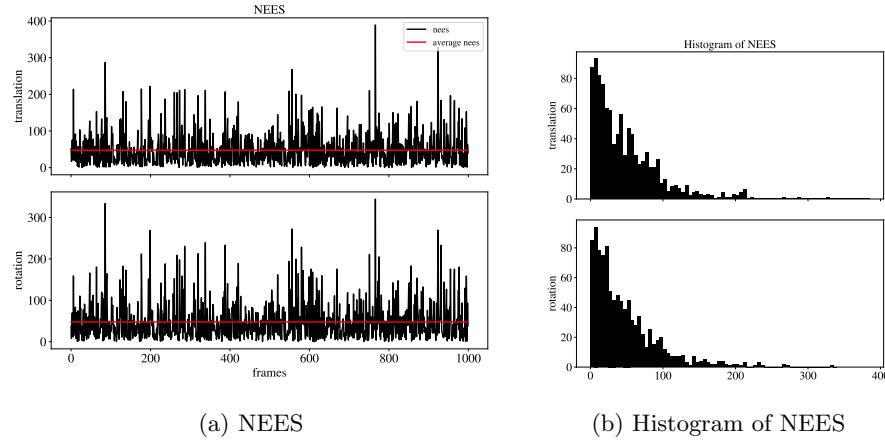


Figure 6.2.6: Remove Y axes title and Y axes thick from graphs!!!

These results show that the estimator is not consistent. This is because the projection operation introduces non-linearity to the system and we only approximate covariance of 3D feature points with first-order taylor expansion. This linearization process does not work well with large noises. In this case, [[Yaa01], pp.395-397] suggests couple of heuristic methods to make the estimator consistent. Among them, we apply multiplication of estimated covariance $\mathbf{Q}_{t,q}^{(k,k+1)}$ by scaling factor ϕ after optimiziton.

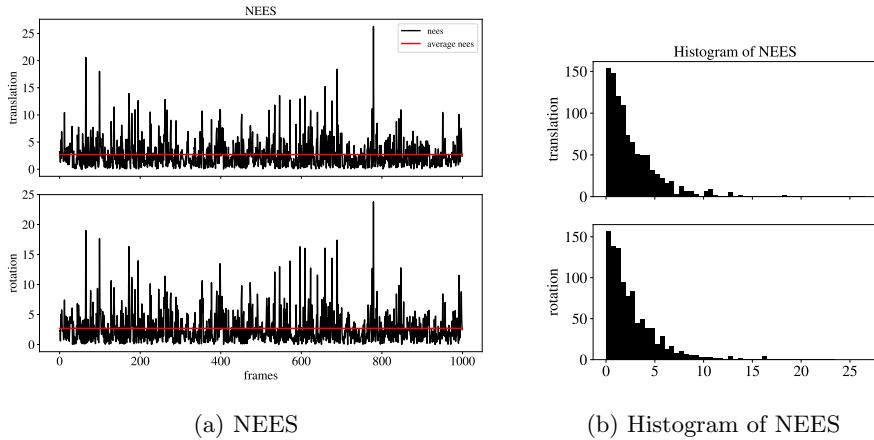


Figure 6.2.7: Remove Y axes title and Y axes thick from graphs!!!

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 6.2.2. These values can be accepted as we consider the estimator to be consistent for this setup.

6.3 TUM RGB-D Dataset

TUM dataset [SBC12] 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 environment; 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 6.3. In addition, calibration parameters of RGB and depth images are validated by the time-synchronized ground truth measurements recorded by a motion capture system.

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

In the following sections, we will use these datasets to validate our relative pose estimation and its covariance with regards to accuracy and consistency.

6.3.1 Discovering Effect of Pseudo Inliers on Pixel Uncertainty

As discussed earlier in section 5.3.1, pixel uncertainty caused by quantization operation is treated as half pixel. Nevertheless, this error is itself not sufficient when choosing standard deviation of pixel noise for covariance of 3D features to be propagated through the ?? equation. 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, but they will increase the pixel uncertainty. We also assume that RANSAC will bound these pseudo inliers. Thus, we are interested in determining the boundaries which we will set them as pixel uncertainties. We determine them using TUM RGB-D and the ground truth measurements.

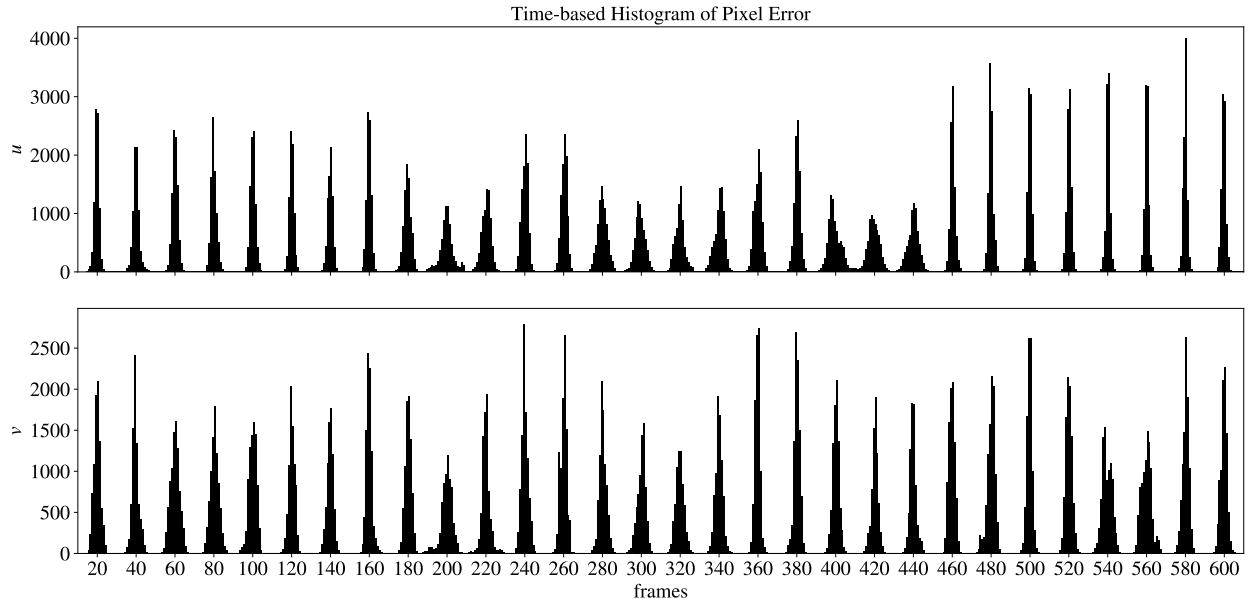


Figure 6.3.1: explain the window

Let's remind ourselves with the procedures of the pipeline. To find the pixel error caused by pseudo inliers, we run our VO pipeline up until the optimization part estimating the pose. To do so, we first extracted features, then matched them and finally filtered outliers with RANSAC. At this point, we have feature matches that includes pseudo inliers. Now, instead of estimating the relative pose, we can identify boundaries of the pseudo inliers caused by RANSAC by using ground truth data. With using translation and rotation information from ground truth, we project features matches one on another. Then, for some matches, we expect to get pixel errors to be greater than 1 as they are the pseudo inliers. This test case is applied on every consecutive frame of the whole trajectory. Next, 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 observed, pixels error mainly caused

by pseudo inliers are distributed as gaussian and its standard deviation changes over the trajectory. To see how it changes we draw the time-based standard deviation of the pixel errors in figure 6.3.1.

TODO: THERE IS SOMETHING WRONG WITH THE NUMBERING OF FIGS.

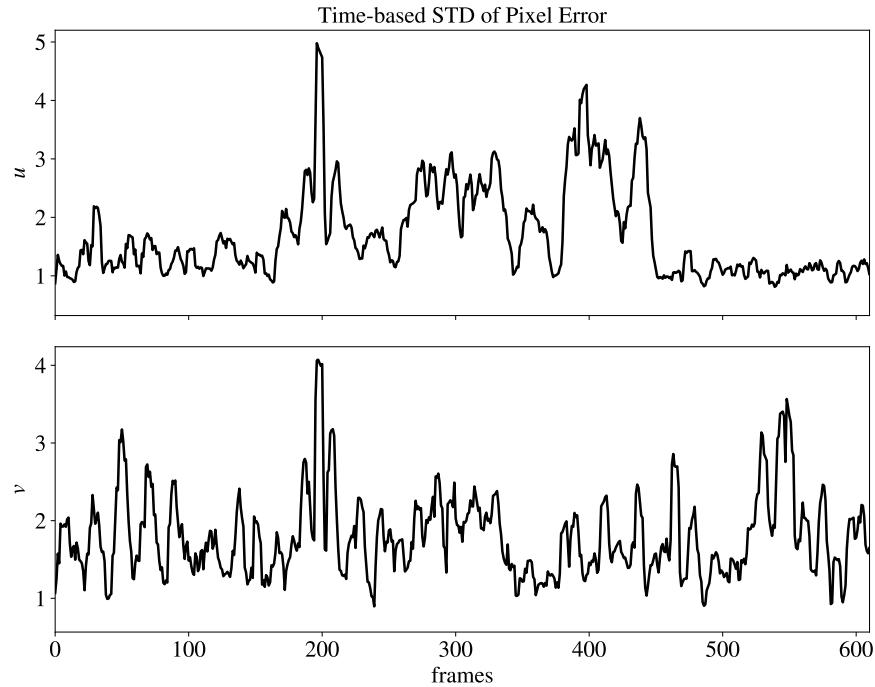


Figure 6.3.2: asda

The goal is to run this test over many datasets so that we can find the largest standard deviation. The result of these tests are represented with the boxplots in figure 6.3.1.

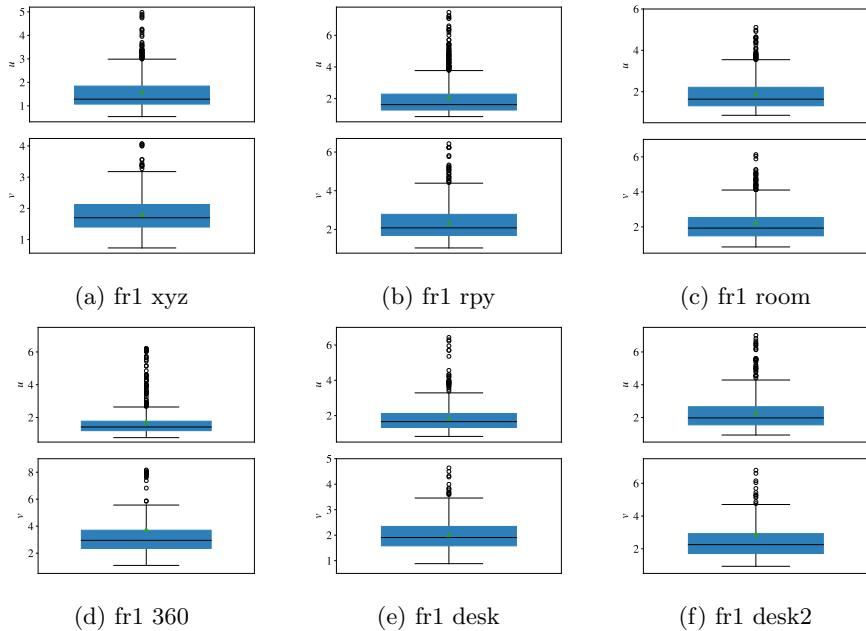


Figure 6.3.3: Remove Y axes title and Y axes thick from graphs!!!

To cover the worst case scenario, we take these values as largest value of standard deviations of the pixel noise when forming covariance matrix $\mathbf{Q}_{uvz}^{(i,k)}$.

6.3.2 Evaluation of Estimated Covariance

Similar to simulation data, we calculate NEES for covariance of the estimated poses after running the VO algorithm 6 different datasets in TUM. Note that we set standard deviation of pixel uncertainty according to the largest value $\sigma_u = 7$ and $\sigma_v = 8$ determined in the experimental analysis done in previous section. After each optimization, we scale resulting covariance $\mathbf{Q}_{\mathbf{t}, \mathbf{q}}^{(k, k+1)}$ with the $\phi = 4^2$ to compensate the errors due to the linearization. Under these parameterizations, we run the algorithm and calculate histogram of NEES and ANEES for the selected datasets. The resulting histograms are illustrated in 6.3.2 and their ANEES are given in table 6.3.2.

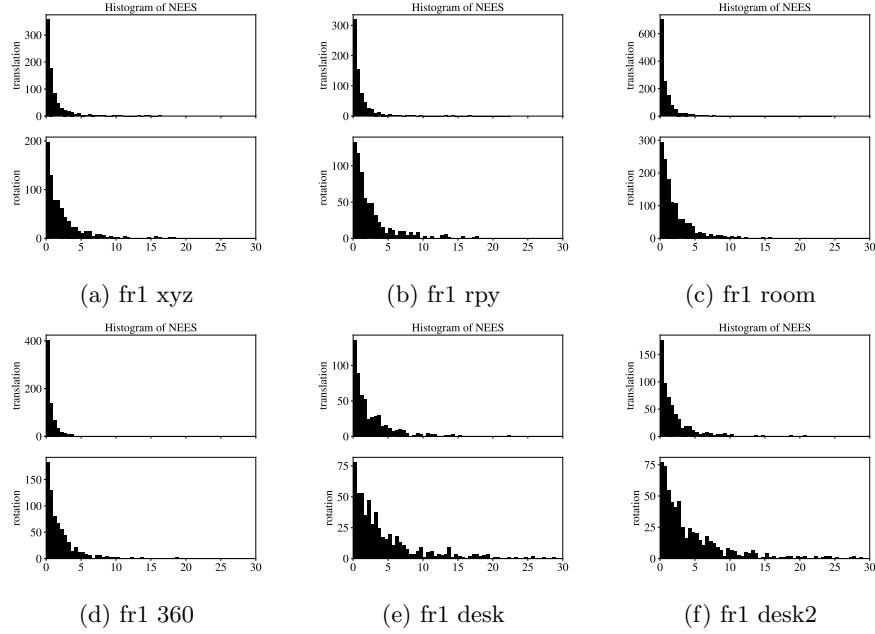


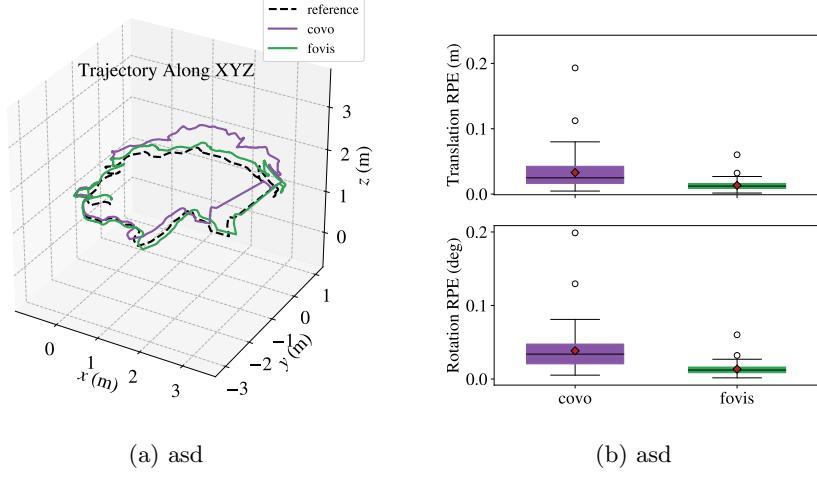
Figure 6.3.4: 4×4

We observe that covariance of the translation estimations are conservative, 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.

	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

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 as to building trajectory from relative pose estimations, we draw estimations results of 'fr2 desk' which is suitable to present drift effects and it is given in 6.4 along with the boxplot representation of RPEs.



(a) asd

(b) asd

Figure 6.4.1: Remove Y axes title and Y axes thick from graphs!!!

We run both algorithm for the 7 datasets and the resulting RMSEs of RPEs are given in table 6.4. Note that we take $\Delta = 30$ frames while calculating RPEs.

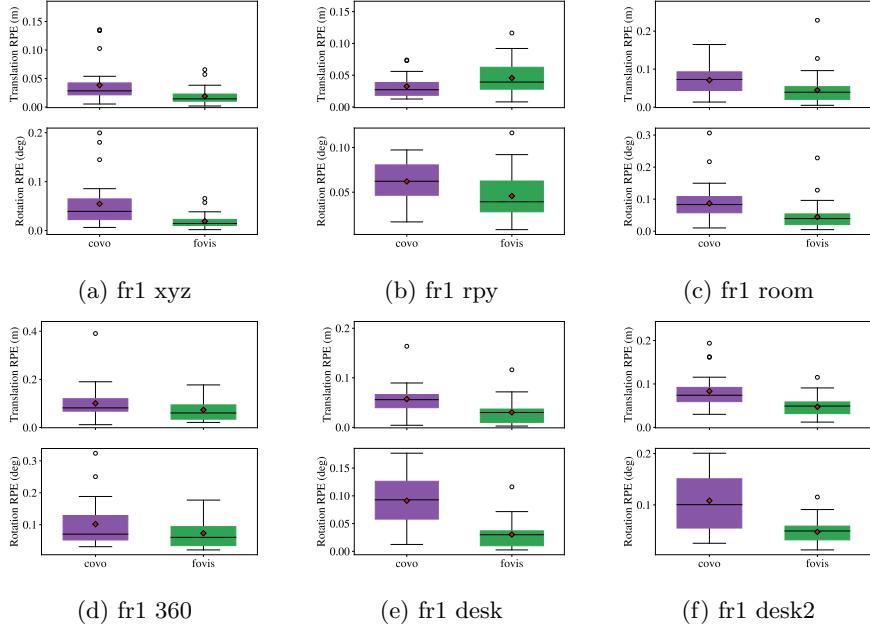


Figure 6.4.2: 4 x 4

	FOVIS	COVO
	RMSE	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

We can say that FOVIS is more accurate than the proposed algorithm and name number of reasons. FOVIS utilizes the keyframe scheme and different outlier rejection algorithm than RANSAC and provides a better initial guess to its optimizer after several refinement processes throughout the pipeline.

Chapter 7

Conclusion

This thesis was undertaken to design an error-aware RGB-D Visual Odometry system and evaluate its credibility. This achieved by identifying the errors occurring 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 uncertainty 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 correctness 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 chapter, 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

- [AA98] Ko Arras and Ko Arras. An Introduction to Error Propagation: Derivation, Meaning, and Examples of Equation $cy = fx$ cx fx . *Lausanne: Swiss Federal Institute of Technology Lausanne (EPFL)*, (September):98–01, 1998.
- [AGC⁺17] V. Angladon, S. Gasparini, V. Charvillat, T. Pribanić, T. Petković, M. onlić, B. Ahsan, and F. Bruel. An evaluation of real-time RGB-D visual odometry algorithms on mobile devices. *Journal of Real-Time Image Processing*, 2017.
- [AMSI16] Mohammad O.A. Aqel, Mohammad H. Marhaban, M. Iqbal Saripan, and Napsiah Bt Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1), 2016.
- [BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.
- [BNS18] Dominik Belter, Michał Nowicki, and Piotr Skrzypczyński. Modeling spatial uncertainty of point features in feature-based RGB-D SLAM. *Machine Vision and Applications*, 29(5):827–844, 2018.
- [CKH16] Daniel Herrera C., Juho Kannala, and Janne Heikkila. Joint depth and color camera calibration with distortion. pages 144–147, 2016.
- [CLDB14] Benjamin Choo, Michael Landau, Michael Devore, and Peter A Beling. Statistical Analysis-Based Error Models for the Microsoft. *sensors*, pages 17430–17450, 2014.
- [DVXM13] Ivan Dryanovski, Roberto G Valenti, Jizhong Xiao, and Senior Member. Fast Visual Odometry and Mapping from RGB-D Data. *2013 IEEE International Conference on Robotics and Automation*, pages 2305–2310, 2013.
- [DZW⁺16] Kaichang Di, Qiang Zhao, Wenhui Wan, Yexin Wang, and Yunjun Gao. RGB-D SLAM based on extended bundle adjustment with 2D and 3D information. *Sensors (Switzerland)*, 16(8), 2016.
- [EHS⁺14] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. 3D Mapping with an RGB-D Camera. *IEEE TRANSACTIONS ON ROBOTICS*, 30(1):1–11, 2014.

- [FB81] Martin A Fischler and Robert C Bolles. Random Sample Paradigm for Model Consensus: A Appcatlons to Image Fitting with Analysis and Automated Cartography. 24(6), 1981.
- [Fre10] Udo Frese. Interview: Is SLAM Solved? *KI - Künstliche Intelligenz*, 24(3):255–257, 2010.
- [FS11] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part I: The First 30 Years and Fundamentals. *IEEE Robotics and Automation Magazine*, 19(2):78–90, 2011.
- [FZ15] Zheng Fang and Yu Zhang. Experimental evaluation of RGB-D visual odometry methods. *International Journal of Advanced Robotic Systems*, 12:1–16, 2015.
- [GZS11] Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3d Reconstruction in Real-time. *IEEE Intelligent Vehicles Symposium (IV)*, pages 1–9, 2011.
- [HBH⁺11] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. *International Symposium of Robotics Research*, 2011.
- [HH18] Prof R Herzog and Prof C Helmberg. Lectures Notes Optimization for Non-Mathematicians. Technical report, 2018.
- [KA08] Kurt Konolige and Motilal Agrawal. FrameSLAM : From Bundle Adjustment to Real-Time Visual Mapping. *IEEE TRANSACTIONS ON ROBOTICS*, 24(5):1066–1077, 2008.
- [Kar15] Branko Karan. Calibration of kinect-type RGB-D sensors for robotic applications. *FME Transactions*, 43(1):47–54, 2015.
- [KC13] Christian Kerl and Daniel Cremers. Robust Odometry Estimation for RGB-D Cameras. *IEEE International Conference on Robotics and Automation*, pages 3748–3754, 2013.
- [KE12] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.
- [KSLK15] Time-of-flight Kinect, Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect. 2015.
- [LLP11] Giuseppe Di Leo, Consolatina Liguori, and Alfredo Paolillo. Covariance Propagation for the Uncertainty Estimation in Stereo Vision. *IEEE Transactions on Instrumentation and Measurement*, 60(5):1664–1673, 2011.
- [LZJ01] X Rong Li, Zhanlue Zhao, and Vesselin P Jilkov. Practical Measures and Test for Credibility of an Estimator. *Proc. Workshop on Estimation, Tracking, and Fusion*, pages 481—495, 2001.

- [MDM14] Tanwi Mallick, Partha Pratim Das, and Arun Kumar Majumdar. Characterizations of noise in Kinect depth images: A review. *IEEE Sensors Journal*, 14(6):1731–1740, 2014.
- [Mor80] Hans P Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, 1980.
- [MS87] L Matthies and S A Shafer. Error Modeling in Stereo Vision. *IEEE J. Robotics and Automation*, 3(3):239–248, 1987.
- [MS93] Jun Miura and Yoshiaki Shirai. An Uncertainty Model of Stereo Vision and its Application to Vision-Motion Planning of Robot. *Proc. 13th Int. Joint Conf. on Artificial Intelligence*, (August):1618–1623, 1993.
- [NB04] David Nist and James Bergen. Visual Odometry. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.
- [NIL12] Chuong V. Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3D reconstruction and tracking. *Proceedings - 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2012*, pages 524–530, 2012.
- [OMSM03] Clark F Olson, Larry H Matthies, Marcel Schoppers, and Mark W Maimone. Rover navigation using stereo ego-motion. *Robotics and*, 43:215–229, 2003.
- [PSBB12] Jae-Han Park, Yong-Deuk Shin, Ji-Hun Bae, and Moon-Hong Baeg. Spatial Uncertainty Model for Visual Features Using a Kinect Sensor. *Sensors*, 12(12):8640–8662, 2012.
- [Ric03] Andrew Zisserman Richard Hartley. *Multiple View Geometry*, volume 53. 2003.
- [RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient Variants of the ICP Algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 2001.
- [SBC12] Jürgen Sturm, Wolfram Burgard, and Daniel Cremers. Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark. *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS IROS*, 2012.
- [SC86] Randall C. Smith and Peter Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.
- [SF11] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, 18(4):80–92, 2011.

- [SJP11] Jan Smisek, Michal Jancosek, and Tomas Pajdla. 3D with Kinect. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1154–1160, 2011.
- [Sol07] Joan Solà. Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: A Geometric and Probabilistic Approach. page 199, 2007.
- [WS17] Oliver Wasenmüller and Didier Stricker. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10117 LNCS:34–45, 2017.
- [Yaa01] Thiagalingam Kirubarajan Yaakov Bar-Shalom, X. Rong Li. *Estimation with applications to tracking and navigation*. 2001.
- [YBHH15] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.
- [Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.
- [ZKS14] Ji Zhang, Michael Kaess, and Sanjiv Singh. Real-time Depth Enhanced Monocular Odometry. *IROS*, pages 4973–4980, 2014.

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:

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

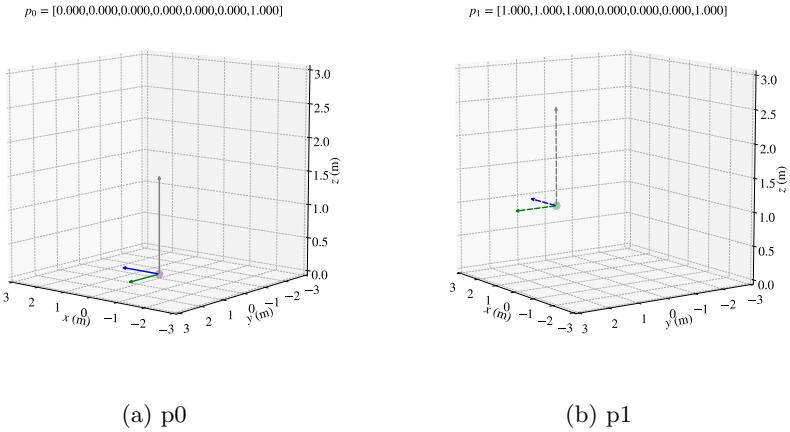


Figure 9.1.1

Rotation with Quaternions

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

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{q}_{01} \otimes \mathbf{q}_0 \\ \mathbf{q}_1 &= [\mathbf{q}_{01}]_L \mathbf{q}_0 \end{aligned} \quad \left[\begin{array}{l} q_w^{01} q_w^0 - q_x^{01} q_x^0 - q_y^{01} q_y^0 - q_z^{01} q_z^0 \\ q_w^{01} q_x^0 + q_x^{01} q_w^0 + q_y^{01} q_z^0 - q_z^{01} q_y^0 \\ q_w^{01} q_y^0 - q_x^{01} q_z^0 + q_y^{01} q_w^0 + q_z^{01} q_x^0 \\ q_w^{01} q_z^0 + q_x^{01} q_y^0 - q_y^{01} q_x^0 + q_z^{01} q_w^0 \end{array} \right] = \left[\begin{array}{l} q_w^{01} - q_x^{01} - q_y^{01} - q_z^{01} \\ q_w^{01} + q_x^{01} + q_y^{01} - q_z^{01} \\ q_w^{01} - q_x^{01} + q_y^{01} + q_z^{01} \\ q_w^{01} + q_x^{01} - q_y^{01} + q_z^{01} \end{array} \right] \left[\begin{array}{l} q_w^0 \\ q_x^0 \\ q_y^0 \\ q_z^0 \end{array} \right] \quad (9.2) \right]$$

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

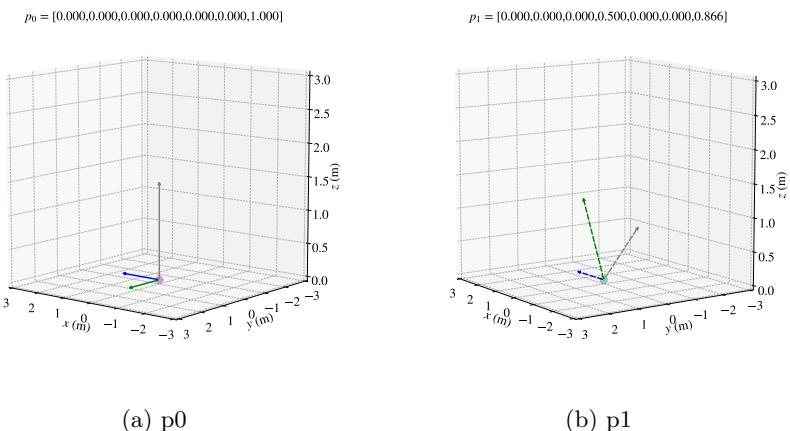


Figure 9.1.2

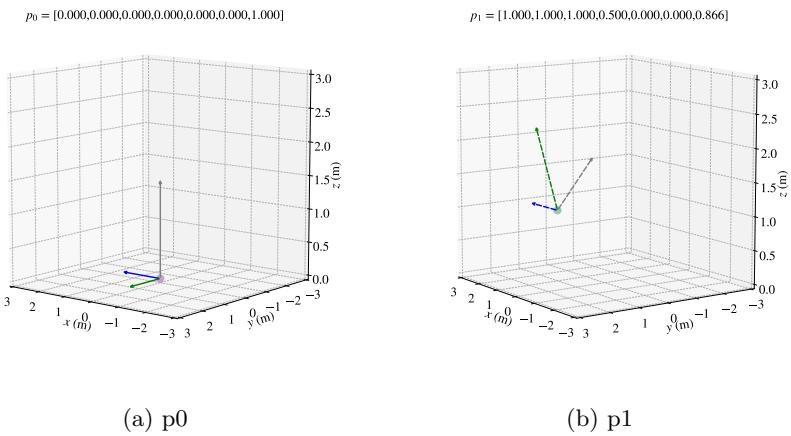
Transformation

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

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

$$\mathbf{q}_1 = \mathbf{q}_{01} \otimes \mathbf{q}_0 \quad (9.4)$$

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



9.2 Least Squares

9.2 Least Squares

Throughout this thesis, least squares method empowered many different components of our 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-linearity 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.

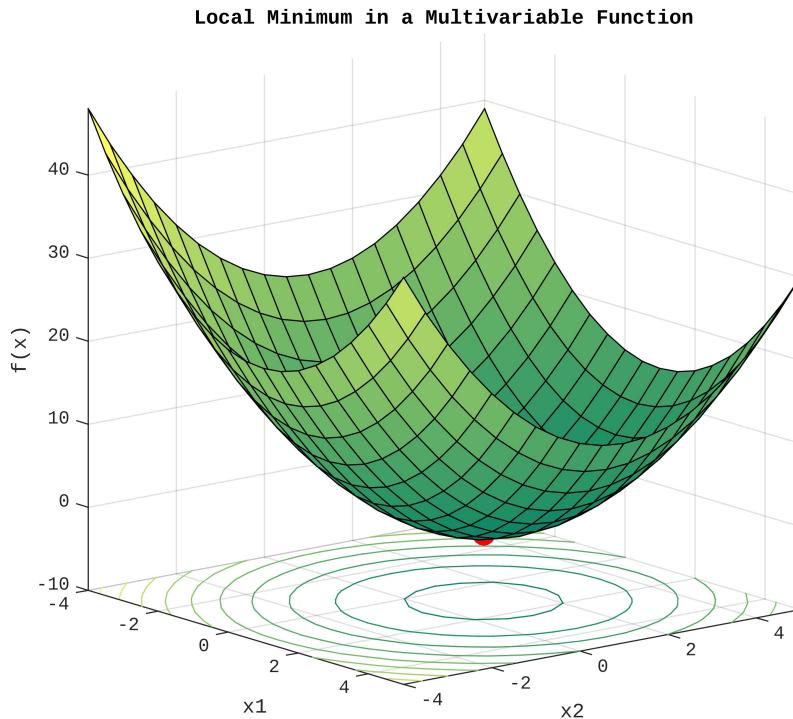


Figure 9.2.1: Local Minimum at a Quadratic Function

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 explanations.

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

$$\eta = g(\mathbf{x}; \boldsymbol{\alpha}).$$

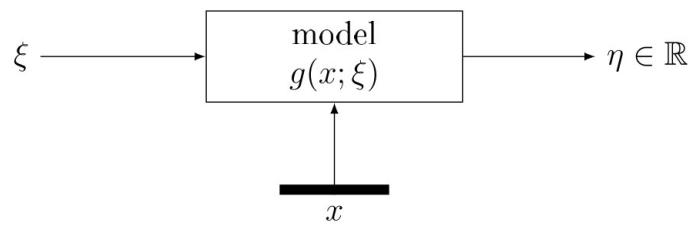


Figure 9.2.2: Least Square Model

$$\xi = a \text{ (independent variable)} \in \mathbb{R} \quad (9.5)$$

$$\eta = S \text{ (dependent variable)} \in \mathbb{R} \quad (9.6)$$

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

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

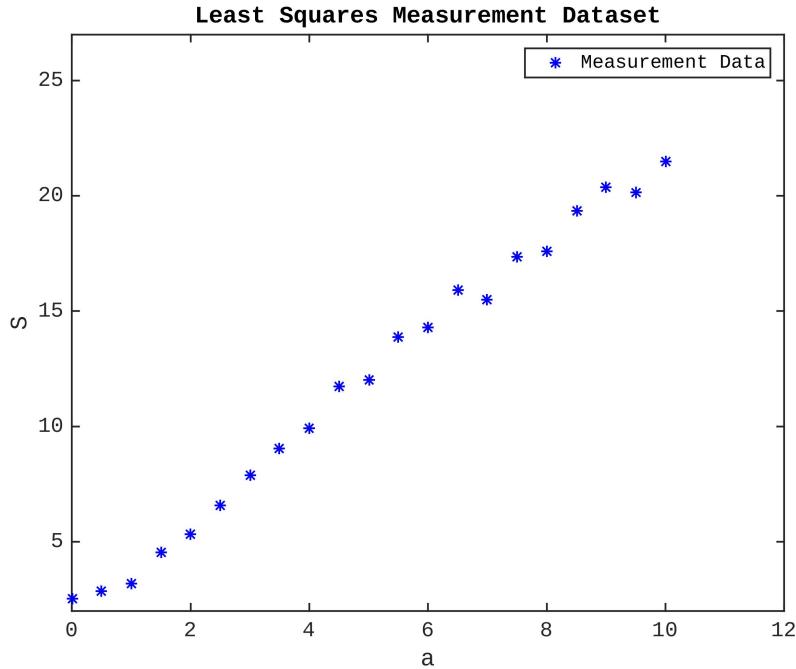


Figure 9.2.3: Least Squares Measurements

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 $g(\mathbf{x}; a_i)$ 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.

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

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

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

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

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

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

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

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:

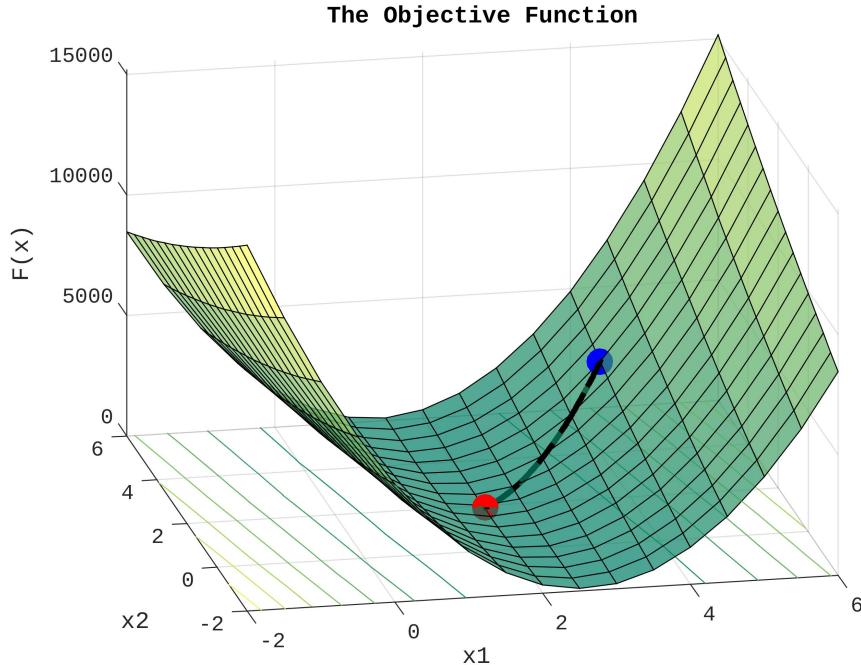


Figure 9.2.4: Local Minimum at Sum of Squared Error Function

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.

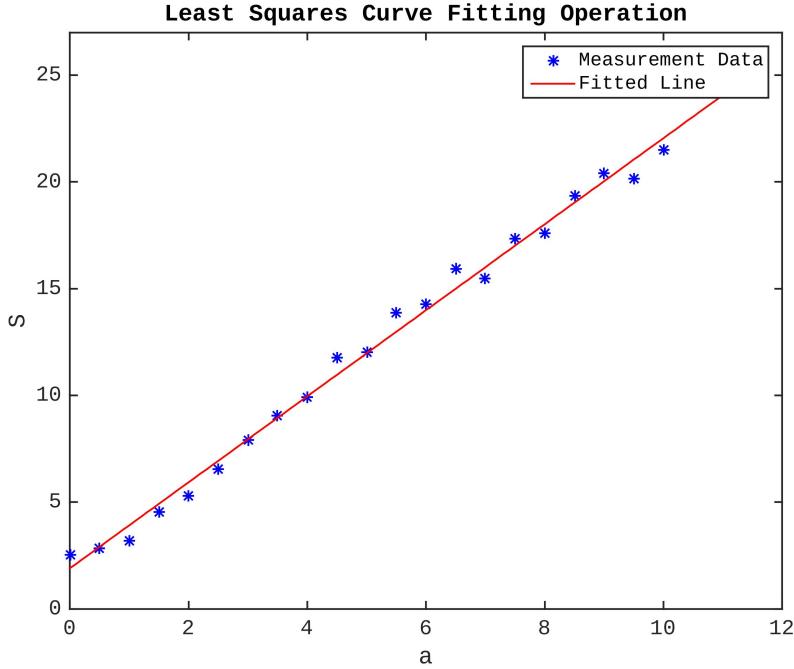


Figure 9.2.5: Least Squares Curve Fitting Operation

Now that we have an intuition 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, minimum 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 $F(\mathbf{x}) = \frac{1}{2}\mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}) = \frac{1}{2}\sum_i f_i(\mathbf{x})^2$ where it has $\mathbf{x} = [x_1, \dots, x_n] \in R^n$ 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:

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

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

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

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

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

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:

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

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

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

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

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

Given 9.2.2 and 9.2.2, we elaborate the quadratic model:

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

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

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

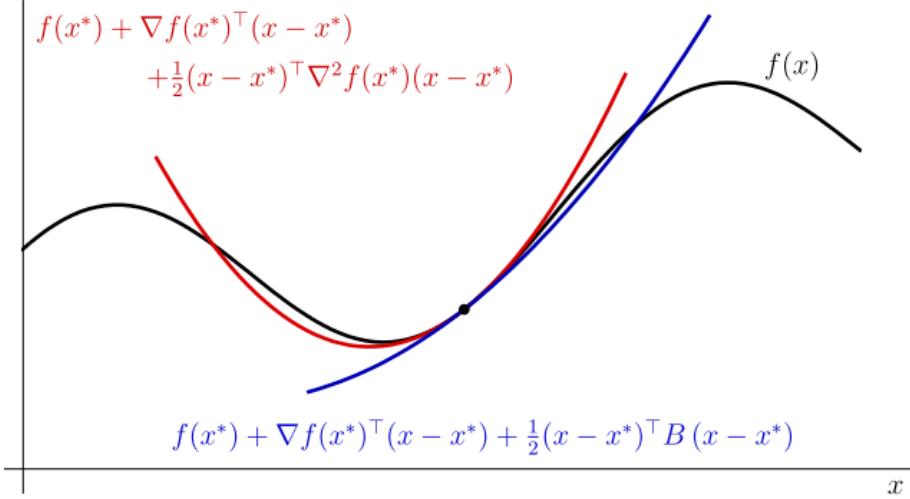


FIGURE 21.2. A function $f : \mathbb{R} \rightarrow \mathbb{R}$ and its quadratic Taylor polynomial $T_2(x; x^*)$ (in red) about the point x^* . We also show another quadratic model of the function (blue), whose second derivative (Hessian) B does not coincide with $\nabla^2 f(x^*)$. It can be seen that 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 $\nabla^2 f(x^*)$) 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 computationally 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:

$$\mathbf{B}_{\text{LM}}^n = \mathbf{J}(\mathbf{x}^n)^T \mathbf{J}(\mathbf{x}^n) + \lambda^n \mathbf{I} \quad (9.24)$$

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

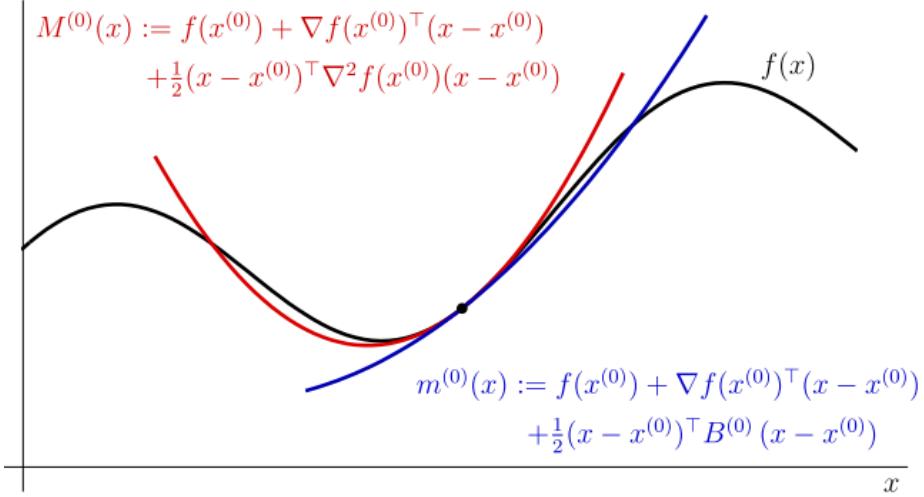


FIGURE 3.1. A function $f : \mathbb{R} \rightarrow \mathbb{R}$ and its quadratic Taylor polynomial $M^{(0)}(x)$ (in red) about the point $x^{(0)}$. We also show a simplified quadratic model $m^{(0)}(x)$ of the function (in blue) whose second-order derivative $B^{(0)}$ does not coincide with $\nabla^2 f(x^{(0)})$. 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:

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

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

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

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:

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

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

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

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:

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

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 factor 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.

Fundamentally, 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 $q_{LM}^n(\Delta \mathbf{x}^n)$ 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 Error Propagation Law

In state estimation applications, sensor measurements and their uncertainty are usually fused in order to keep the uncertainty 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 probability distribution function of X , $p(X)$, to the probability 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.

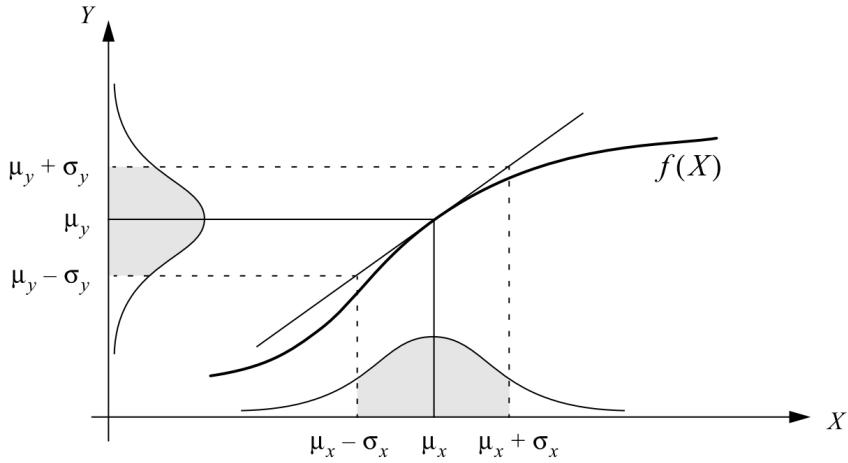


Figure 9.3.1: Error Propagation [AA98]

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:

$$Y \approx f(\mu_x) + \frac{\partial f}{\partial X} \Big|_{X=\mu_x} (X - \mu_x) \quad (9.30)$$

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

$$\mu_y = f(\mu_x) \quad (9.31)$$

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

$$\sigma_y = \left. \frac{\partial f}{\partial X} \right|_{X=\mu_x} \sigma_x \quad (9.32)$$

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

$$\begin{aligned}\mathbf{Y} &\approx f(\mu_1, \dots, \mu_n) + \sum_{i=1}^n \left[\frac{\partial f}{\partial X_i}(\mu_1, \dots, \mu_n) \right] [X_i - \mu_i] \\ \mathbf{Y} &\approx f(\mathbf{X}_{\mu_x}) + \mathbf{J}(\mathbf{X}_{\mu_x})(\mathbf{X} - \mathbf{X}_{\mu_x})\end{aligned}\quad (9.33)$$

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

$$\mathbf{Q}_y = \mathbf{J}(\mathbf{X}_{\mu_x})^T \mathbf{Q}_x \mathbf{J}(\mathbf{X}_{\mu_x}) \quad (9.34)$$

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