Visual Odometry for Autonomous Vehicle Navigation

Development of a monocular visual odometry system for ARTEMIPS

Nour SAEED

Academic Supervisors:
Prof. Stefano Alberto MALAN
Prof. Massimiliano GOBBI

Laboratory Supervisors:
Prof. Stephane BAZEILLE
Dr. Jonathan LEDY

A thesis presented for the fulfillment of the requirements for the
Masters of Science in Mechatronics Engineering
at Politecnico di Torino

POLITECNICO
DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica
Politecnico di Torino
Italy

Abstract

For Autonomous Vehicles (AVs) to take over, they must reach a high level of reliability. Different aspects are to be considered in the development of well-performing AVs. An important one is the development of a robust and accurate localization for AVs. Localization is the act of determining the vehicle position with respect to its environment. Cameras are available on almost all AVs and mobile robots, and research shows the benefits using them in the localization process.

Monocular Visual Odometry is the process of determining the position and orientation of a vehicle using one camera. Visual odometry systems are usually based on epipolar geometry which embeds information on the geometric relation between two views of a scene. In the case of planar scenes, such as empty wide roads, the epipolar geometry fails. Homographies which provide the geometric relation between two views of a mostly planar scene can provide better motion estimates in this case.

After a review of the literature and state-of-art of Visual Odometry (VO), a homography-based monocular VO system was developed. This system uses *parallax beams* which is a method that allows to recover the essential matrix without having to discard the previously estimated homography. The developed VO system has been tested on the KITTI dataset and on a custom sequence. The results of these tests show that this system can provide reliable localization.

Acknowledgements

First, I would like to thank my academic supervisor Prof. Stefano Alberto Malan for his constant support during my masters thesis.

I would also like to thank the team at the Université de Haute-Alsace UHA for allowing me to develop my master thesis in their premises. I especially want to thank my UHA supervisors, Prof. Stephane Bazeille and Dr. Jonathan Ledy.

Moreover, I would like to thank the teams of Scribit & Makr Shakr for an amazing year. I would especially like to thank Andrea Bulgarelli for being both a great mentor and friend.

I also thank my friends, from all nationalities, for all the amazing experiences and unforgettable memories.

Last but not least, I would like to thank my family for their never-ending support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the recent years, Autonomous Vehicles (AVs) have started getting more and more attention. It is even expected that the global autonomous vehicle market will reach $557 billion by 2026[1]. This is due to the potential effects AVs can have on our lives. Less traffic congestion and decreased accident rates are only some of these benefits. Autonomous vehicles can also be used in environments which are considered to be a potential threat to humans. The possibilities are endless. But for us to be able to reap the benefits of autonomous vehicles, reliable AV systems must be developed.

In order to develop a well-performing AV, different aspects come in play. An important one is the development of a robust and accurate localization for AVs. Localization is the act of determining the AV's position with respect to its environment. It is usually associated with the question *"Where am I?"* [2]. Localization is considered to be critical as a correct location is required for other tasks like navigation.

Today many different technologies are used in the localization process. One common method is the use of the Global Position System (GPS). While this method can provide good localization in outdoor environments, it can not be used indoors. And even though an accuracy of 1 cm can be achieved (using a Real Time Kinematic GPS), it is not possible in certain environments such as urban environments where building can mask some satellites or underground tunnels and parking areas. Other technologies rely on inertial measurement units (IMUs) or/and wheel speed sensors [3]. However, high-quality IMUs are expensive and wheel odometry measurements

are subject to wheel slippage (which occurs on wet roads for example) that lead to inaccurate localization. LIght Detection And Ranging (LIDAR) and/or RAdio Detection And Ranging (RADAR) are also used in localization. And even though LIDARs and RADARs provide good localization and mapping for AVs, they come with a relatively high price tag.

As humans, we use our vision in order to localize ourselves in a given environment. Considering this fact, cameras can potentially be a cheap and reliable alternative used for localization [4].

1.2 Thesis Objective and Description

The master's thesis was developed in *Institut de Recherche en Informatique, Mathématiques, Automatique et Signal* (IRIMAS) at the Université de Haute Alsace (UHA).

The main objective of this thesis is to develop a system capable of localizing a vehicle and estimating its displacement using a monocular camera system so that it can navigate in case of momentary absence or inaccuracy of GPS signals. This process of estimation is called Monocular Visual Odometry (Monocular VO). VO usually relies on epipolar geometry which provides the geometric relation between two scenes. However, epipolar geometry can be unreliable in the presence of a planar scene.

The idea of this system comes from the presence of planar structures in the environment, such as roads and building surfaces. The system relies on estimating the motion through homographies - a matrix that relates planar surfaces in different image views with each other. However, the inevitable presence of disturbances (a car or pedestrian passing) makes estimation of motion from homography inaccurate. The concept of parallax beam is introduced. Parallax Beams provides the means to convert the homography matrix into the essential matrix which provides a more general relation between different image views.

Chapter 2 presents important computer vision concepts used in this context. It also introduces the standard visual odometry pipeline and the concept of parallax beams. In chapter 3, the datasets and main library used are briefly introduced. Chapters 4 and 5 deal with the methodology, algorithm used, and results. The final chapter is dedicated to drawing conclusions and reflecting on the obtained results.

Chapter 2

Preliminaries

This chapter will introduce the topic of visual odometry. It will first introduce the topics of image formation, the pin-hole camera model, and camera calibration and image geometry. It will then go through the visual odometry pipeline. It will also present the concept of parallax beam, a way to estimate the position of the epipole.

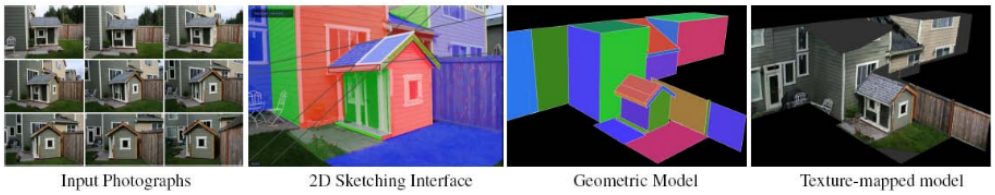
2.1 Computer Vision Concepts

As humans, we perceive the world visually with ease. Using visual cues such as light patterns and shadings on a surface, we are able to tell shapes and reconstruct a three dimensional image in our heads. Looking at someone's photos, we are able guess their emotions from their facial expressions. Computer Vision (CV) is the science that deals with the extraction of data from images or videos. CV includes image processing, scene reconstruction, object recognition, pose estimation ... (see Figure 2.1)

This section will introduce the basics of image formation and the well-known pinhole camera model. It will also talk about camera calibration.

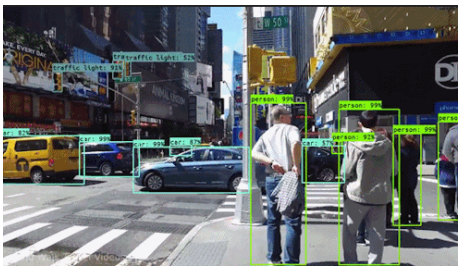
2.1.1 Image Formation and Camera Model

All computer vision applications start with images taken from a camera. So, it is essential to understand how these images are formed. In the Figure 2.2a, the *pinhole camera model* is shown.



(a) 3D reconstruction, source: Computer Vision: Algorithms and Applications

(b) Image processing, source: Digital Image Processing - Dr.Siddiqui



(c) Object detection, source: Towards Deep Learning

Figure 2.1: Computer Vision Examples

The pinhole camera model is based on the pinhole camera which relies on a small aperture, called pin-hole, in which light from a scene passes and projects an inverted image on an image plane (see Figure 2.2a). Let f, the focal length, be the distance between the pinhole and the image plane. For simplicity, it is assumed that there is a virtual image plane (yellow in Figure 2.2b) at a distance f in front of the pinhole ($\mathcal{C}$ in Figure 2.2b). This model is also called the the central projection model.

Consider an arbitrary point P of coordinates (X, Y, Z). Projecting this point on the image plane yields

$$p = (\frac{f}{Z}X, \quad \frac{f}{Z}Y)$$

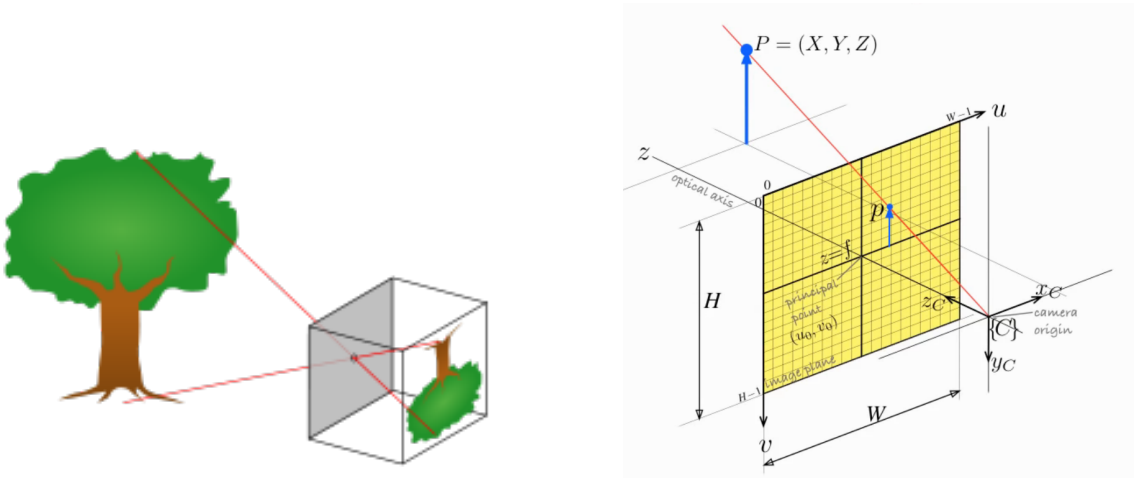
To convert into a projective geometry framework, point p can be reformulated using homogeneous coordinates as

$$\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})^T = (fX, fY, Z)^T$$

or in matrix form as

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.1}$$

In digital cameras, the image falls on an array of pixels. The origin of the pixel coordinate system is located on the top left corner of the image plane and is directed as shown in Figure 2.2b. The next step is to convert the position of p on the image panel from meters to its corresponding pixel position. By dividing each axis with the corresponding pixel dimension along that axis, the coordinate are transformed



(a) Diagram (b) Model, source: ENB339 P. Corke

Figure 2.2: Pinhole Camera Model

from meters to pixels. Moving also the coordinate system to the the top left corner, the transformation becomes

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \tag{2.2}$$

$$u = \tilde{u}/\tilde{w} \qquad v = \tilde{v}/\tilde{w}$$

where ρ_u and ρ_v denote the length of the pixel along each axis and (u_0, v_0) is the principal point - the point at which the optical axis intercepts the image plane.

Plugging the equation 2.1 in equation 2.2 yields

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \frac{f}{\rho_u} & 0 & u_0 \\ 0 & \frac{f}{\rho_u} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \mathbf{K}P$$

K is called the intrinsic camera matrix.

In many cases, it is desired to describe the world coordinates with respect to a camera reference system instead of the world reference system or vise-versa. An example of this is trying to find the position of an agent relative to the given point but not relative to the camera. A rigid body transformation is applied in this case. (see Figure 2.3)

Let R and t respectively be the rotation matrix and translation vector that transform the system from the world coordinate system to that of the camera. Let

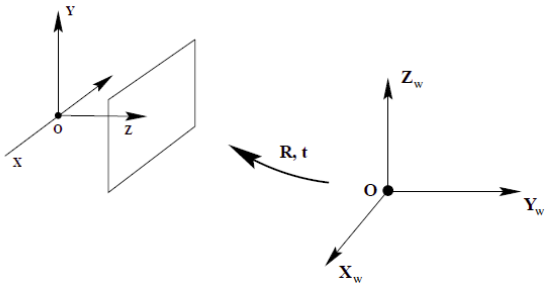


Figure 2.3: Euclidean Transformation

T be the 3×4 transformation matrix defined as

$$\mathbf{T} = \begin{pmatrix} R & t \end{pmatrix}$$

Considering an arbitrary world point $Q(X_w, Y_w, Z_w)$, the overall transformation becomes

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \frac{f}{\rho_u} & 0 & u_0 \\ 0 & \frac{f}{\rho_u} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{T} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \mathbf{K}\ \mathbf{T}\ \tilde{Q}$$

where $\tilde{Q}$ represents the homogeneous coordinates of Q. Note that $\frac{f}{\rho_u} = f_x$ and $\frac{f}{\rho_v} = f_y$ is also used. f_x and f_y represent the focal length of the lens along each axis respectively.

2.1.2 Camera Calibration

The parameters of matrix **K** are called intrinsic parameters because they describe values related to a certain camera like focal length and principal point whereas the values of $\mathbf{T} = (R|t)$ are called extrinsic parameters since they provide coordinate system transformations from 3D world coordinates to 3D camera coordinates. In other words, they provide the camera center position and heading information. The exact measurement of both the intrinsic and extrinsic parameters is difficult.

Also another problem with using a pinhole is that we need a small hole in order to get better images. But this leads to darker images due to less light entering. One way to get around this is to use a parabolic lens. Geometric properties of parabolas make them good replacements for pinholes as they allow the concentration of light in one point. But considering the difficulties in manufacturing, lenses are actually more spherical and are not always centered exactly parallel to the image plane. The former leads to a radial distortion (see Figure 2.4a) of the image whereas the latter leads to tangential distortion (see Figure 2.4b).

(a) Radial Distortion (b) Tangential Distortion

Figure 2.4: Camera distortions, source: AIShack

Camera calibration is proposed as a solution for these problems. Camera Calibration [5] allows us to estimate the focal lengths, principal points and pose of camera relative to the real world. Calibration algorithms also take into account the distortion and applies the corresponding correction.

For the radial factor, one uses the following formula:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

where k_1, k_2, k_3 are the radial distortion coefficients and $r^2 = x^2 + y^2$

Tangential distortion can be corrected using the following formulas:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

where p_1 and p_2 are the tangential distortion coefficients and $r^2 = x^2 + y^2$

Several algorithms exist to solve the problem of estimating all these parameters [5, 6, 7].

A standard way to calibrate a camera is to use a chessboard pattern. The computer vision library OpenCV provides built-in functions that allow us to calibrate cameras using standard shapes (chessboards, circles, etc.). A chessboard has certain characteristics that makes it interesting for calibration:

- It has a planar shapes which allow easier calculations.

- Points can exist on this plane.
- These points can be easily extracted considering the geometry and color of the chessboard squares.
- These points are located on sets of straight lines.

Several pictures of the chessboard in different orientations are taken and the extracted points are used to estimate the camera parameters and distortion parameters. The figure 2.5 shows an example of an image before and after calibration.

(a) Original (b) Undistorted (c) Undistorted Optimized

Figure 2.5: Camera Calibration

Camera calibration plays an important step as after calibrating the camera, measurements can be done using the camera and transformation from pixel coordinates to real-world coordinates are made possible.

2.2 Image Geometry

This section will deal with the image geometry, more specifically epipolar geometry and the concept of homographies.

2.2.1 Epipolar Geometry

One important concept in the study of visual odometry is that of the epipolar geometry. In multiple view geometry, relationships exist between these different views. These relationships encode some useful information.

Consider two images (or views) of a scene taken by a pair of camera, or by the same camera but from two different locations. See Figure 2.6.

The points O_c and O_p denote the optical centers of the camera(s) at each view, whereas π_c and π_p represent the image planes. The line O_cO_p is called the baseline.

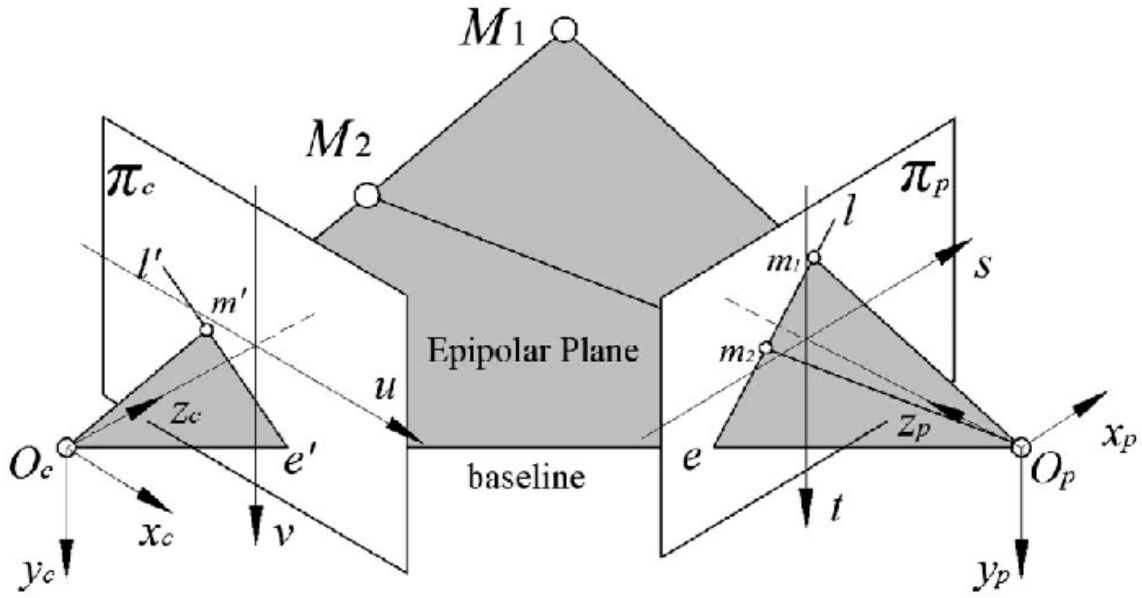


Figure 2.6: Epipolar Geometry

Consider now the point M_1. m' and m_1 denote the projections of this point on planes π_c and π_p respectively. The plane defined by points M_1, O_c, and O_p form what is called the epipolar plane. The intersections of this plane with the image planes are called the epipolar lines.

The epipolar lines encode information on where the matches of a certain point in another image are. For example consider point m'. This point can be the projection of either M_1, M_2 or any point on the line joining the two points. Then, its corresponding match on the second image plane can be anywhere on the line l, the epipolar line.

A relative transformation exists between the camera in view 1 and the camera in view 2. Let R be the rotation and $t = \overrightarrow{O_c O_p}$ be the translation. Also, let us denote $p_c = \overrightarrow{O_c m'}$ and $p_p = \overrightarrow{O_p m_1}$. Thus, $R\ p_p$ is the vector p_p in the basis of p_c. Taking into consideration that p_p, p_c and t are co-planar, we can write that

$$p_c^T\ [t \times R p_c] = 0$$

This can be rewritten as

$$p_c^T\ [\hat{t}\ R]\ p_p = p_c^T\ E\ p_p = 0 \tag{2.3}$$

where

$$\hat{t} = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}$$

The equation 2.3 is called the epipolar constraint. The matrix E is called the essential matrix. The essential matrix is independent of the scene structure and only

relies on image correspondences. It is possible to estimate this matrix from a set of corresponding image points. Several algorithms exist to estimate the essential matrix of which the most common is the eight-point algorithm.

To generalize even more, the assumption of an uncalibrated camera is made. Let $p = K^{-1}q$, where p is the calibrated point, K is the unknown calibration matrix and q is the un-calibrated point coordinate. The epipolar constraint can be rewritten as

$$q_c^T \; K_c^{-T} \; [\hat{t} \; R] \; K_p^{-1} \; q_p = q_c^T \; F \; q_p = 0 \tag{2.4}$$

$$F = K_c^{-T} \; E \; K_p^{-1} \tag{2.5}$$

where F is called the fundamental matrix and the subscripts c and p denote the different camera views. F can be also estimated. Moreover, apart from being independent of the scene structure, the fundamental matrix does not require knowledge of the camera's intrinsic parameters.

2.2.2 Homography

Consider a planar structure. Two images of the planar structure are taken from different views. With the assumption that we have a pinhole camera model, two images are related by what is called a *homography*.

$$m^{'} = Hm \tag{2.6}$$

where $m^{'}$ and m are a corresponding points pair belonging to the planar structure and H is the homography matrix.

The equation 2.6 can be expanded to

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.7}$$

The homography parameters can be estimated. One famous method is the Direct Linear Transform (DLT). Dividing the first row of equation 2.7 by the third row and the second row by the third row, we get the following two equations

$$-h_1x - h_2y - h_3 + h_7xx' + h_8yx' + h_9x' = 0$$
$$-h_4x - h_5y - h_6 + h_7xy' + h_8yy' + h_9y' = 0$$

To be able to estimate the homography, 4 point pairs are needed. That gives us 8 equations. One degree of freedom is lost due to the assumption that no 3 points

are to be co-linear. In practice, there will always be uncertainty due to difficulty in finding exact correspondences. Algorithms like RANSAC (see Chapter 2.3.5) add robustness to the homography estimation procedure.

2.3 Visual Odometry

Visual Odometry (VO) is the process of estimating the ego-motion of a robot, vehicle or any other agent using as input one or more cameras attached to the given agent. Its application domain are, but not limited to, robotics, Augmented Reality (AR) and Autonomous Driving (AD).

In various driving conditions (e.g. braking and presence of ice), a vehicle can slip and there becomes a relative motion between the car and the road called wheel slip. VO holds a great advantage compared to wheel-based odometry as VO is not affected by wheel slip. It has been also shown that VO provides more accurate estimates with relative error in position ranging from 0.1% to 2% [8].

Also, considering the fact that cameras are relatively cheap, VO provides a good addition to navigation systems. Moreover, using cameras for motion estimation allows us to integrate this data into other vision-based algorithms like obstacle and lane detection for example, without the need for calibration between sensors.

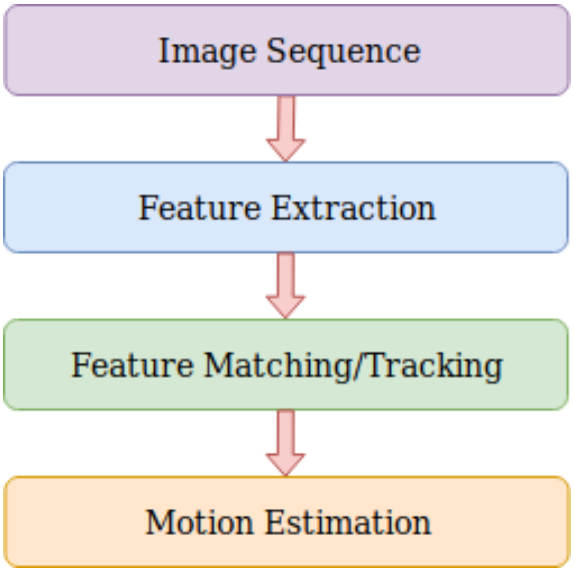


Figure 2.7: Visual Odometry Pipeline (Simplified)

VO is actually a subset of a larger computer vision problem called *Structure from Motion* (SfM). SfM deals with the 3D reconstruction of shapes and camera poses from a set of images. The final structure and poses are then refined with

certain offline optimization processes [9, 10]. VO deals with estimating 3D motion from cameras as new frames arrive and in real time.

VO started in the early 1980s and was done by Moravic [11]. His work demonstrates the first motion-estimation pipeline and one of the first feature extractors. In fact, the fundamental functioning blocks he proposed are still used today.

Figure 2.7 demonstrates the simplified pipeline of a visual odometry system. The general pipeline goes as follows:

1. Starting from an image sequence, an optional first step is to pre-process the images. Depending on the quality of the images provided by the camera and on the possible real-time constraints imposed by the hardware used, several operations can be done on an image. Sharpening, resizing, de-noising, morphology, etc. can be done on the image to improve its quality for the subsequent steps.
2. Image features are extracted.
3. Features between 2 or more consecutive frames are matched OR features are tracked throughout the frame sequence. Thus, a correspondence between image features is found.
4. From these feature correspondences, the motion between the camera pose in an image and that of its subsequent images is estimated.

The following sections deal with the different elements of the VO pipeline.

2.3.1 Feature Extraction

Before talking about feature extraction, it is important to understand what features are. A feature is a distinctive point or zone in an image. What makes features interesting is the possibility of being able to find correspondences between images. Also, considering that many computer vision tasks require finding matching points across several frames or views, having reliable points to be matched is essential. In theory, any point can be a feature. But the problem lies in the selection of good features.

To understand the concept of good and bad features, consider the Figure 2.8. Three patches with different characteristics exist:

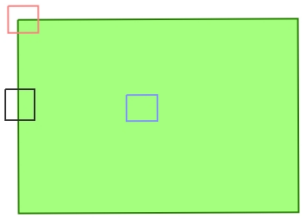


Figure 2.8: Simple Features

- The blue patch: As this patch is moved slightly, the pixel values inside this patch will remain the same. So, possible matches can be any patch inside the rectangle.
- The black patch: Change is noticed as this patch moves along the horizontal axis - or along the gradient. However, if it moves along the vertical axis, no change will be detected.
- The red patch: The pixel value inside the patch changes regardless of how the patch moves. This is an indication that it is unique.

Nonetheless, all the patches represent possible features - some (like that of the red patch) are better then others (blue patch).

So, intuitively speaking, good features can be considered as regions in images which have maximum variation when moved by small amounts. There exists many algorithms which deal with the features extraction process. There are mainly two different types of feature extractors: corner detectors (whose concept is similar to the one demonstrated in the previous paragraph) and blob detectors. A blob is a region of interest in an image which has different properties from its adjacent regions.

Listed below are some of the main feature extractors.

Harris Corner Detector

One of the first feature extractors is the Harris Corner Detector. Its functionality relies on the idea that the intensity of pixels, in a grayscale image, around a corner change dramatically. Let $I(x, y)$ be the intensity of an image point (x, y). Considering a small displacement (u, v) in the x and y direction respectively, the squared difference over a patch is:

$$\sum_{patch} [I(x + u, y + v) - I(x, y)] \tag{2.8}$$

In order to find out where the intensity is changing the most, we have to maximize the above equation. Applying the 1st order Taylor Expansion to the equation 2.8 and reforming yields:

$$\begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix} \mathbf{M} \begin{pmatrix} u \\ v \end{pmatrix}$$

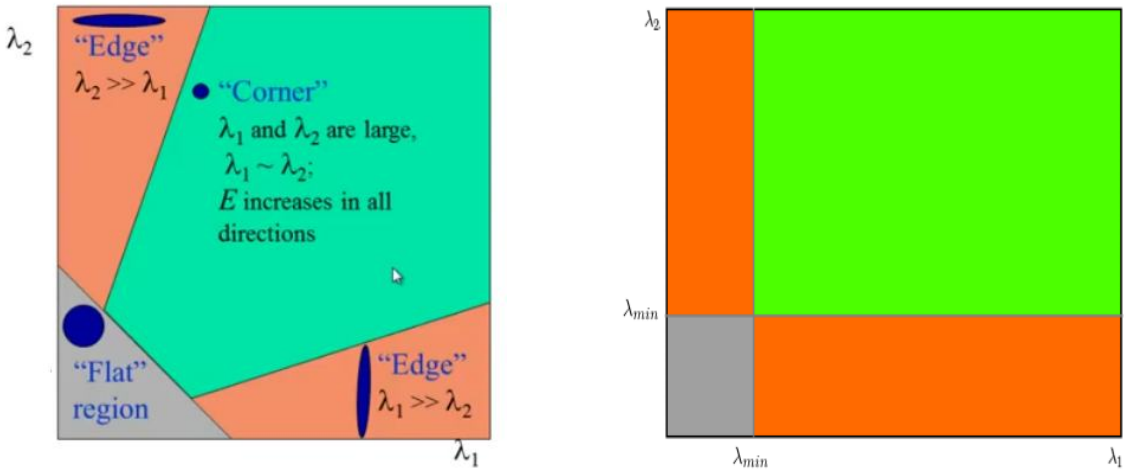
where I_x and I_y are the derivatives of the intensity along x and y, and $\mathbf{M}$ is the Hessian Matrix.

By considering the eigenvalues of $\mathbf{M}$, one can get an intuition whether the feature is a corner, edge, or flat surface. If both eigenvalues λ_1 and λ_2 are small, then the intensity is not changing and the region can be considered flat. In case one eigenvalue is significantly bigger than the other, then the intensity is changing in one direction only so there is an edge at that point. If both eigenvalues are large, then there is an intensity change in both directions and the point is a corner. Harris introduced the following score:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 = det(M) - k \cdot trace^2(M) \tag{2.9}$$

Where k is an empirical number usually between 0.04 and 0.06. The algorithm can be speeded up by directly considering the Hessian matrix without the need to calculate the eigenvalues as seen in equation 2.9.

Based on the comparison of the score of equation 2.9 with a threshold, it can be determined if a point is a corner or not. Figure 2.9a shows the eigenvalue space for for the Harris corner detector.



(a) Harris (b) Shi-Tomasi

Figure 2.9: Harris and Shi-Tomasi $\lambda_1 - \lambda_2$ space, source: OpenCV

Shi-Tomasi Corner Detector

This feature extractor is based on the Harris corner detector above. It works in exactly the same way with a slight difference – the score. In Harris corner detector, a score is calculated for each pixel. This score in a way is calculated from the eigenvalues of the Hessian matrix. Shi and Tomasi suggested a new score:

$$R = min(\lambda_1, \lambda_2)$$

Figure 2.9b demonstrates the eigenvalue space of this score. This score provides better result for corner detection in some cases. Nonetheless, the need of the eigenvalues requires the decomposition of the Hessian matrix $\mathbf{M}$, which in turn leads to an increased computation time.

SIFT

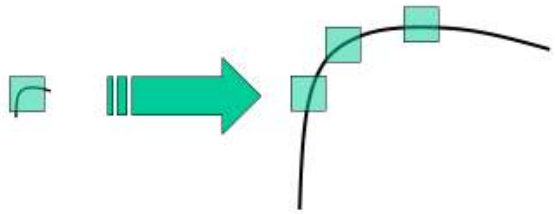


Figure 2.10: Scaling of a corner

When the size of the corner is less than that of the patch, the corner can be detected. Scaling the corner can lead to problems in finding that corner. See Figure 2.10. Scale Invariant Feature Transform (SIFT) is a feature extractor that helps solve this problem. SIFT-extracted features are invariant to scaling, rotations, changes in 3D view point, noise and change in illumination.

SIFT first of all constructs a scale space for an image. The procedure is as follows:

1. Gaussian blur is applied to the image a certain number of times. This number is called scale.

2. The image is resized (usually by half).

3. Steps 1 and 2 are repeated for a fixed number of times called octave.

Figure 2.11 shows the generated scale space of an image with 3 scales and 4 octaves. Note that the images were not resized in half here. The number of scales depends on the original size of the image. Usually, 4 octaves and 5 blur scales are used.

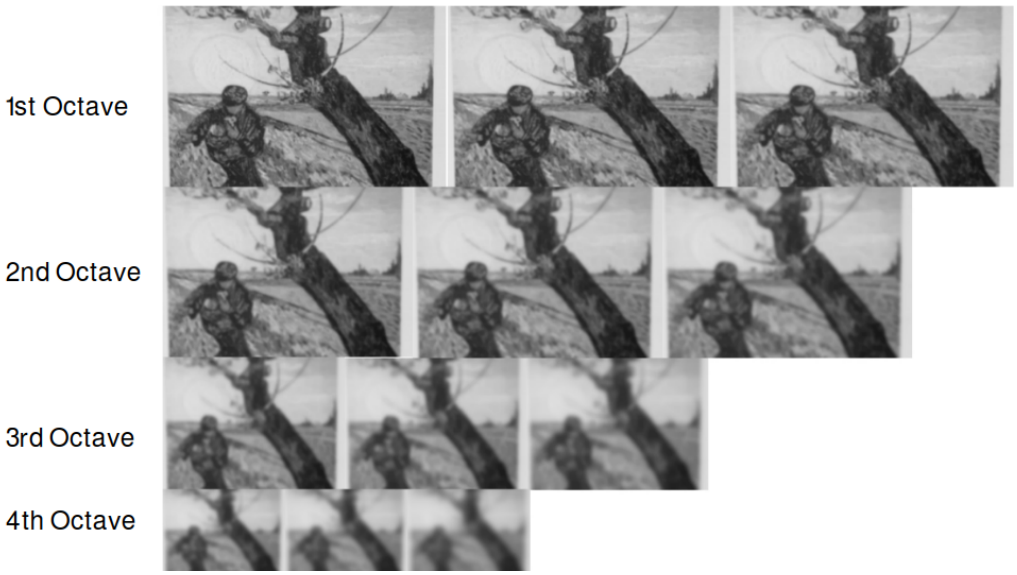


Figure 2.11: Scale Space

The next step is finding the key points. The Laplacian of Gaussian (LoG) is a great way to find interesting points in an image, but it is computationally heavy. One way to get around that is to use the Difference of Gaussian (DoG) (see Figure 2.12) instead. The maxima and minima in the DoG are located. This is simply done by iterating through each pixel and checking all of its neighbors. Figure 2.13 shows the pixel of interest (X) and its neighboring pixels (green circles). After that the extrema are refined and through the Taylor expansion, the sub-pixel maxima and minima are located.

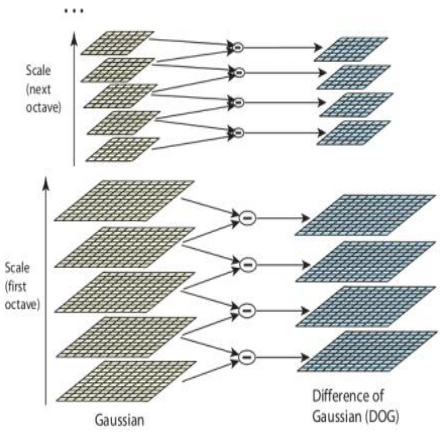


Figure 2.12: DoG

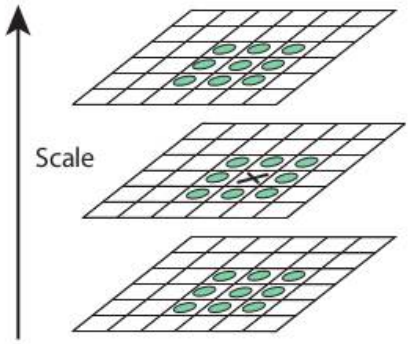


Figure 2.13: Local Extrema

Points with low contrast are then discarded by considering the intensity at a current pixel of the DoG and comparing it with a certain threshold. Also edges are removed by using the Hessian Matrix as it was used in the Harris Corner Detector.

Many algorithms for feature extraction exist. Other examples of feature detectors are Speeded-Up Robust Features (SURF), and Features from Accelerated Segment

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figure 2.14: Comparison of some feature detectors [12]

Test (FAST).

Figure 2.14 shows a comparison between some of the common feature detectors based on some properties and performance indicators.

When choosing a feature extractor, several aspects need to be taken into consideration such as robustness and computational cost.

2.3.2 Feature Description and Matching

After extracting features, it is essential find the same features in the following frames. One way to do so is to match features of two corresponding frames which have similar characteristic. To find these *similar characteristics*, we need to define them first. Here is where feature description comes in play.

Feature description is a process of providing a feature with *numerical fingerprint* that can be used to differentiate one feature from another. One of the simplest descriptors is to consider a simple window around the feature, with the pixel values inside this window forming the so-called *fingerprint*. This kind of description is not efficient since it is not scale nor rotation invariant.

For a descriptor to be considered "good", some requirements are desirable of which (1) translation and rotation invariance, (2) scale invariance, (3) illumination and blur invariance, and (4) low memory requirements. Usually, it is difficult to satisfy all these requirements and a trade-off occurs between robustness (requirements 1,2,3) and computational time (requirement 4).

Now consider 2 images each with enough overlap. The features are extracted and described. We want to find matches in the features between these two set of features. A match occurs when, with respect to a certain measure, two features from two corresponding images are considered similar.

One well-known matcher is the *Brute Force* (BF) matcher. The BF matcher takes in the descriptor of one feature from one image and calculates the distance between it and each descriptor of the features of the other image. This is done to all features. The feature-pairs that correspond to the minimum distance form the set of matched features. In Figure 2.15, matches between features of two images can be seen in blue.

Figure 2.15: Brute-Force Matching

Some of the most common norms used to calculate the distance between descriptors are:

- L1 Norm $\sum_I |\texttt{des1}(I) - \texttt{des2}(I)|$
- L2 Norm $\sqrt{\sum_I (\texttt{des1}(I) - \texttt{des2}(I))^2}$
- Hamming Distance $des1 \bigoplus des2$

The OpenCV documentation section on NormType provides a list of the most frequently used norms. The selection of the norm depends on the descriptor used. For example, with a binary descriptor like BRIEF, using a hamming distance norm makes more sense than using a regular L2 norm.

Another popular matcher is the Fast Library for Approximate Nearest Neighbors (FLANN) based matcher [13]. This algorithm is based on the K-D trees algorithm and allows for faster matching processes especially with large amounts of descriptors. Both matchers rely on the concept of K-Nearest Neighbors (KNN) in the calculation.

The output of the matchers almost always contains false matches. One way to filter out some of the bad matches is to use *Lowe's Ratio Test*.

In Figure 2.16, the solid line shows the Probability Density Function (PDF) of the ratio of the nearest and 2^{nd} nearest matches for correct matches, while the

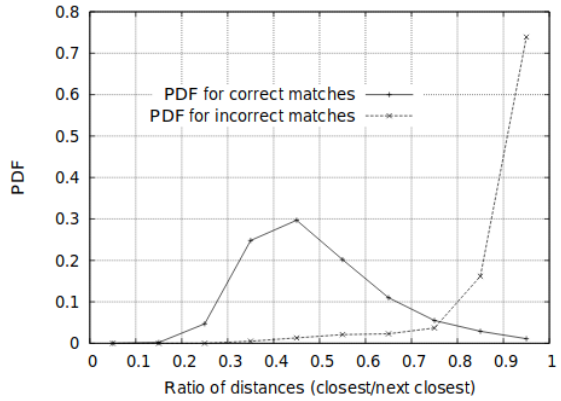


Figure 2.16: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest.

dotted line is the PDF for matches that were incorrect. Lowe's Ratio Test considers the KNN with $k = 2$. It states that if the ratio of the distance of the closest to that of the 2^{nd} nearest is less than a certain threshold (usually between 0.7 and 0.8), then we can consider the closest match a good match. If not, then the feature is ambiguous and we discard this match.

2.3.3 Feature Tracking

Aside from feature matching, which as seen in the previous section is about finding correspondences between two images, feature tracking is another way of finding corresponding point pairs in consecutive images.

As the name implies, feature tracking is to follow a feature as it moves in a sequence of images. One important point to take into consideration when using tracking is that the scene (between two images) must not change a lot. One famous feature tracker is the Lucas-Kanade tracker.

For successful tracking, three assumptions must be made: (1) motion is small such that points do not dramatically change position, (2) similar intensity in each frame, (3) spatial coherence exists so, for example, points close to each other make the same movement.

Starting from feature points, the motion of point between consecutive frames is computed. In case we are interested in translation, the optical flow approach is used. Optical flow is used for many application like SfM, video stabilization, and video compression. For other more complex transformations such as scaling, a local affine transformation approach is used.

We are mainly interested in locating the position of the feature in the following frame, so the optical flow approach is better.

Optical flow was introduced by *Horn & Schunck* and is based on the assumption of intensity consistency. Let $I(u, v, t)$ be the intensity of pixel (u, v) at time t. Assuming that after time dt the pixel displaces by (du, dv) and taking into consideration the previously stated assumption, we can write:

$$I(u, v, t) = I(u + du, v + dv, t + dt) \tag{2.10}$$

Applying Taylor expansion to the right-hand side of the equation 2.10, it becomes:

$$f_x\dot{u} + f_y\dot{v} + f_t = 0 \tag{2.11}$$

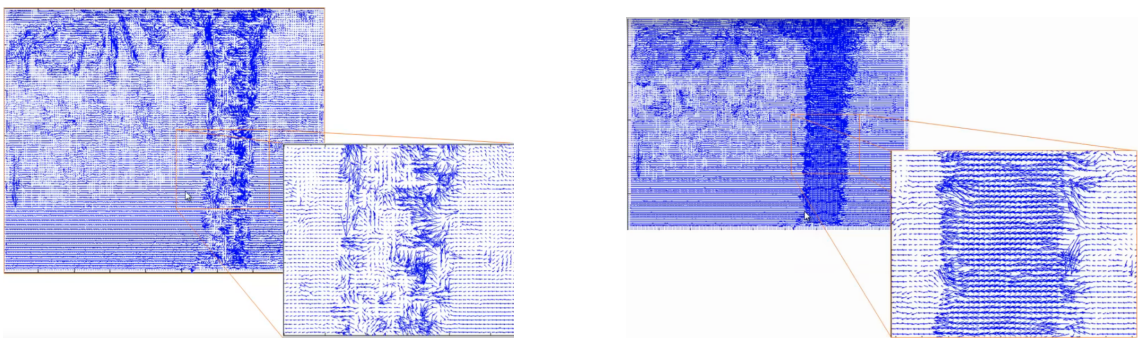
where $f_x = \frac{\partial I}{\partial x}$, $f_y = \frac{\partial I}{\partial y}$, $f_t = \frac{\partial I}{\partial t}$, $\dot{u} = \frac{\partial u}{\partial t}$, and $\dot{v} = \frac{\partial v}{\partial t}$.

The equation 2.11 is called the optical flow equation. There being two unknowns and one equation, this equation can not be directly solved. Several methods exist, of which we will talk about the Lucas-Kanade solution.

Let us take a 3x3 window around the point of interest. Considering the spatial coherence assumption, all the 9 points of this window have similar motion. The gradients f_x, f_y, and f_t for these points can be found. The problem can be then transformed into a least squares fit problem and the solution becomes:

$$u = \frac{-\sum f_{yi}^2 \sum f_{xi}f_{ti} + \sum f_{xi}f_{yi} \sum f_{yi}f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi}f_{yi})^2}, \quad v = \frac{-\sum f_{xi}^2 \sum f_{yi}f_{ti} + \sum f_{xi}f_{yi} \sum f_{xi}f_{ti}}{\sum f_{xi}^2 \sum f_{yi}^2 - (\sum f_{xi}f_{yi})^2}$$

This method however works only for small motion. A concept similar to the scale space concept, called pyramids, is used to overcome this problem. With pyramids, small motions are removed and larger motion becomes smaller. Thus it helps add robustness to the tracker. Figure 2.17 shows how adding pyramids can provide smoother results.



(a) without pyramids (b) with pyramids

Figure 2.17: Lucas Kanade Tracker, source: UCF Computer Vision

2.3.4 Motion Estimation

From a set of corresponding pair points of two successive frames, the camera motion between these two frames can be estimated. As stated previously, a transformation can be defined between two similar images. This transformation can be further decomposed into rotation and translation.

Consider two sets of feature points pts_{t-1} and pts_t of two successive images taken at times $t-1$ and t respectively. Since, in our case, the features are from a monocular camera, both feature point sets are 2-dimensional. The dimensions correspond to the pixel coordinate of the feature in the image. The method of motion estimation is called *2-D-to-2-D*.

Two other methods exist. The first one is 3-D-to-3-D in which both feature point sets are 3-dimensional. In this case, a stereo camera system is used in order to triangulate the 3-D coordinates of the points. The other is 3-D-to-2-D. Here, feature points pts_{t-1} are 3-dimensional, whereas pts_t are the projections of the $t-1$ feature points on the image taken at time t.

To estimate the motion between consecutive images using the 2-D-to-2-D method, the Essential Matrix E is used. In E, information about the camera motion are encoded. However, this motion is up to an unknown scale for the translation. E at time t can be expressed as:

$$E_t \simeq \hat{t}_t R_t$$

where $t_t = [t_x, t_y, t_z]^T$ and

$$\hat{t}_t = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}$$

The $\simeq$ denotes that equivalence is up to a scale.

Hence, the essential matrix must be found. This can be done by taking into consideration the epipolar constraint (eq. 2.3).

We can then decompose the essential matrix E into the rotation and translation. For an essential matrix, there are in general 4 possible combinations of R and t

solutions. The four solutions are:

$$R = U(\pm W^T)V^T$$
$$\hat{t} = U(\pm W)SU^T$$

where singular value decomposition of E is $E = USV^T$ and

$$W^T = \begin{pmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A more comprehensive decomposition of the essential matrix into rotation and translation is demonstrated by Nister in [14]. A similar decomposition is also available for the homography matrix [15]. To find the correct combination of R and t, a cheirality check needs to be done. The cheirality check basically means that the triangulated 3D points have positive depth or in other words in front of the camera pose.

A 4×4 transformation matrix T can be built such that

$$T = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

This is the relative transformation between two images, or more accurate, two camera poses. Assuming that the initial pose T_0 is known (which can be defined as a 4×4 identity matrix in case no prior information exists), the updated pose of the camera at time t can be found by the post-multiplication of the estimated pose with the pose of the camera at time $t - 1$.

Note that since the translation is up to a scale, then we can not have a good estimate on the translatory motion of the vehicle. That is the curse of monocular visual odometry - no depth information. However, several techniques exist to address this problem like using information from other sensors, using prior knowledge of the camera position, horizon line estimation, vanishing point estimation, and deep-learning-based methods for depth estimation.

2.3.5 Outlier Removal

Visual Odometry relies mainly on the idea that our features are perfect and the correspondence between two features sets is ideal. However, since this is never the case in real life applications, a way to filter out these false correspondences is needed. False correspondences result in a wrong estimation of the essential matrix and in turn yields wrong rotation and translation estimations.

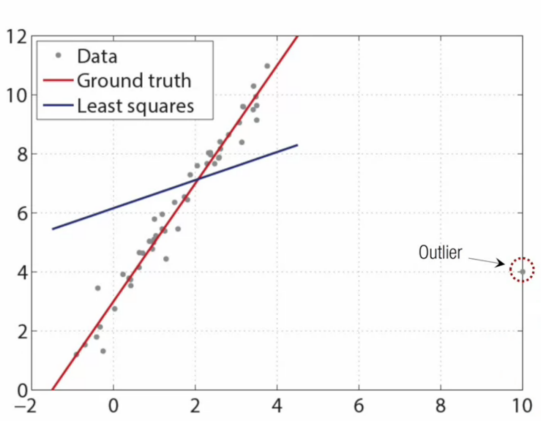


Figure 2.18: 2-D fitting with an outlier

One method used to solve this problem is RANdom SAmple Consensus (RANSAC).

RANSAC is an iterative method used to estimate the parameters of a mathematical model. The algorithm starts out sampling a subset from the data points. The size of this subset depends on what we are trying to fit. If the goal was to fit a line, then the sample size is two points. Three pairs points are needed for an affine transformation and four pairs for homographies.

The next step is to fit a model for the selected data. All the other data points are then tested against the model and number of inliers is found. This is done by comparing the distance between the model and the given data to a predefined threshold. In case of a line, the distance between the line and point can be calculated and compared. In that of an affine transform, the points can be projected using the transformation and then the projections are compared with there corresponding original matches.

The set of inliers obtained is called the *consensus set*. The procedure is repeated for a certain number of times. The model with the highest number of inliers is chosen. Algorithm 1 shows the pseudo-code for a RANSAC for homography.

An important point to take in consideration is the number of samples needed. Let ϵ be the probability that a point is an outlier, s be the sample size, K be the number of samples, and ρ the desired probability that at least one sample has the required sample size.

We can write that $P(inlier) = 1 - \epsilon$. Since s points are needed to build our model, $P(s\ inliers) = (1 - \epsilon)^s$. Then $P(at\ least\ 1\ outlier) = 1 - (1 - \epsilon)^s$. For K samples, it becomes $(1 - (1 - \epsilon)^s)^K$. The compliment of the latter probability corresponds to the probability that there is at least one sample with s inliers. This probability corresponds to the predefined ρ. So, $\rho = (1 - (1 - \epsilon)^s)^K$ which can be

Algorithm 1: RANSAC for homography estimation

```
Result: inliers, best model H
input : Set of points P
H = eye(3,3);
mostInliers = [] ;
for K iterations do
    SubSet = SelectRandomSubsetOf_N_points(P, N = 4) ;
    Hk = ComputeHomography(SubSet);
    inliers = ComputeInliers(Hk,P);
    if inliers ¿ bestNum then
        H = Hi;
        mostInliers = inliers;
    end
end
```

rewritten as

$$K = \frac{log(1-\rho)}{log(1-(1-\epsilon)^s)}$$

The table 2.1 shows how the number of iterations is affected by the inlier probability and the sample size. For estimation purposes, we want to almost always find a good sample - one with the required sample size. Therefore, the desired probability ρ is usually set to be high (99%)

Numer of Iterations Required

	Sample Size (s)					
$1-\epsilon$ (%)	2	3	4	5	6	7
90	3	4	5	6	7	8
80	5	7	9	12	16	20
70	7	11	17	26	37	54
50	17	35	72	146	293	588
30	49	169	567	1893	6315	21055

Table 2.1: Number of iterations required as a function of sample size and inlier probability for $\rho = 99\%$

2.4 IRIMAS Odometry System

One of the most recent research done at IRIMAS and the French-German Research Institute of Saint-Louis (ISL) was on a vision-based epipole estimation method that is robust to nearly planar scenes using parallax beams [16]. This allows the conversion of homographies into essential matrices.

Since co-planar points do not provide enough constraints to help determine the

epipolar geometry [17], the well-known algorithms usually used in estimation like 5-point algorithm and 8-point algorithm do not work well with largely planar scenes. In that case, homographies must be used. However, in the presence of a structured scene, homographies fail.

In this section, the parallax beam method is introduced.

2.4.1 Parallax Beam

In order to estimate the fundamental matrix from image features that are partially co-planar, the plane-and-parallax formulation can be used. Features which belong to the same plane are related to each other by a homography H.

Let x_i and $x_i^{'}$ be corresponding image features from two consecutive images. We have

$$x_i^{'} = Hx_i$$

For the feature pairs that do not satisfy this relationship, they do not belong to the homography and the line joining $x_i^{'}$ and Hx_i is called the parallax line. This line and the epipolar line passing through feature $x_i^{'}$ are theoretically equivalent. See figure 2.19.

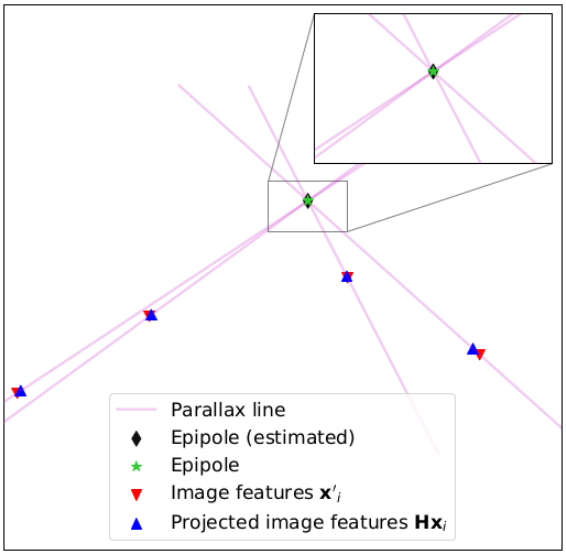


Figure 2.19: Parallax lines and the epipole

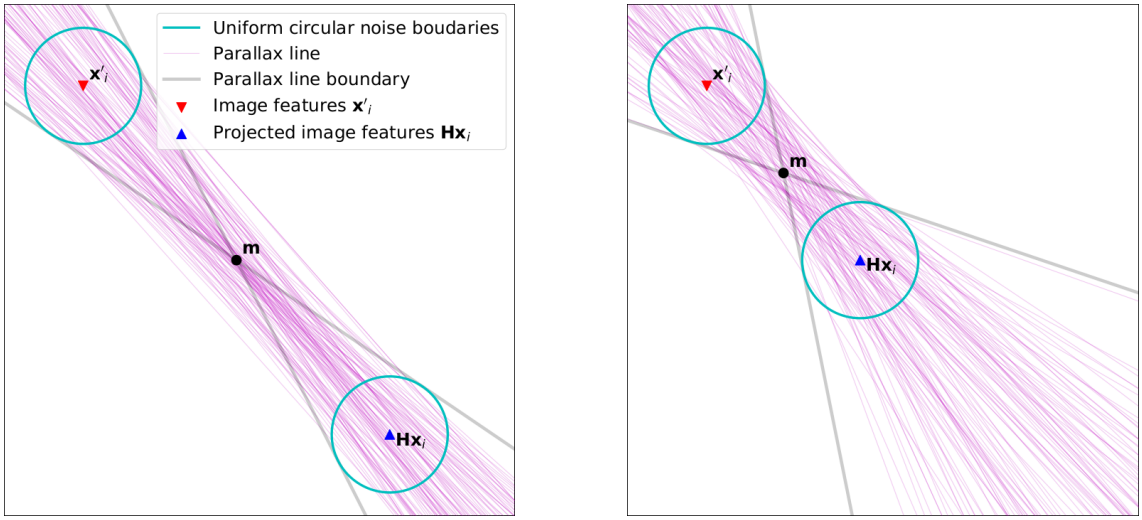
However, due to noise from the camera and imperfections in feature detection and homography estimation, parallax lines do not have a common intersection. One way to overcome this problem is to find the intersection point through which most lines pass (with a certain threshold around this point). This method provides

limitations due to (1) the fact that the epipole might not be estimated correctly and (2) that parallax lines are not equally influenced by noise.

The parallax beam paradigm takes into consideration the noise in the feature positions, the noise in the feature projections by the homography, and that image features close to the homography plane are less reliable than more distant ones.

Due to the noise in the feature position, the true position of the feature can located in the vicinity of the detected feature. Usually, the noise is modelled as Gaussian with a standard deviation of σ. For simplicity, the noise around the point is considered to be uniform circular with a radius r. We take $r = 3\sigma$.

Considering that the images are consecutive, the distortion caused by the homography is small. So, we can consider that the circular noise propagates to the projected feature.



(a) Larger Parallax (b) Small Parallax

Figure 2.20: Set of Parallax lines

The circles around the feature and its homographic projection define a set of possible parallax lines (see Figure 2.20). The orientations of these lines are bounded by the inner tangent of these two circles. The area between the two tangents is called the *parallax beam*.

Notice how the smaller parallax (Figure 2.20b) leads to larger opening angle compared to slightly larger parallax (Figure 2.20a). The effect of the noise is more significant of smaller parallax.

The parallax beam can be defined by 3 points: the midpoint of segment $[x_i^{'}, \quad Hx_i]$ denoted by m, and the 2 points of tangency each corresponding to one parallax boundary line denoted by t_1 & t_2. See figure 2.21a.

Taking the coordinate frame to be centered around $Hx_i^{'}$. Let $m = (x_m, y_m)$ and $t_1 = (x, y)$. We can write that:

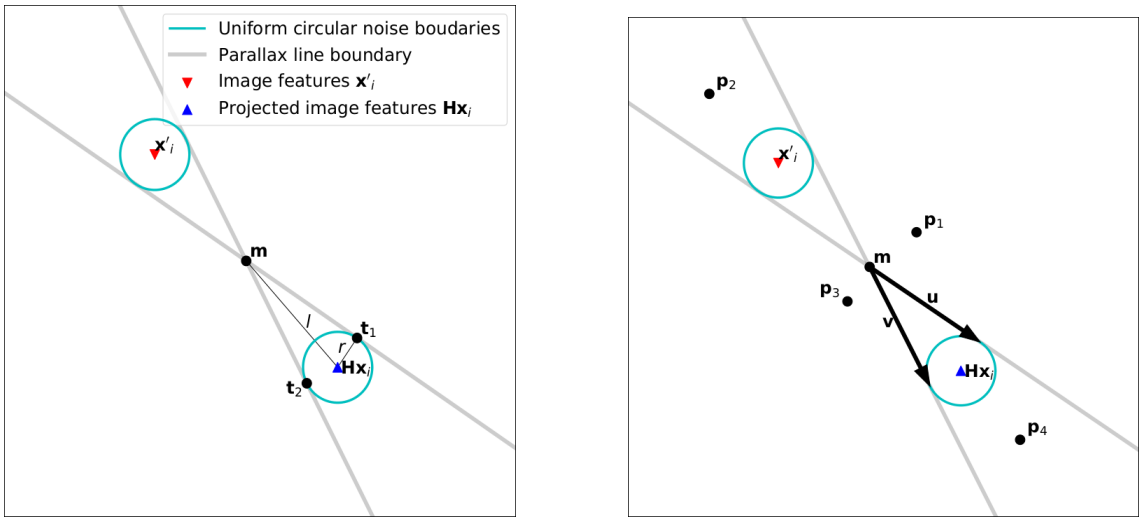
$$r^2 = x^2 + y^2 \tag{2.12}$$

$$l^2 = r^2 + (x - x_m)^2 + (y - y_m)^2 \tag{2.13}$$

$$x_m^2 + y_m^2 = l^2 \tag{2.14}$$

Combining equations 2.12, 2.13 and 2.14 yields

$$x = \frac{r^2 - y_m y}{x_m} \tag{2.15}$$



(a) (b)

Figure 2.21: Parallax Beam Diagram

Plugging equation 2.15 into 2.12, we get a 2^{nd} degree polynomial whose roots are the coordinates of t_1 and t_2 with respect to $Hx_i^{'}$:

$$y^2 l^2 - y(2r^2 y_m) + r^4 - x_m^2 r^2 = 0$$

$$y = \frac{r(ry_m \pm \sqrt{(ry_m)^2 + (lx_m)^2 - (lr)^2})}{l^2}$$

The coordinates of $Hx_i^{'}$ have to be added to go back to the image coordinate frame. Having defined the parallax beam for one point, the same can be done for the others. See figure 2.22.

The step now is to find the epipole. To do so, the area in which most parallax

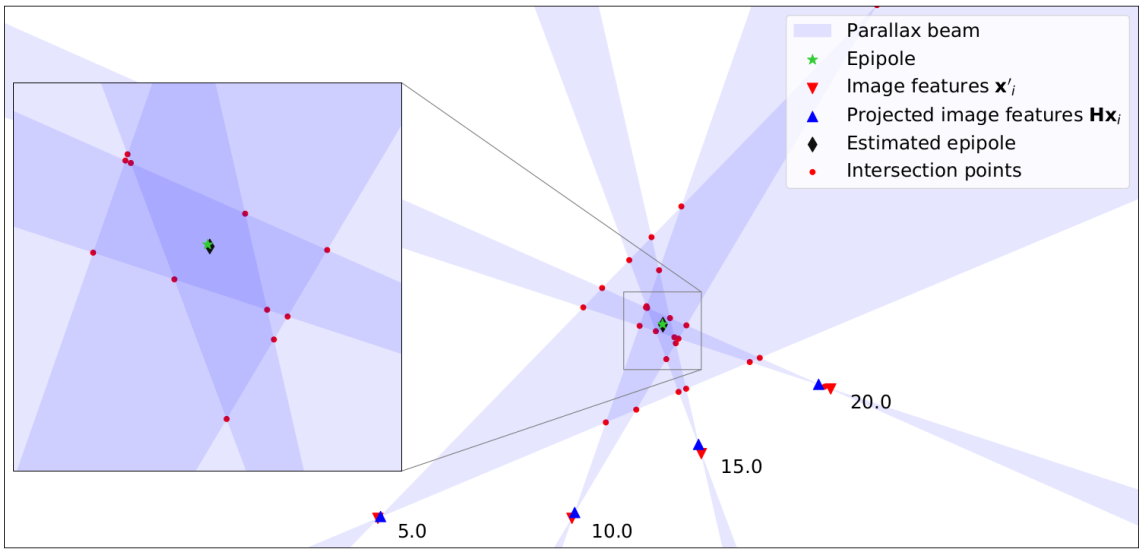


Figure 2.22: Parallax Beams and the epipole

beams overlap must be found. The intersection of all parallax boundaries defines a set of intersection points. The intersection points which are located inside the most number of parallax beams are define the vertices of the area. A simple test can be done to check that. The point is changed to the (m, u, v) basis (figure 2.21b). A straightforward sign test for the coordinates can tell the position of the point with respect to the parallax beam. The table 2.2 summarizes the results.

Point in 2.21b	**Position**	**Sign of u**	**Sign of v**
p_1	Out	+	-
p_2	In	-	-
p_3	Out	-	+
p_4	In	+	+

Table 2.2: Relation between point sign (in (m, u, v) basis) and position relative to the parallax beam

2.4.2 System Overview

During these studies, the small rover shown in Figure 2.23 was used. The developed system was directed towards military applications. The camera used on the rover was a wide angle camera and was tilted around 16^o downwards so that the images taken by the rover were mostly planar. This tilt also reduces the glare caused by the sun.

The parallax beam concept has been applied to a VO pipeline. Two images, one at time t and the other at time $t-1$, have been provided as input to the system. Features in both images are detected using a Harris detector. Features are

Figure 2.23: Rover used during IRIMAS Studies

then described using a BRIEF descriptor and matching is done using the Hamming distance as the matching rule.

The system then tries to estimate the homography. Tests are done to check the viability of the estimated homography. In case the test fails, the image is considered to be not completely planar and the epipolar geometry approach is used. In order to make use of the previously calculated homography matrix, the fundamental matrix is estimated using homography and the epipole. This epipole can be found using the parallax beam method. The fundamental matrix in this case is

$$F = e \times H \tag{2.16}$$

where e is the epipole [17].

With the knowledge of the calibration matrix K, the fundamental matrix is converted to the essential matrix.

$$E = K^T F K \tag{2.17}$$

The essential matrix can be then decomposed to recover the camera motion.

Chapter 3

Resources

3.1 ARTEMIPS

The Autonomous Real-Time Experimental platform of MIPS (or ARTEMIPS for short) is an autonomous test car owned by Université de Haute-Alsace. The car is a Renault Grand Scenic 3 (Figure 3.1).

Figure 3.1: ARTEMIPS

ARTEMIPS is equipped with various sensors and actuators. It has:

- high precision Oxford IMU (Inertial Measurement Unit) RT-3002 with DGPS technology
- 2 IBEO LUX laser scanners

- 2 VLP-16 Velodyne laser scanners
- MANTA G-125 camera

The position of the sensors are depicted in millimeters with an accuracy of ± 5mm in all three axes X, Y, Z. The coordinate system considered here is defined such that:

- X is oriented along the longitudinal axis
- Y is oriented along the lateral direction (positive to the left)
- Z is vertically upwards.

The center of gravity (CoG) has been measured with respect to the vehicle wheelbases and is located at 1224mm from front axis, at 787mm from the center of the left wheels and at around 400mm from the ground with the car unloaded. The table 3.1 shows the (X, Y, Z) position of each sensor and its orientation with respect to the center of gravity.

Sensor	**Orientation**	**Position**
Left VLP-16	(-1,1,-1)	(-246,710,1230)
Right VLP-16	(1,-1,-1)	(-240,-750,1230)
MANTA G-125	(1,1,1)	(646,0,856)
IMU RT-3002	(1,1,1)	(3,-19,0)
Front LUX	(1,1,1)	(1825,0,-8)
Rear LUX	(-1,-1,1)	(-2398,0,-135)

Table 3.1: Sensor position and orientation

ARTEMIPS is also equipped with 3 actuators: 2 integrated servo motors MAC-141 to pilot the steering wheel and the brake of the car, as well as a NI multi-function DAQ system to pilot the engine of the car.

All sensors and actuators are connected to an embedded computer that runs a software called RTMaps from Intempora. It is a platform dedicated to multi-sensors and multi-actuators systems. Software for the vehicle is developed and deployed using RTMaps. See Figure 3.2

The camera takes photos with a frame rate of 31 frames per second (fps) at full image resolution.

3.2 KITTI Dataset

The KITTI dataset is vision benchmark suite for stereo, optical flow, visual odometry, 3D object detection and 3D tracking. Developed as a joint project between

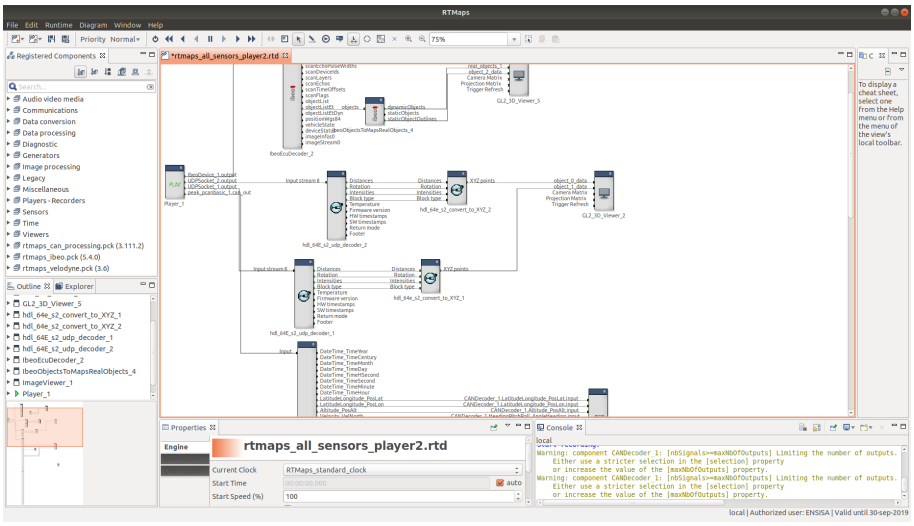


Figure 3.2: RTMaps window

Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, the KITTI dataset provides real-world sequences to test algorithms for autonomous driving.

The dataset has been recorded from a moving platform while driving in and around Karlsruhe, Germany. It includes camera images, laser scans, high-precision GPS measurements and IMU accelerations from a combined GPS/IMU system.

The dataset contains a benchmark for odometry which consists of 22 sequences, 11 of which (sequences 00-10) are for testing and others are for validation.

Figure 3.3: KITTI image sequence (sequence 07)

Images from the KITTI odometry benchmark are 1226x370 pixels in size and are in grayscale. They are taken at a rate of 10 images per seconds. Figure 3.3 shows a sequence of images from sequence 07. Figure 3.4 shows an image from sequence 00.

The poses of the camera at each frame of a given test sequence are provided in a text file were each line contains the raveled $3x4$ transformation matrix $(R|t)$.

The KITTI Dataset provides a good benchmark for testing the algorithms offline.

Figure 3.4: KITTI Image (sequence 00)

3.3 OpenCV

Open-source Computer Vision, or OpenCV for short, is one of the most famous computer vision libraries.

The library has more than 2500 algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. That involves:

- Reading images and loading video frames from files or live cameras
- Image Processing
- Camera Calibration
- Feature detection, description and matching using various algorithm (e.g. Harris, SIFT, AKAZE)

It has interfaces for C++, python, Matlab and Java and can work on Windows, Linux, Android and Mac OS. It also has good documentation and a large community. That is why it has been used in the context of this work.

Chapter 4

Implementation

The objective of this master thesis was to implement a monocular VO system that uses *parallax beams* and study its usage on the system directed for autonomous driving. It is to be used on ARTEMIPS.

To do so, the VO pipeline must be developed. Even though the pipeline is simple, the process of actually developing it is complex. As stated before, different features yield different results. Also, the way that different algorithms work in doing the same task heavily depends of the scenario we are in.

For rapid prototyping and testing, python was used. OpenCV was the main computer vision library used. Other libraries like numpy, OpenGL, pangolin and matplotlib were used for data manipulation and visualization.

4.1 Pre-tests

A preliminary test was done to help decide how to approach the problem. An OpenCV-based simple visual odometry was first developed. Two cases were taken into consideration: (1) matching of the features of corresponding frames and (2) tracking of the features throughout sequence of frames.

Using a simple VO pipeline (see figure 2.7), different feature extractors and descriptors were used for the two aforementioned cases. This way, we are able to deduce which combination of techniques can provide a good candidate for the visual odometry system. For this stage, KITTI dataset was used.

The algorithms goes as such. The input of the algorithm is the image feed and

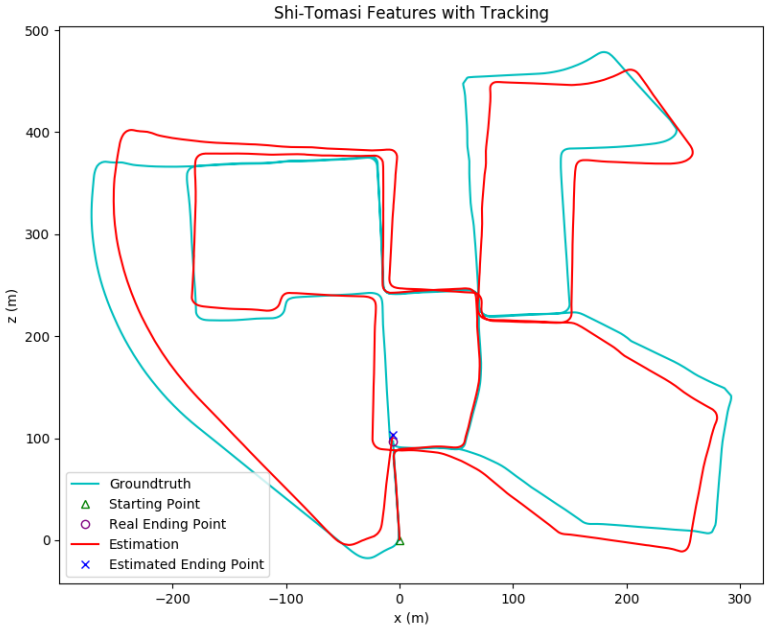
the calibration matrix. Images are feeded to the system one by one. The features are then extracted using the feature extractor of interest. OpenCV has a function for almost all feature extractor (*goodFeaturesToTrack* for Harris and Shi-Tomasi, *SIFT_create* for SIFT, etc.).

The next step is to find feature correspondences in the second image. In case of feature tracking, the Lucas-Kanade tracker with pyramids has been used. Various window sizes and pyramid levels have been tested. A 15x15 to 30x30 window is a good choice for our image sizes. Also 7 levels have provided good results. More than 7 has not done much change on the output points. The feature extractor is used again when the number of points drops below a predefined threshold (< 500 in this case) and when tracking has been done for more than a certain number of frames (15 frames here). With tracking, we are able to find the set of corresponding points. In case of feature matching, the features need to be described first. Several descriptors (BRIEF, SIFT, etc.) have been tested. Then matching is done to find the corresponding points. Lowe's Ratio test is applied in order to get better matches.

The corresponding point pairs (whether from tracking and matching) are used to estimate the motion. Epipolar geometry is used in this case. The essential matrix is estimated using the *findEssentialMatrix* function and then the function *recoverPose* is used to get the camera pose. The *recoverPose* function also performs a cheirality check that can give the correct camera pose.

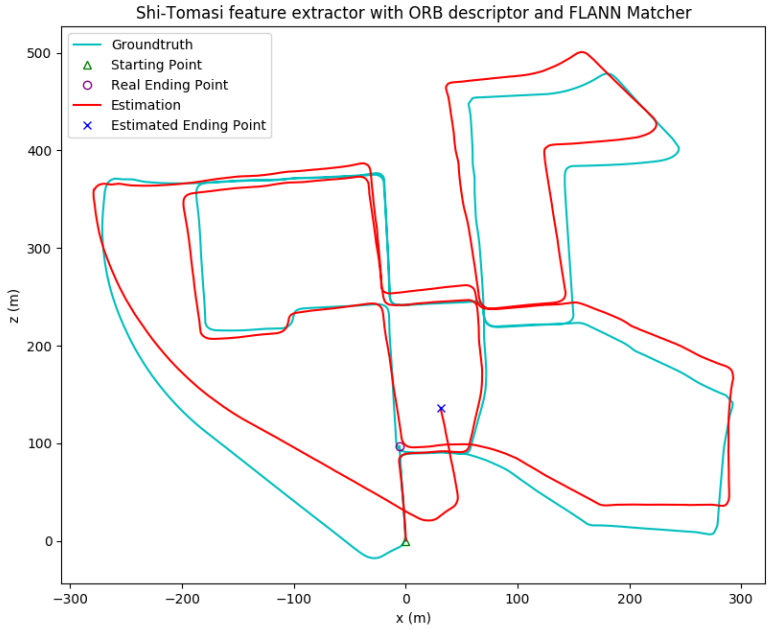
Figures 4.1a through 4.8 show the results of the VO algorithm applied on the sequence 00 of the KITTI Dataset. This sequence is characterized by many turns, and few traffic and external disturbances like cars or crowds of people. The time needed by the tested sequence (00) is 470.5816 s (around 7.8 minutes).

Feature extractors were set to detect around 2500 features. For the Brute Force (BF) matcher L_1 and L_2 losses were tested. For the FLANN matcher, a default value of 100 was used for the checks. In tracking of features, a window size of 21x21 and a 7 level pyramid were used. RANSAC was used to estimate the Essential matrix in all cases. All other parameters were kept at OpenCV defaults.



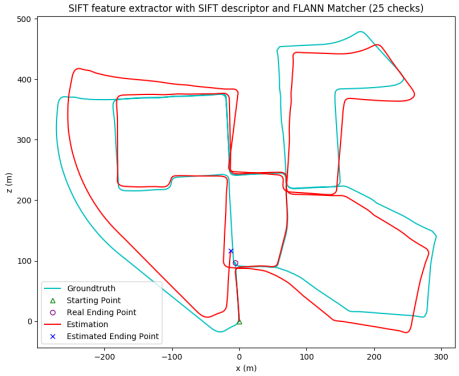
(a) With L2 loss

(b) With L2 loss

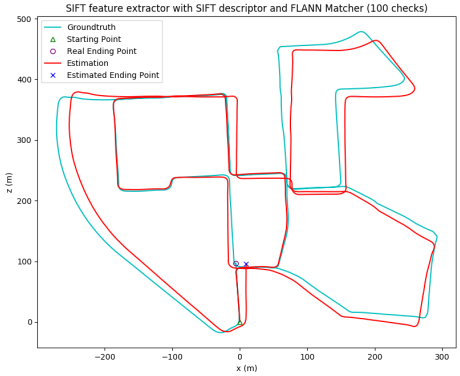


(c) With FLANN

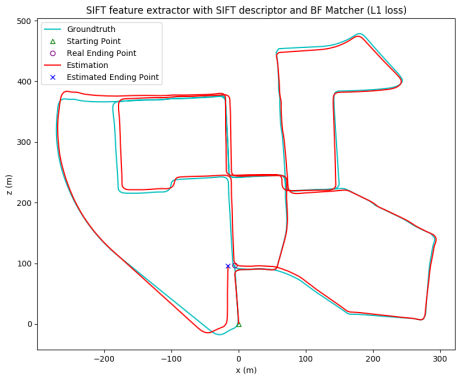
Figure 4.1: Shi Tomasi with ORB Descriptor and Matching



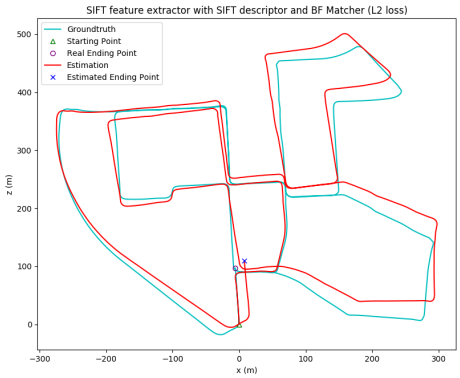
(a) With FLANN (25 checks)



(b) With FLANN (100)

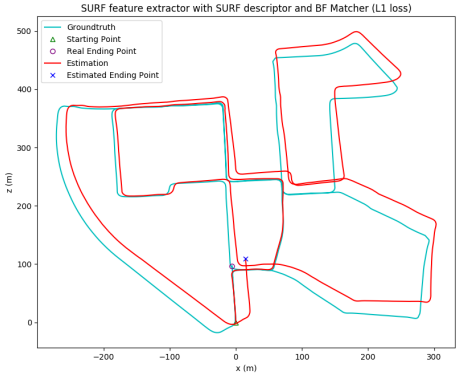


(c) With L1 loss

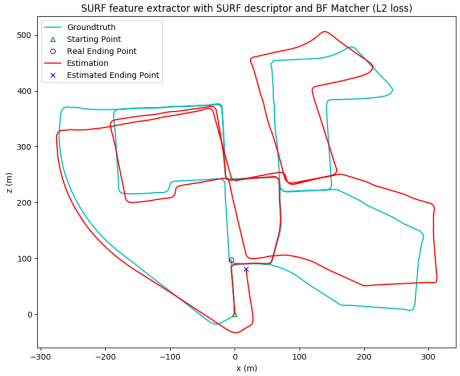


(d) With L2 Loss

Figure 4.2: SIFT Features and Descriptor with Matching

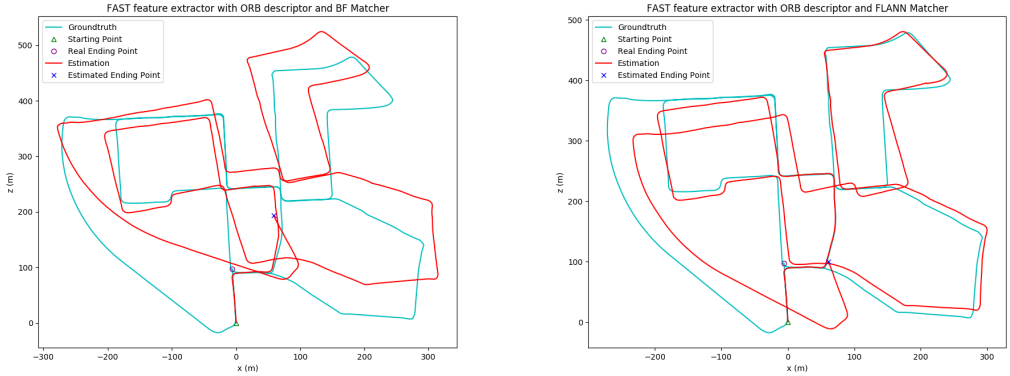


(a) With BF (L1 loss)



(b) With BF (L2 Loss)

Figure 4.3: SURF Features and Descriptor with Matching



(a) With BF (L2 loss) (b) With FLANN

Figure 4.4: FAST Features and ORB Descriptor with Matching

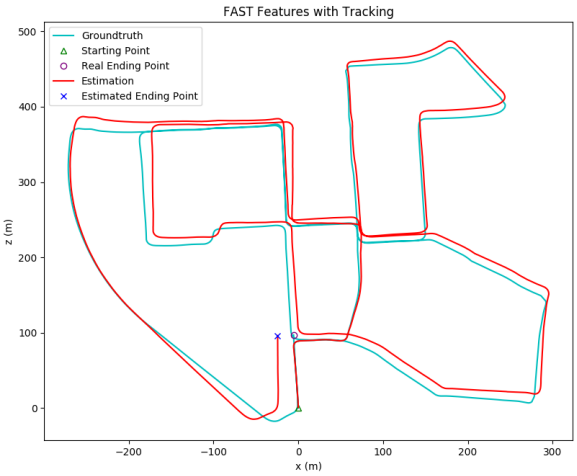


Figure 4.5: FAST features with Tracking

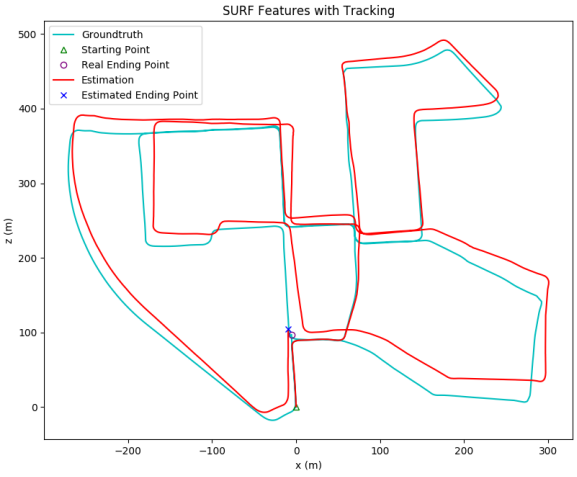


Figure 4.6: SURF features with Tracking

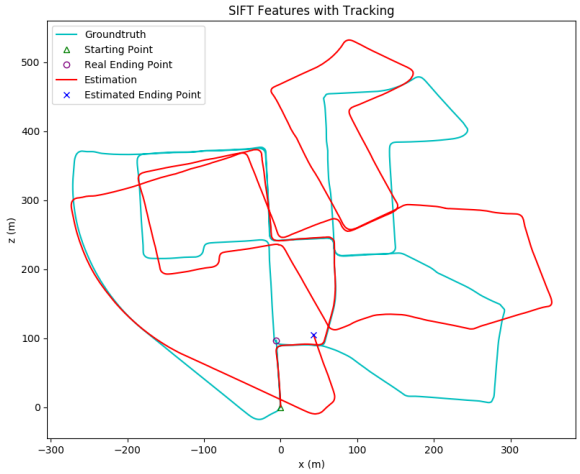


Figure 4.7: SIFT features with Tracking

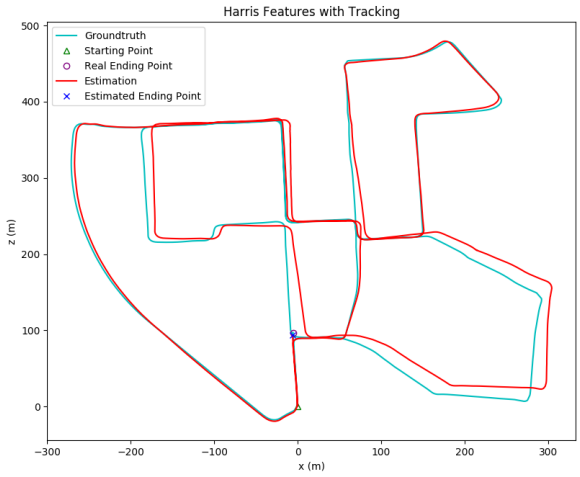


Figure 4.8: Harris features with Tracking

Figures 4.1a through 4.8 represent some the obtained results. They show the birds eye view of the ground-truth path (in cyan) compared with that of the one generated from the VO system (in red). Comparing the real and estimated trajectories, it can be seen that the SIFT with L1 brute force matcher (Figure 4.2c), FAST with tracking (Figure 4.5), and Harris with tracking (Figure 4.8) provided the best results. Others, such as SURF with tracking (Figure 4.6) and SIFT with L2 brute force matcher (Figure 4.2d), have good result however the estimated path in these cases are drifting more away from the real path.

Another criteria was also recorded, the execution time. The code was tested on a computer with i7 Processor (i7-7500U), GeForce 940MX 2GB, 8GB DDR4. The

table 4.1 summarizes the most interesting results. As expected, the overall time with tracking is less than that of matching. This is due to the fact that we are only detecting once and tracking the features rather than detecting the features in two images, describing them and then matching.

Extractor—Descriptor—Matching/Tracking	**Execution Time (s)**
SIFT—SIFT—L2 BF Matching	1488.549
SIFT—SIFT—L1 BF Matching	1336.652
SIFT—SIFT—FLANN Matching	1149.132
SURF—SURF—L2 BF Matching	2442.035
FAST—ORB—L2 BF Matching	2091.549
Shi-Tomasi—ORB—FLANN Matching	436.453
FAST— —Tracking	413.391
SIFT— —Tracking	329.172
SURF— —Tracking	328.477
Shi-Tomasi— —Tracking	255.816
Harris— —Tracking	174.392

Table 4.1: Execution times

4.2 Algorithm

After having studied the possible combinations, tracking was deemed to be suitable for this operation. The algorithm assumes that the camera is calibrated and the camera calibration matrix and distortion parameters are at hand.

The proposed algorithm for localization of the vehicle follows the general VO pipeline.

The first image is sent as input. This image is undistorted. Features are extracted from the images using a modified Harris corner extractor. A grid is applied to the image and the features detector is applied to the different grid sections. This allows for a more homogeneous feature distribution over the image, which helps in the estimation of the essential matrix. The threshold used in feature detection is adjusted according to the maximum pixel value in the grid sections.

The figure 4.9 shows a comparison between the output of the regular Harris corner detector and the proposed solution. The features in figure 4.9b are more distributed along the image and thus provide a better description of the image geometry.

When a new image in the sequence/feed arrives, the loop starts. A LK tracker with pyramids is used to find the position of the previously extracted points in the

(a) Harris Corner Detector

(b) Proposed Corner Detector

Figure 4.9: Comparison between the regular and modified Harris Corner Detector

new image. Thus a feature correspondence is found. The LK tracker has a 30x30 window and a 7 layer pyramid. See Figure 4.10

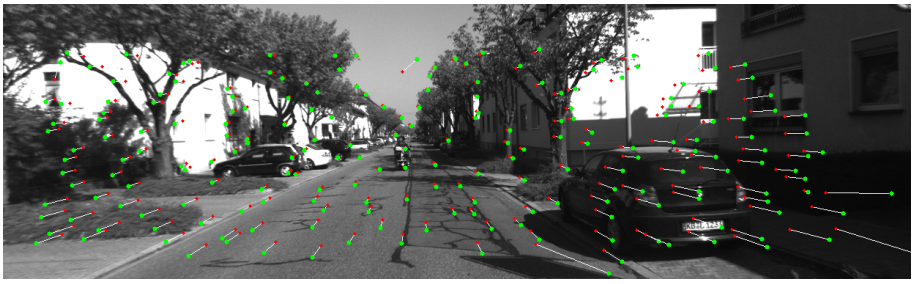


Figure 4.10: Tracking Features. Red points denote features from the previous frame. Green points are their tracked correspondences in the current frame.

The homography matrix is estimated using a RANSAC-based method (implemented in OpenCV's *findHomography* function) with a threshold of 20. If the ratio of inliers to the total number of feature correspondences r used in the estimation process is greater than a predefined value (0.9 in our case), then we can assume that the scene in the image is mainly planar as most of the features satisfy the homography relation. In this case, the homography matrix is decomposed into rotations, translations and normal vectors to the plane [18]. Four possible combinations exist; two of which can be directly discarded as they correspond to the reverse direction of motion. A quick test comparing the normal vectors to the previous normal vector is done to identify the best solution. Since the images are consecutive, the deviation of the normal must be minimal.

If r is less then the predefined value, the scene can not be considered planar and

the essential matrix must be used. In order to make use of the previously estimated homography matrix, the parallax beam method is used. The epipole is estimate and equations 2.16 and 2.17 are used to get the essential matrix. The essential matrix is then decomposed using OpenCV *recoverPose* function which recovers relative camera rotation and translation.

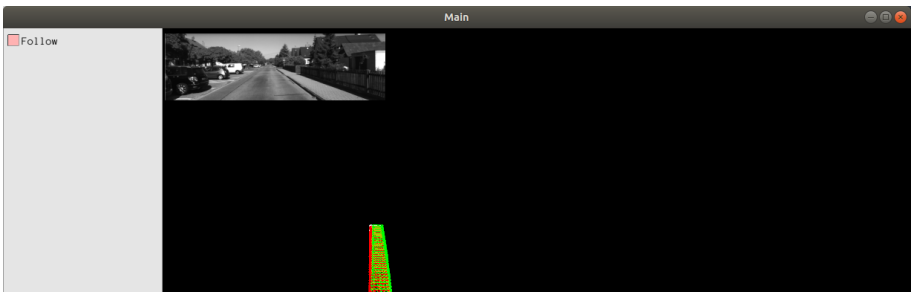


Figure 4.11: 3D Visualization of the localization. The estimation is in red. The groundtruth is in green.

The number of feature correspondences in the current frame is checked. In case that number is less than a certain threshold (70 for example), then the feature detector is used to get more features. These new features are appended to the previously tracked features. The process loop is done again until the image feed stops. The odometry of the vehicle is then gradually built. See Figure 4.11. This is a 3D visualization of the VO process. The green part in the image is the groundtruth path and in red is the estimated trajectory of the vehicle. The upper right corner shows the current frame.

The Figure 4.12 shows the flowchart for the algorithm.

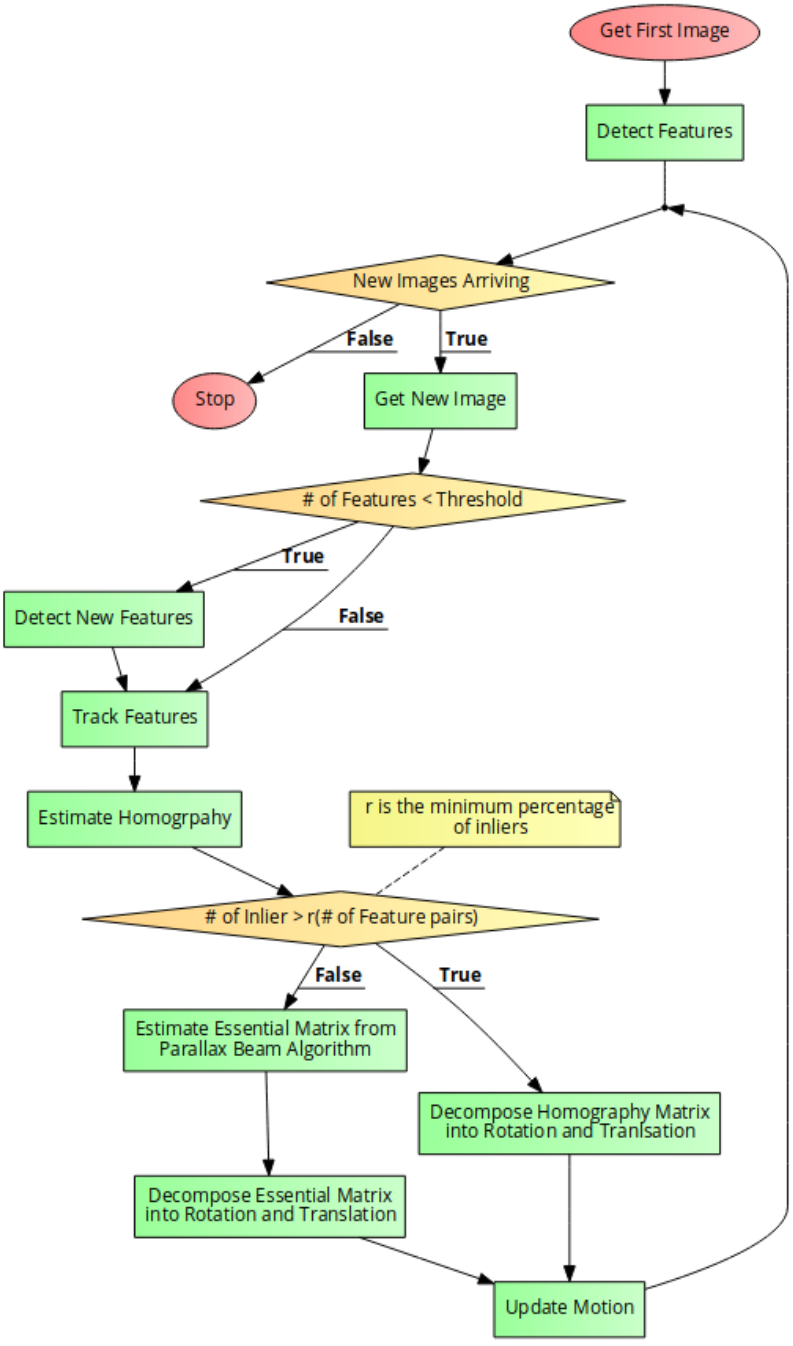


Figure 4.12: Algorithm Flowchart

Chapter 5

Results

The proposed algorithm was tested on both the KITTI Dataset and on ARTEMIPS. In this section, the evaluation criteria are explained and some results are demonstrated.

5.1 Evaluation Criteria

One of the evaluation criteria used is that presented in the KITTI vision benchmark for odometry. Errors are calculated relative to a starting point which changes by a predefined number of steps (10 steps). For each starting point, the errors are measured for path lengths of 100,200,300,400,500,600,700, and 800 meters. Two types of errors are measured: the 3D rotation error ϵ_{rot} and the 3D translation error ϵ_{trans}.

To define these errors, we consider the general transformation matrix of (i) relative to (k)

$$T(k,i) = \begin{pmatrix} R(k,i) & t(k,i) \\ 0^T & 1 \end{pmatrix}$$

where R is the rotation matrix, t is the transformation matrix and $0^T = [0,0,0]$.

For simplicity, we drop the (k,i). Let T_{gt} be the ground-truth pose matrix of the camera at frame *(i)* relative to a starting frame *(k)*. Let T be the estimate at frame *(i)* relative to a starting frame *(k)*. We can write

$$T^{-1}\, T_{gt} = \begin{pmatrix} R^T & -R^T t \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} R_{gt} & t_{gt} \\ 0^T & 1 \end{pmatrix} = \begin{pmatrix} R^T R_{gt} & -R^T(t_{gt} - t) \\ 0^T & 1 \end{pmatrix} = \begin{pmatrix} R_{err} & t_{err} \\ 0^T & 1 \end{pmatrix}$$

From the axis-angle representation, the 3D angle of R_{err} can be extracted:

$$\theta_{err} = arccos(\frac{tr(R_{err}) - 1}{2})$$

where tr corresponds to the trace of the matrix. For each starting point k and length l, the errors are

$$\epsilon_{rot} = \frac{\theta_{err}}{l} \qquad \epsilon_{trans} = \frac{||t_{err}||}{l}$$

The values for equal lengths are averaged. For rotational errors, the unit is degrees per meter. For the translation error, the unit is percent.

Other evaluation criteria are used. In order to have more detailed information on the performance on the algorithm, a direct comparison between the Roll, Pitch, Yaw angles (RPY) and XYZ position of the estimate trajectory and the groundtruth trajectory is done. The frame-by-frame errors in RPY and XYZ are calculated and plotted as histograms with *0.1 degree* bins for rotation and *1 cm* bins for translation. As stated before, the output is stored in the camera coordinate system. In order to have clearer graphs, the data is transformed to a coordinate system with X pointing in front of the vehicle, Y to the left of the vehicle and Z pointing upwards. See Figure 5.1

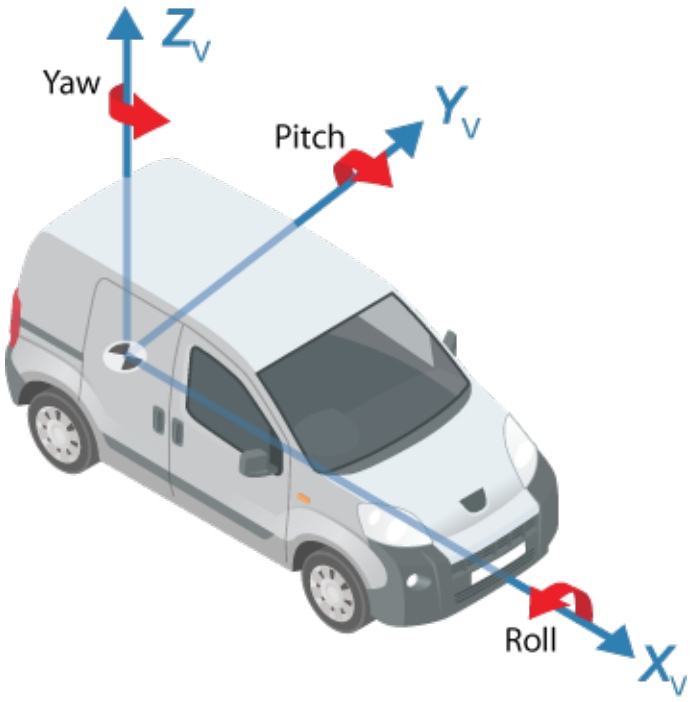


Figure 5.1: Coordinate System, source: MathWorks

5.2 KITTI

Various sequences of the KITTI Dataset have been tested. The results of the VO are demonstrated in this section.

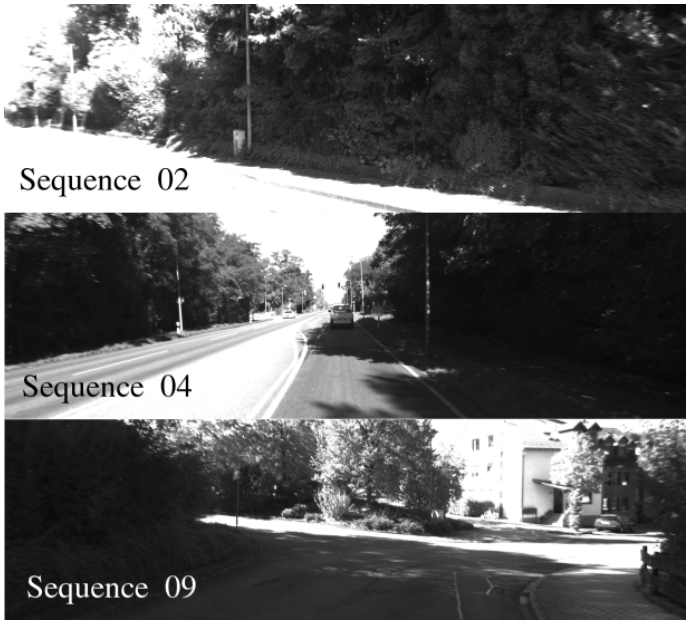


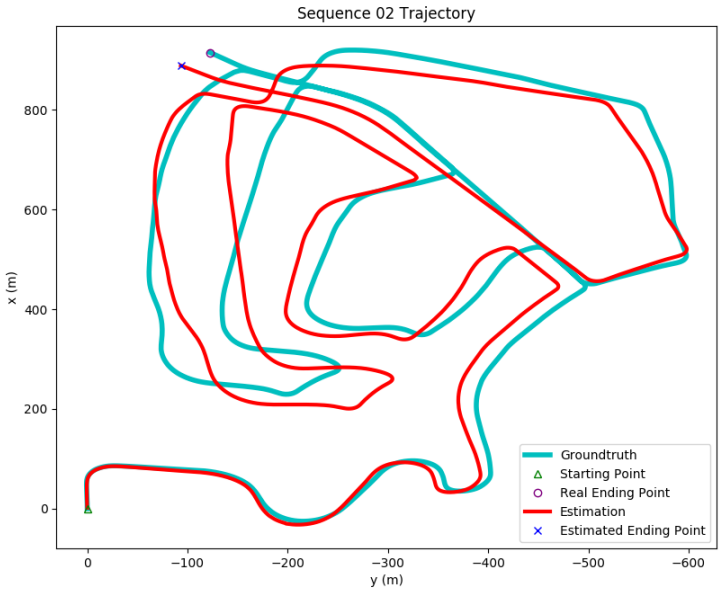
Figure 5.2: Problematic Scenes

As it can be seen, the algorithm developed provides good trajectory estimates on the sequences 03 (figure 5.4), 09 (figure 5.7), and 10 (figure 5.8). While the previous results display the effects of error accumulation (or drift for short) on the visual odometry estimation, it is more evident on sequences 02 (figure 5.3) and 04 (figure 5.5).

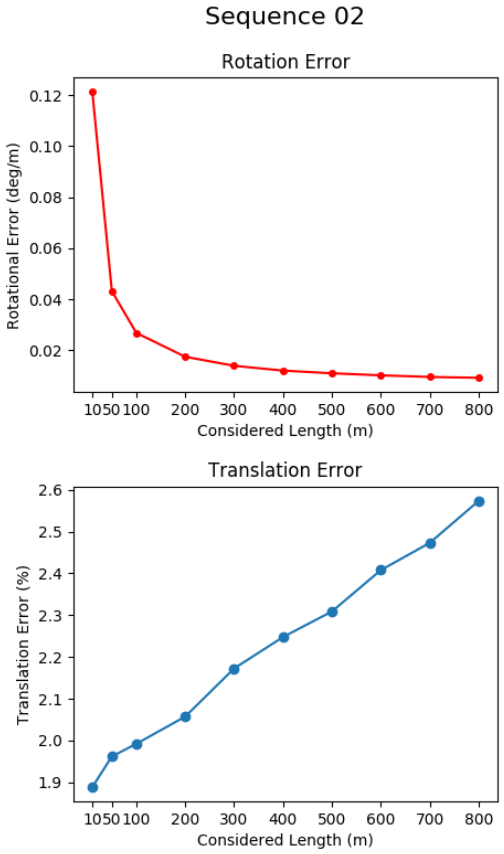
A common point between all the runs is that the errors follow a Gaussian shape. This is evident in figures 5.3d, 5.4d, 5.5d, 5.6d, 5.7d, and 5.8d.

The weak points of the algorithm can be traced back to the features extracted. Various errors seen on the KITTI dataset were due to the difficult in finding features and their correspondences. See Figure 5.2. These scenes have caused high errors. A high contrast between different parts of the images (e.g. the bright left side and dark right side of sequence 04 image in Figure 5.2) has lead to feature detection errors. Also, as tracking relies on pixel intensity, the presence of features near to or in heavily dark (or bright) areas leads to matching errors.

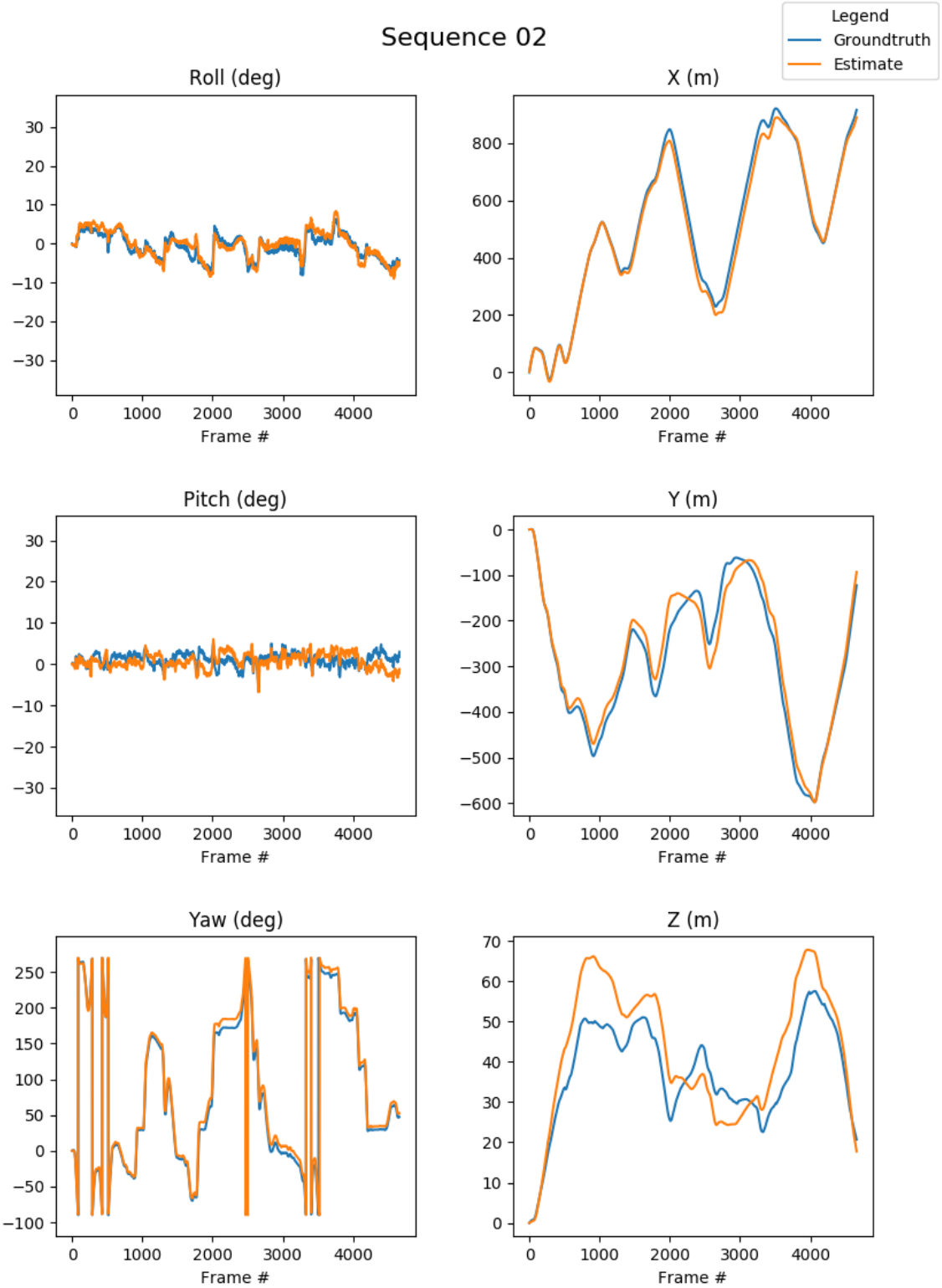
Sequence 02



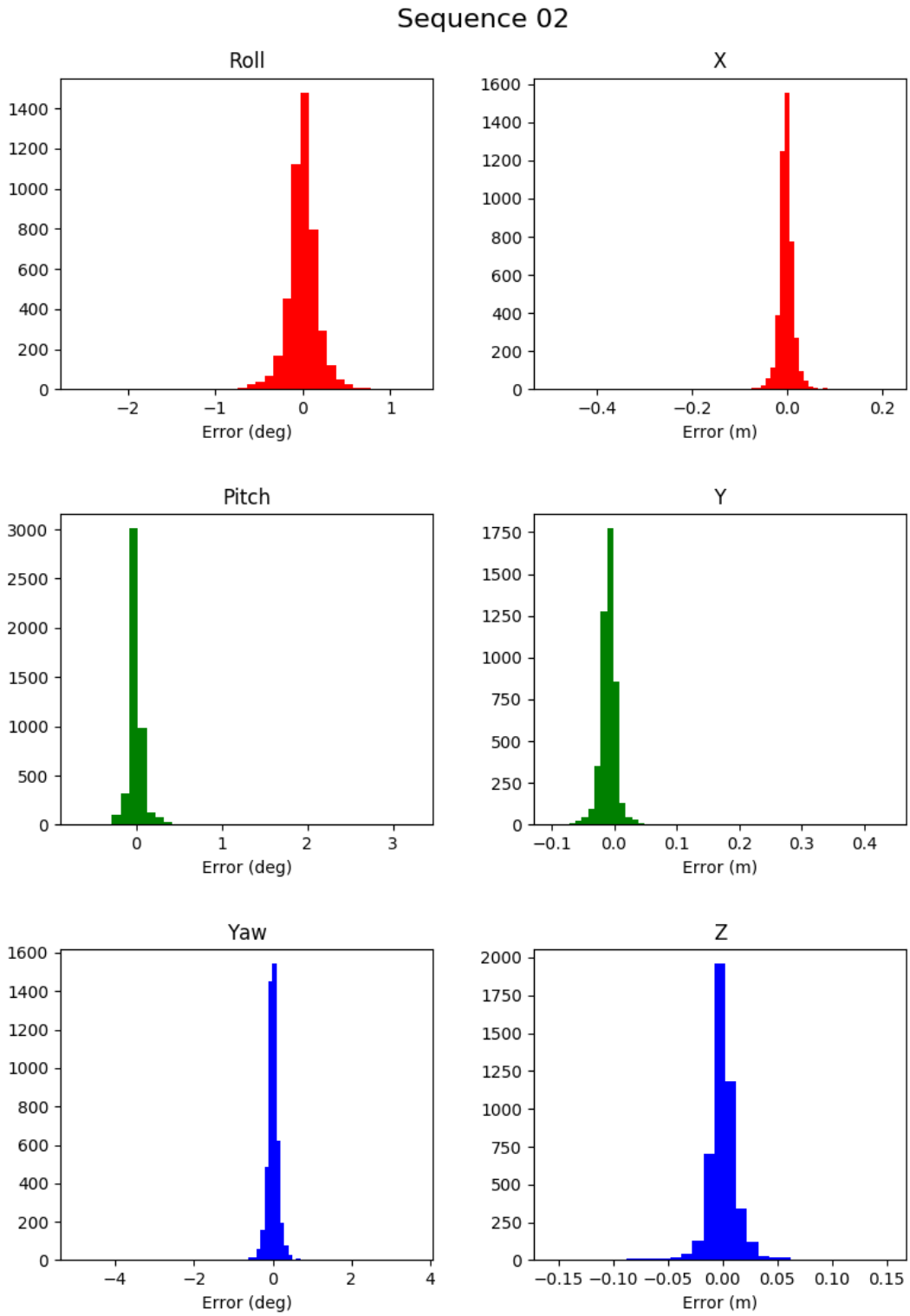
(a) Trajectory



(b) KITTI Evaluation



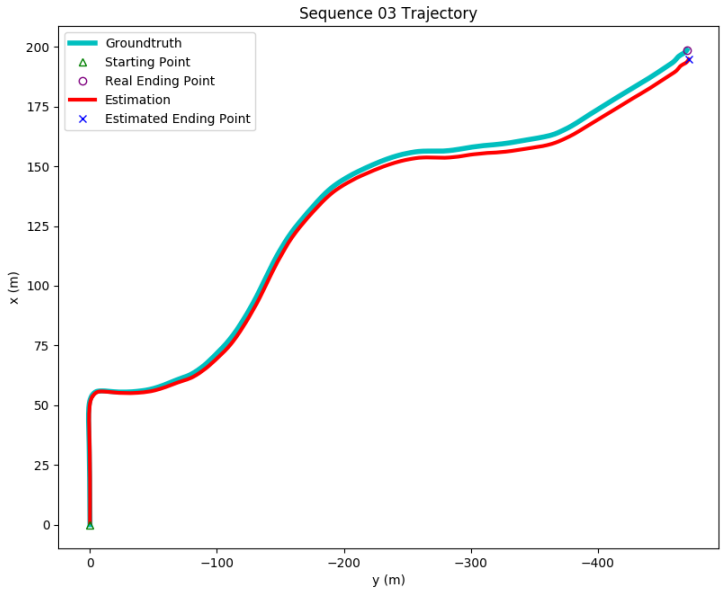
(c) XYZ RPY Plot



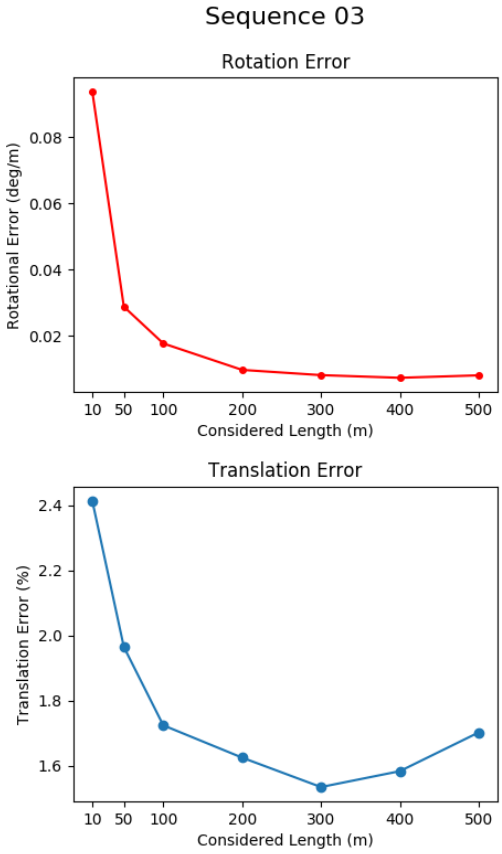
(d) XYZ RPY Error Histograms

Figure 5.3: Sequence 02 Results

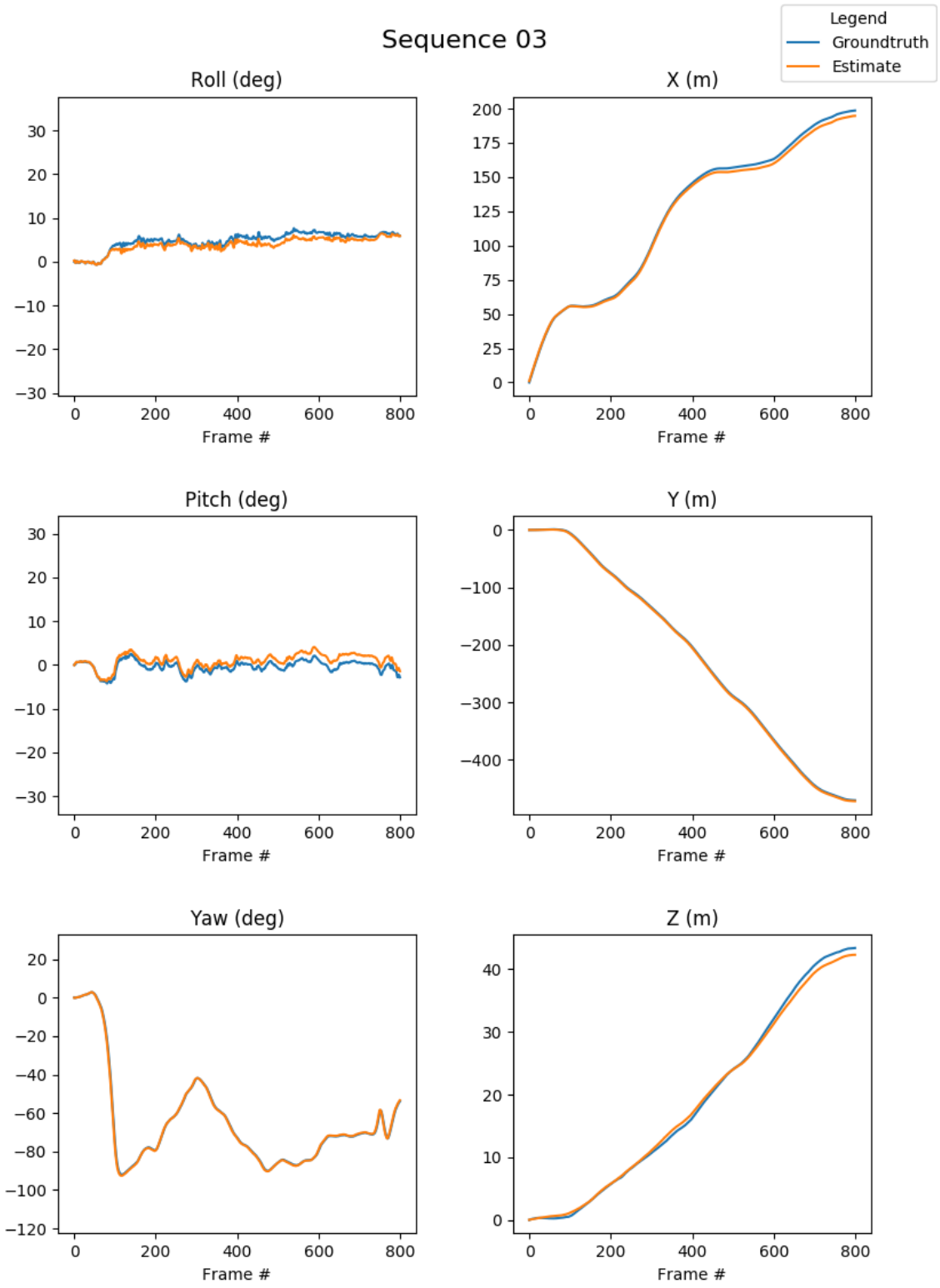
Sequence 03



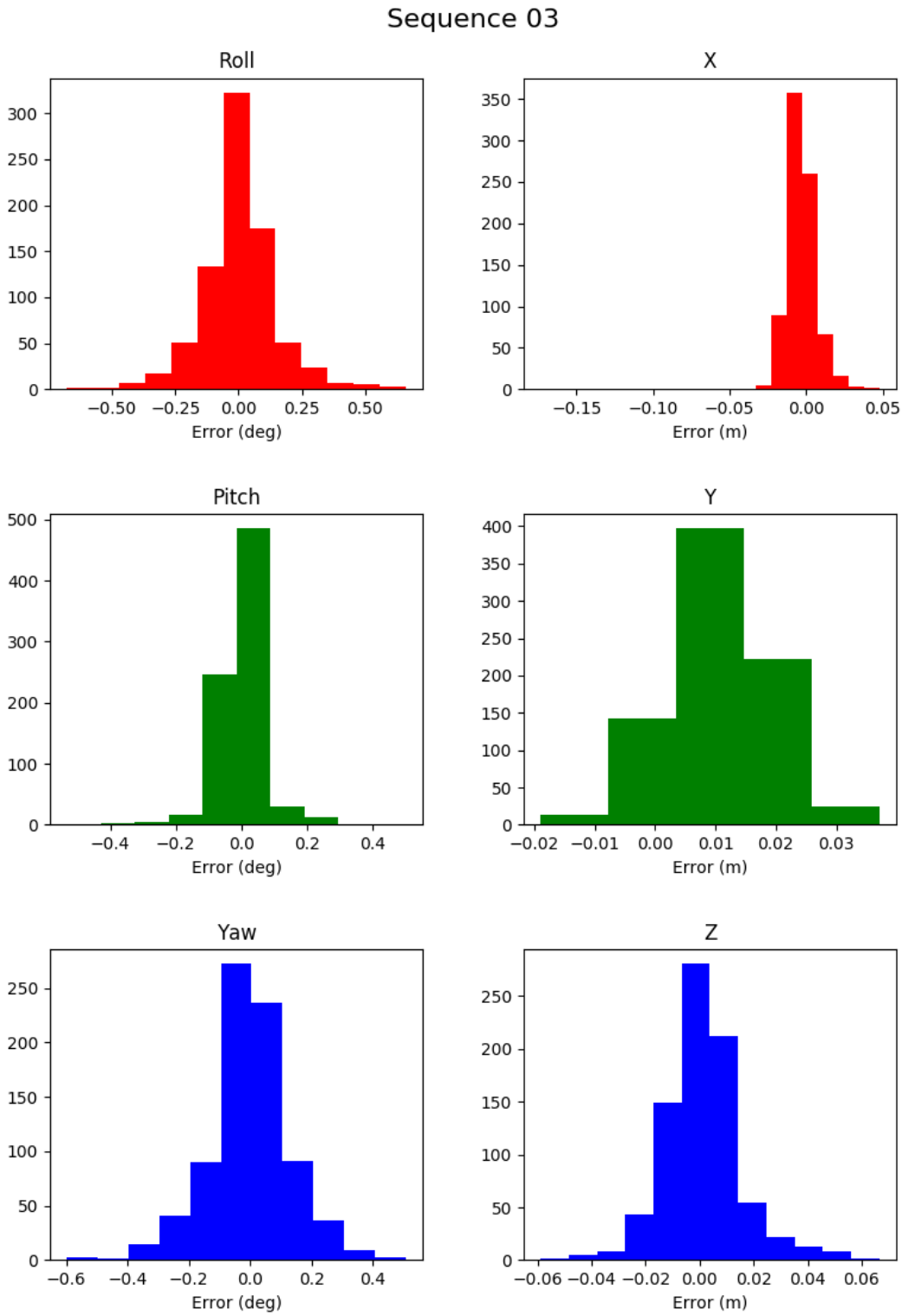
(a) Trajectory



(b) KITTI Evaluation



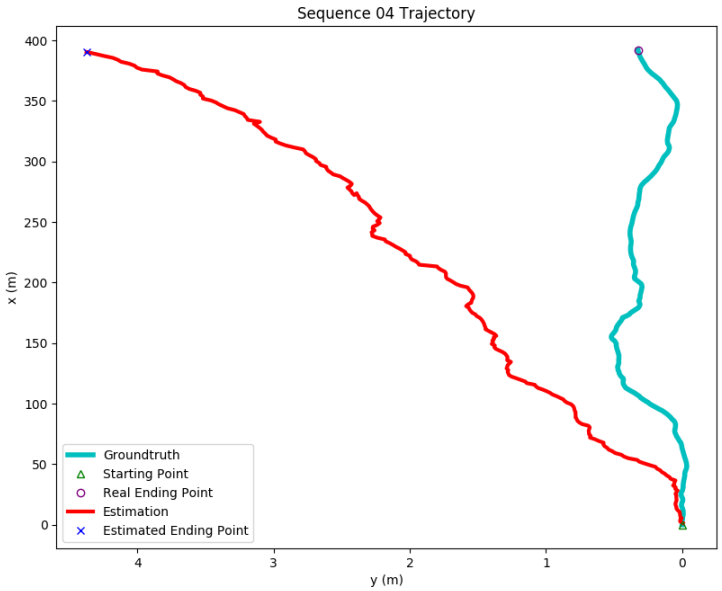
(c) XYZ RPY Plot



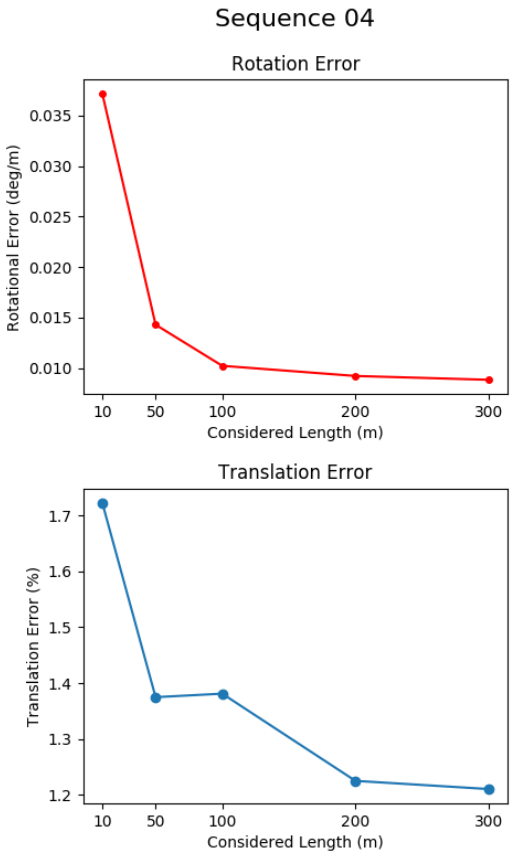
(d) XYZ RPY Error Histograms

Figure 5.4: Sequence 03 Results

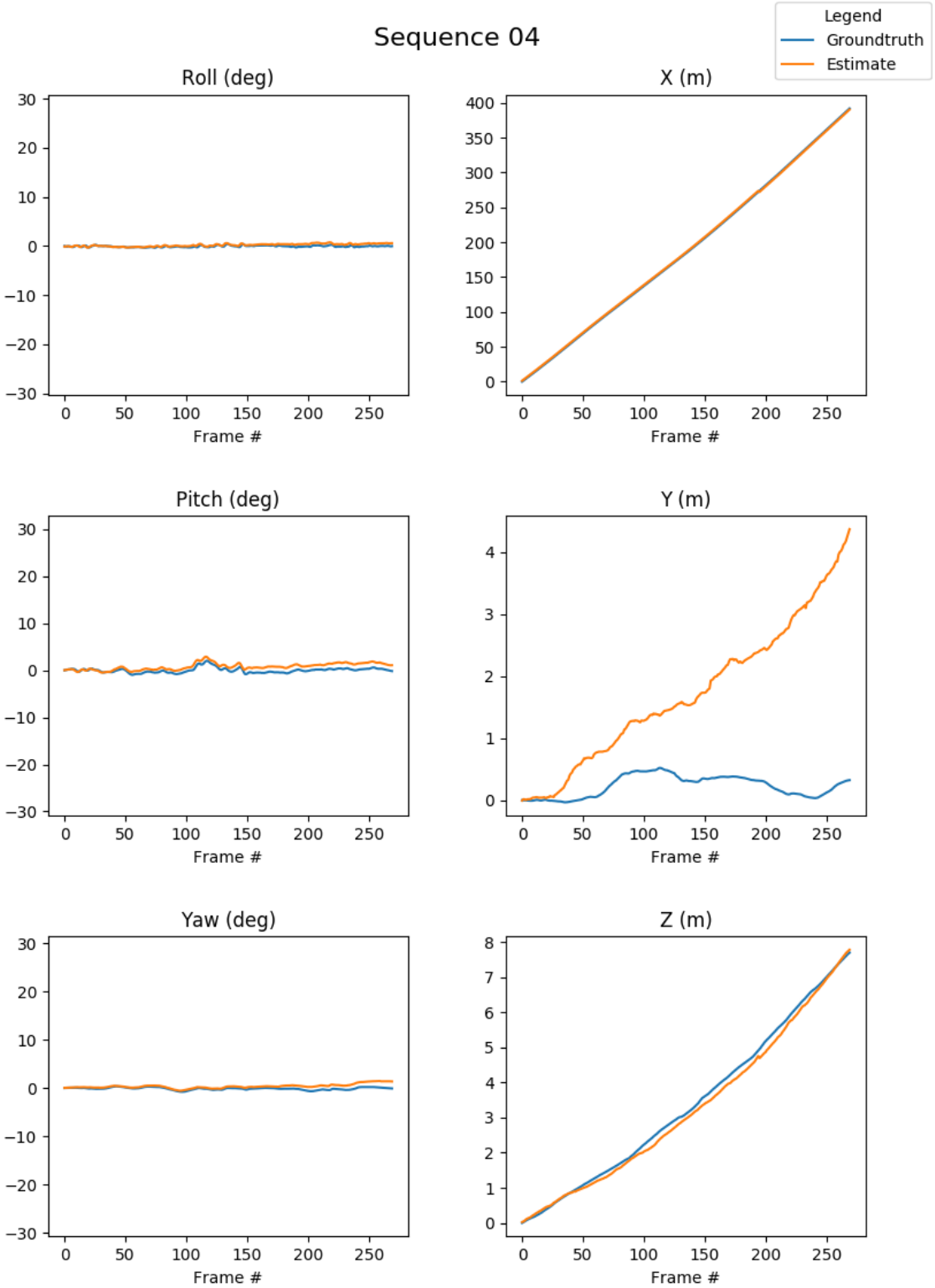
Sequence 04



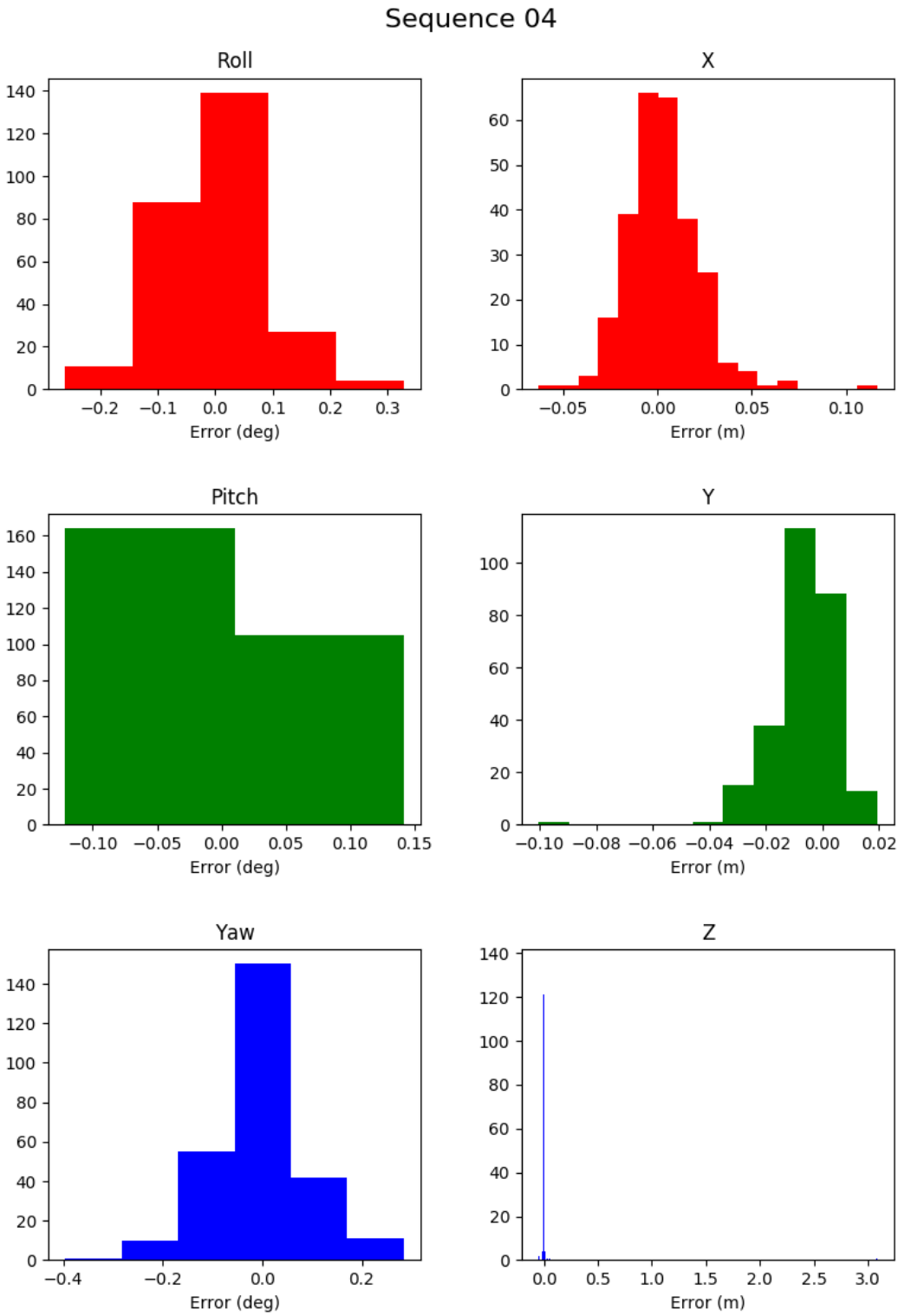
(a) Trajectory



(b) KITTI Evaluation



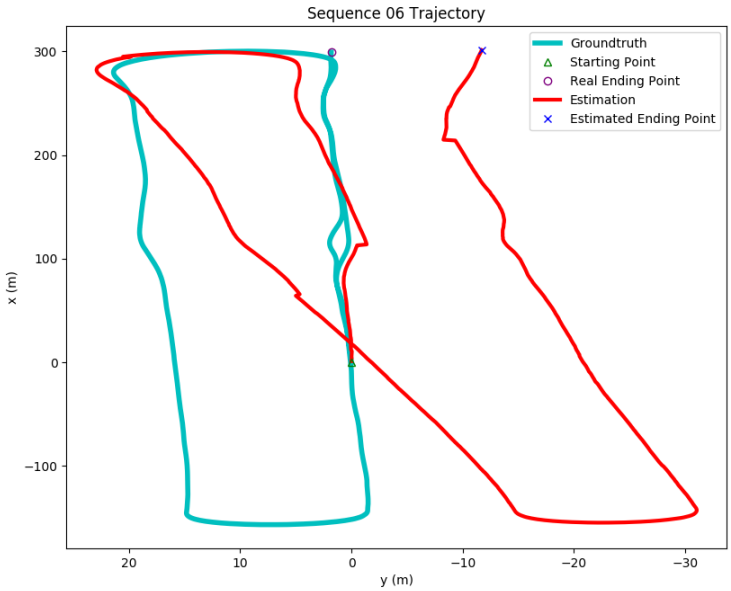
(c) XYZ RPY Plot



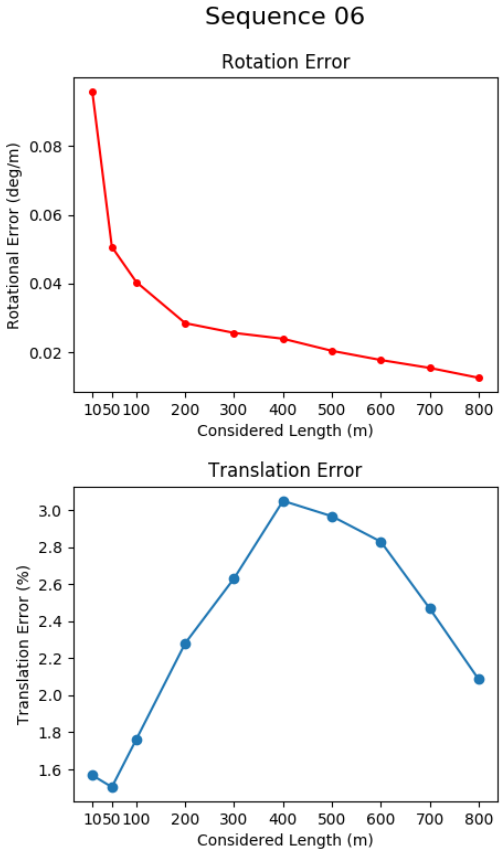
(d) XYZ RPY Error Histograms

Figure 5.5: Sequence 04 Results

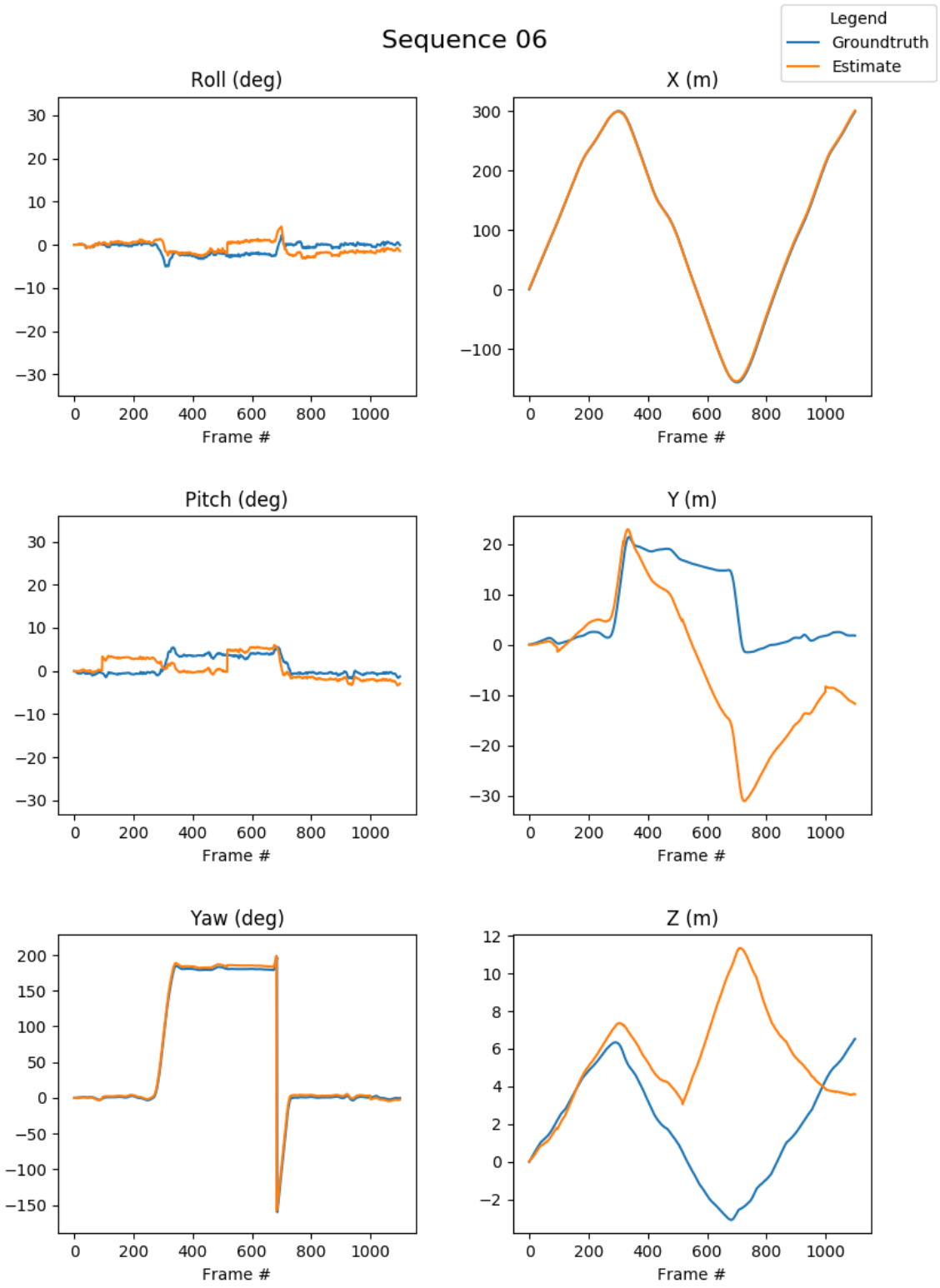
Sequence 06



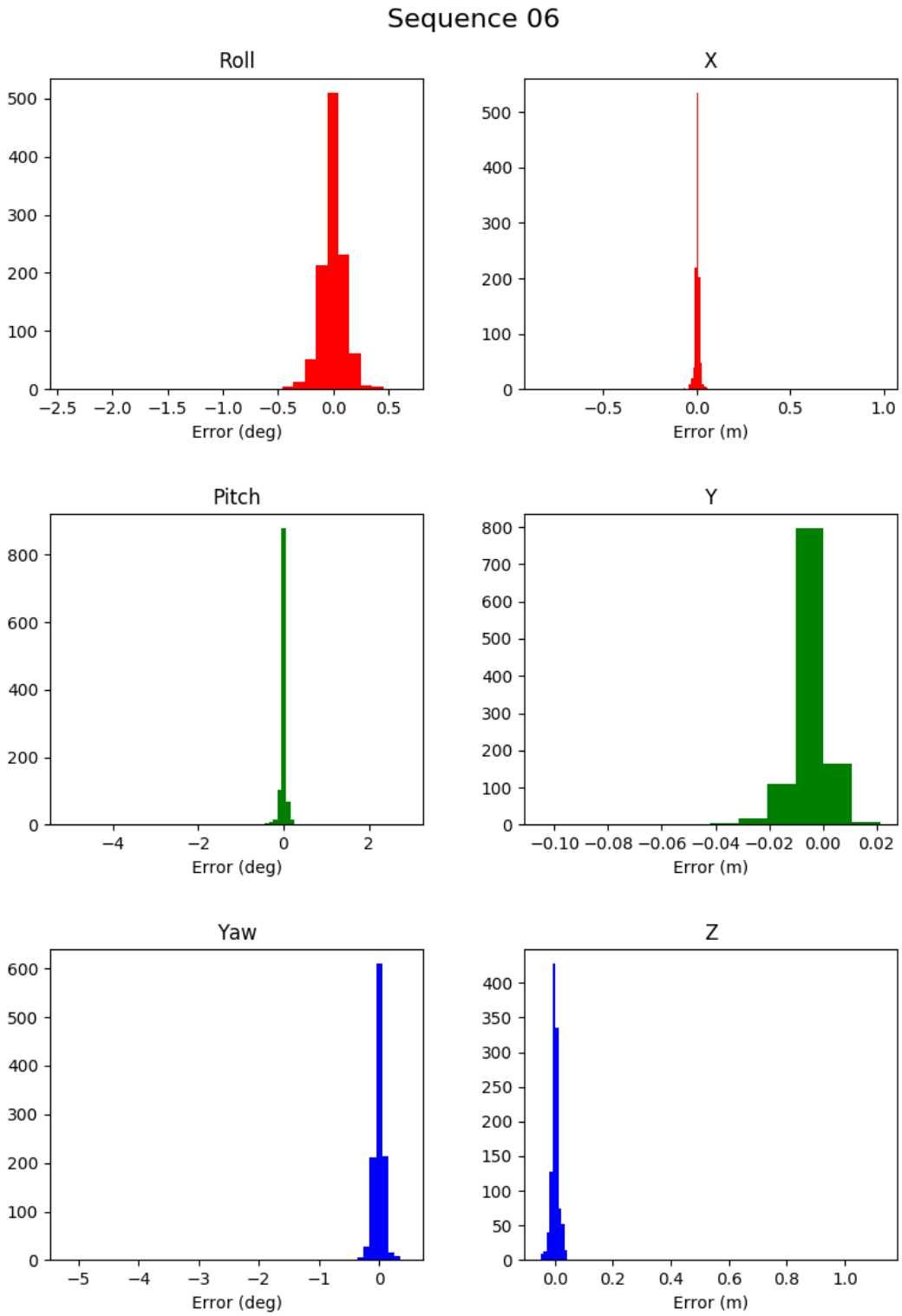
(a) Trajectory



(b) KITTI Evaluation



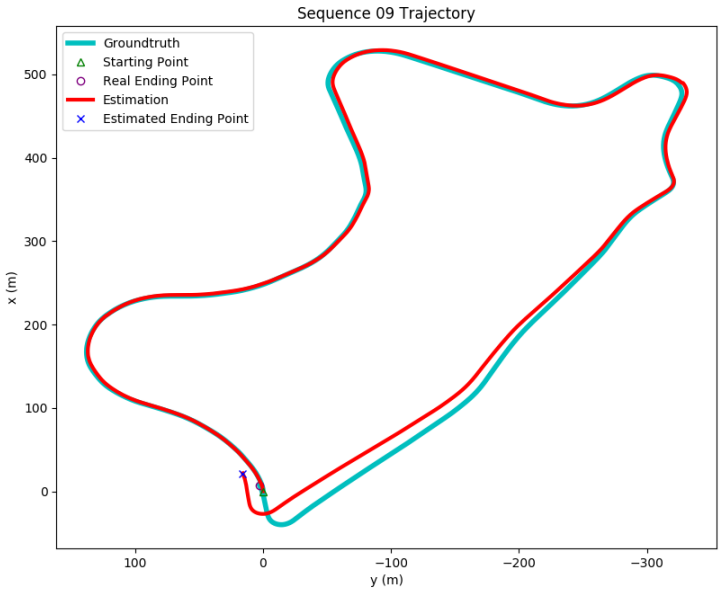
(c) XYZ RPY Plot



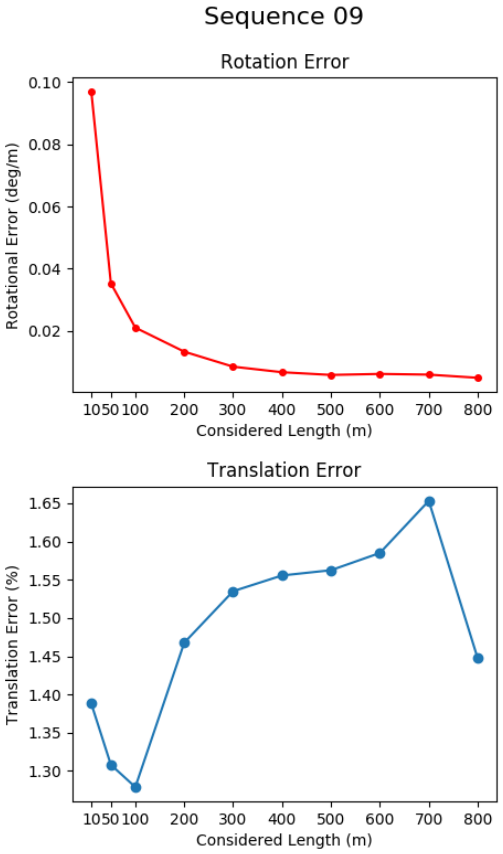
(d) XYZ RPY Error Histograms

Figure 5.6: Sequence 06 Results

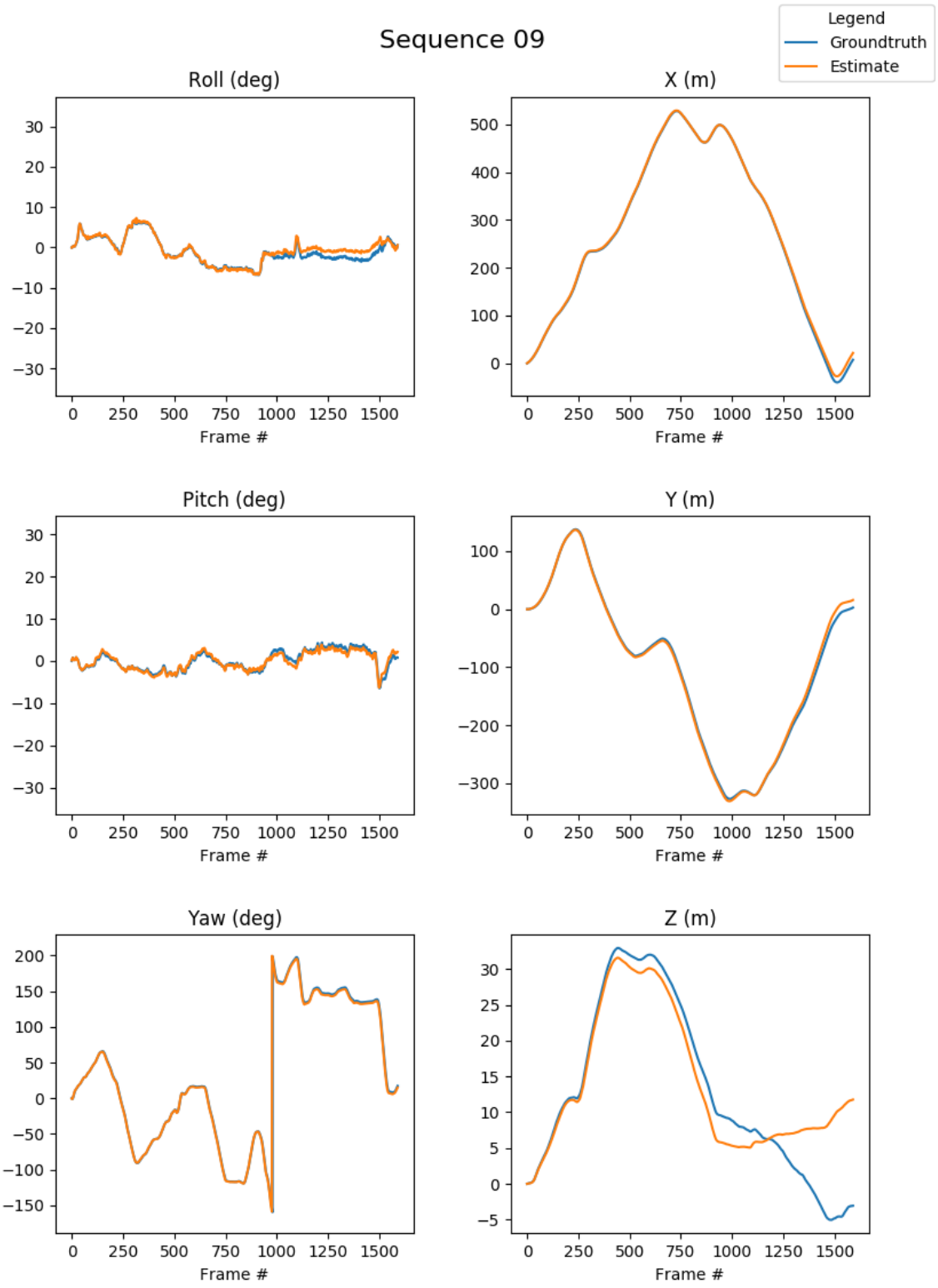
Sequence 09



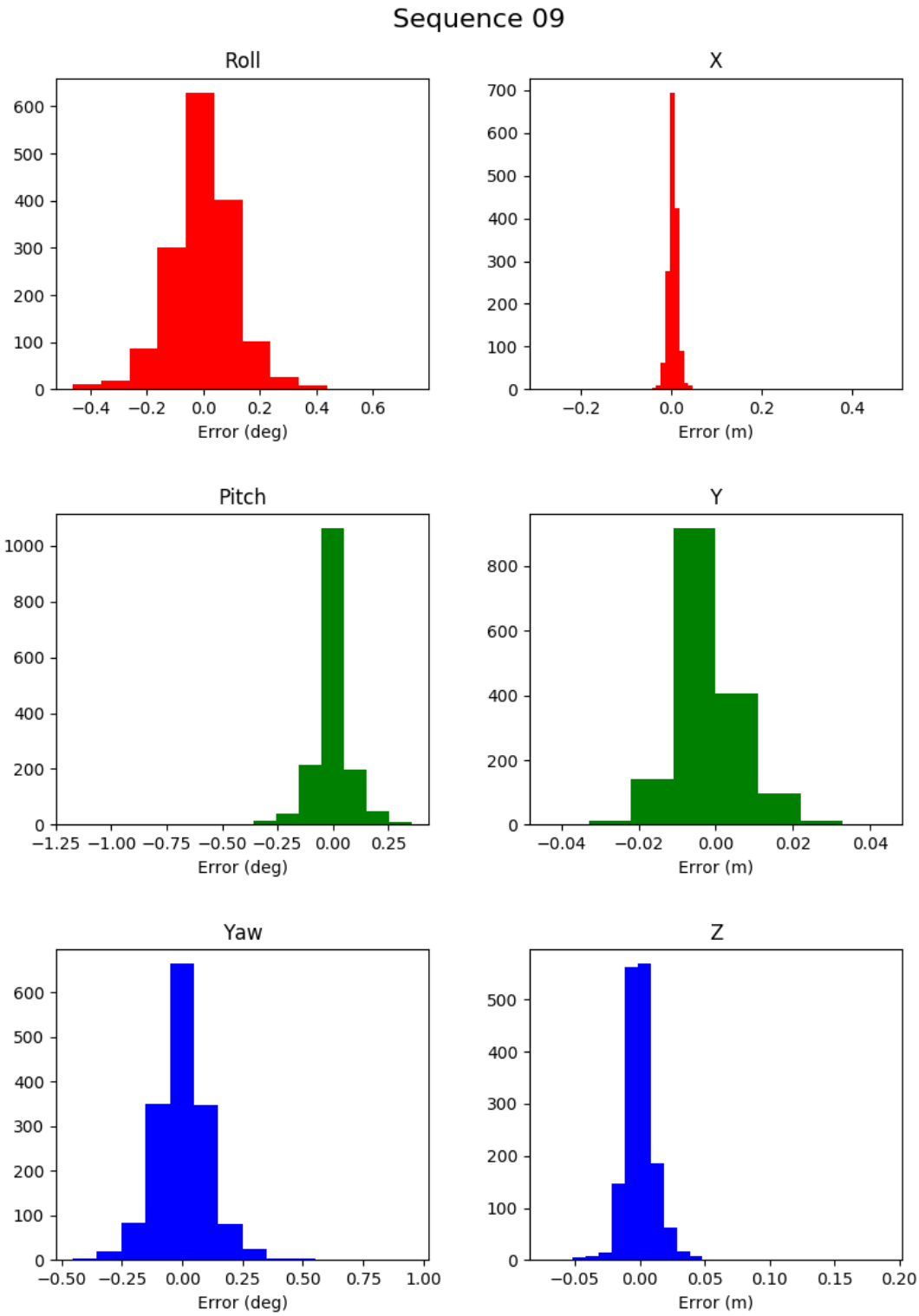
(a) Trajectory



(b) KITTI Evaluation



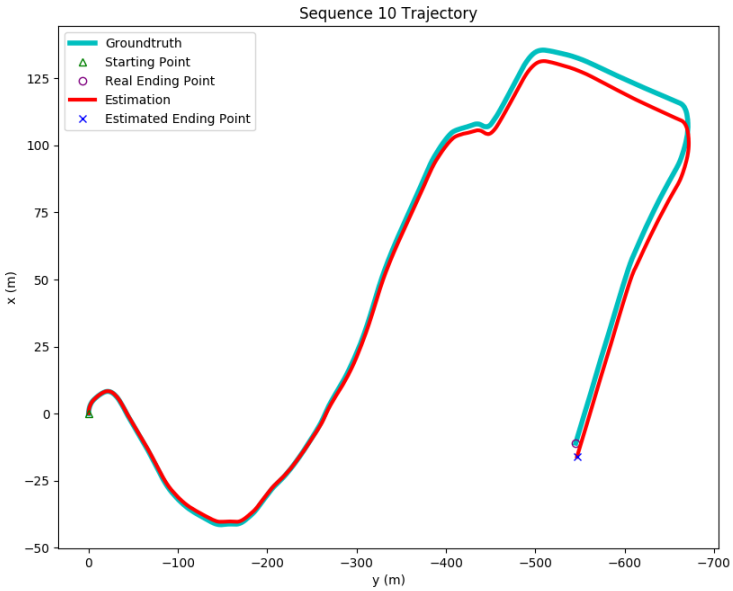
(c) XYZ RPY Plot



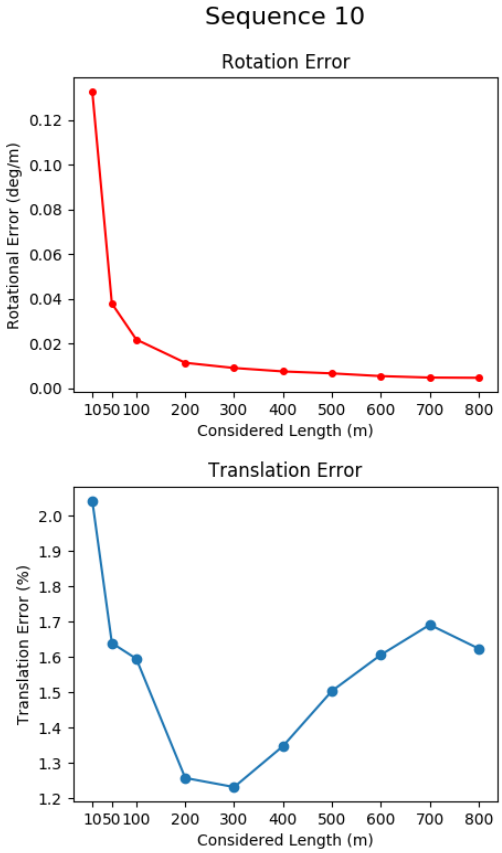
(d) XYZ RPY Error Histograms

Figure 5.7: Sequence 09 Results

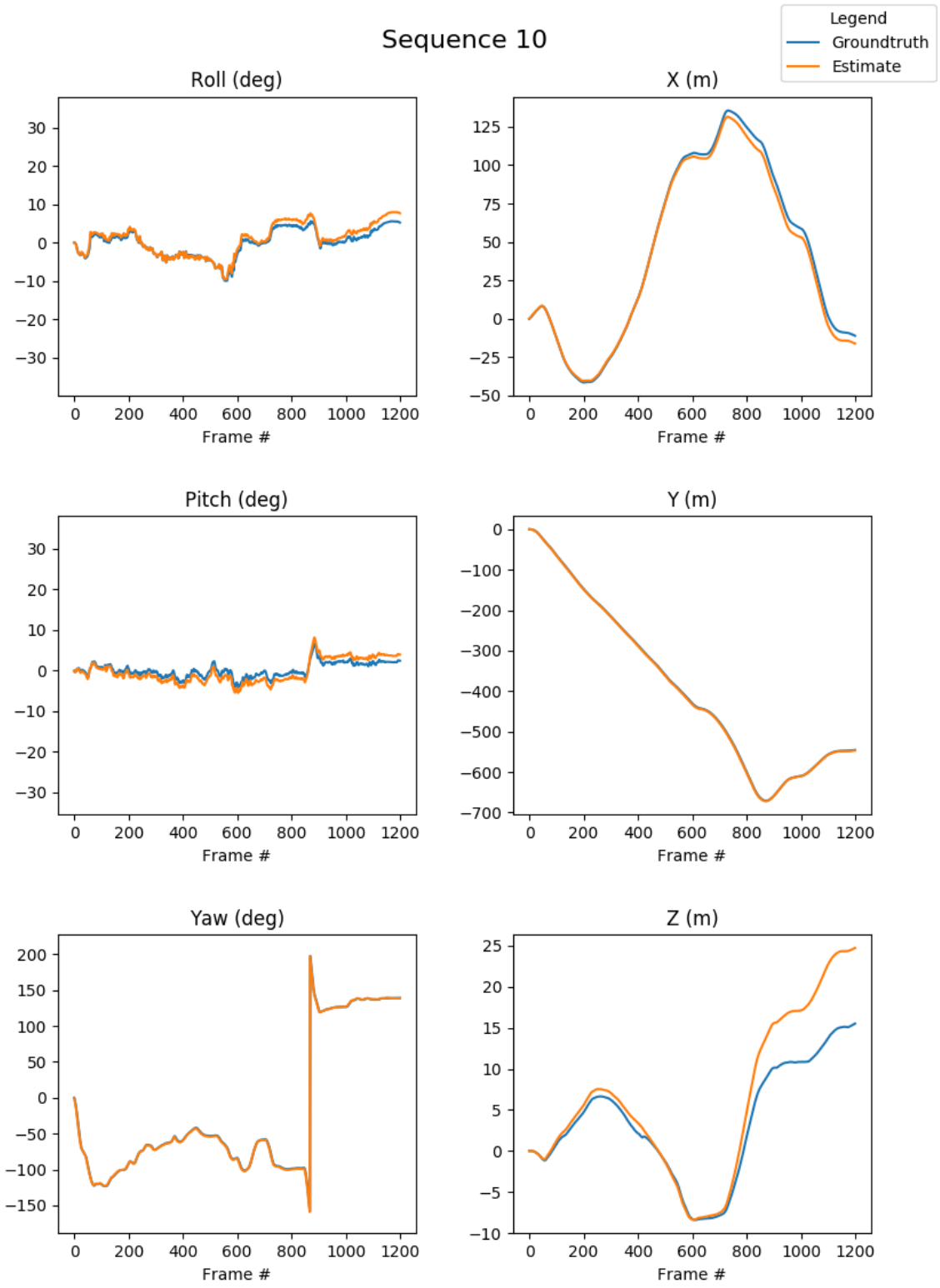
Sequence 10



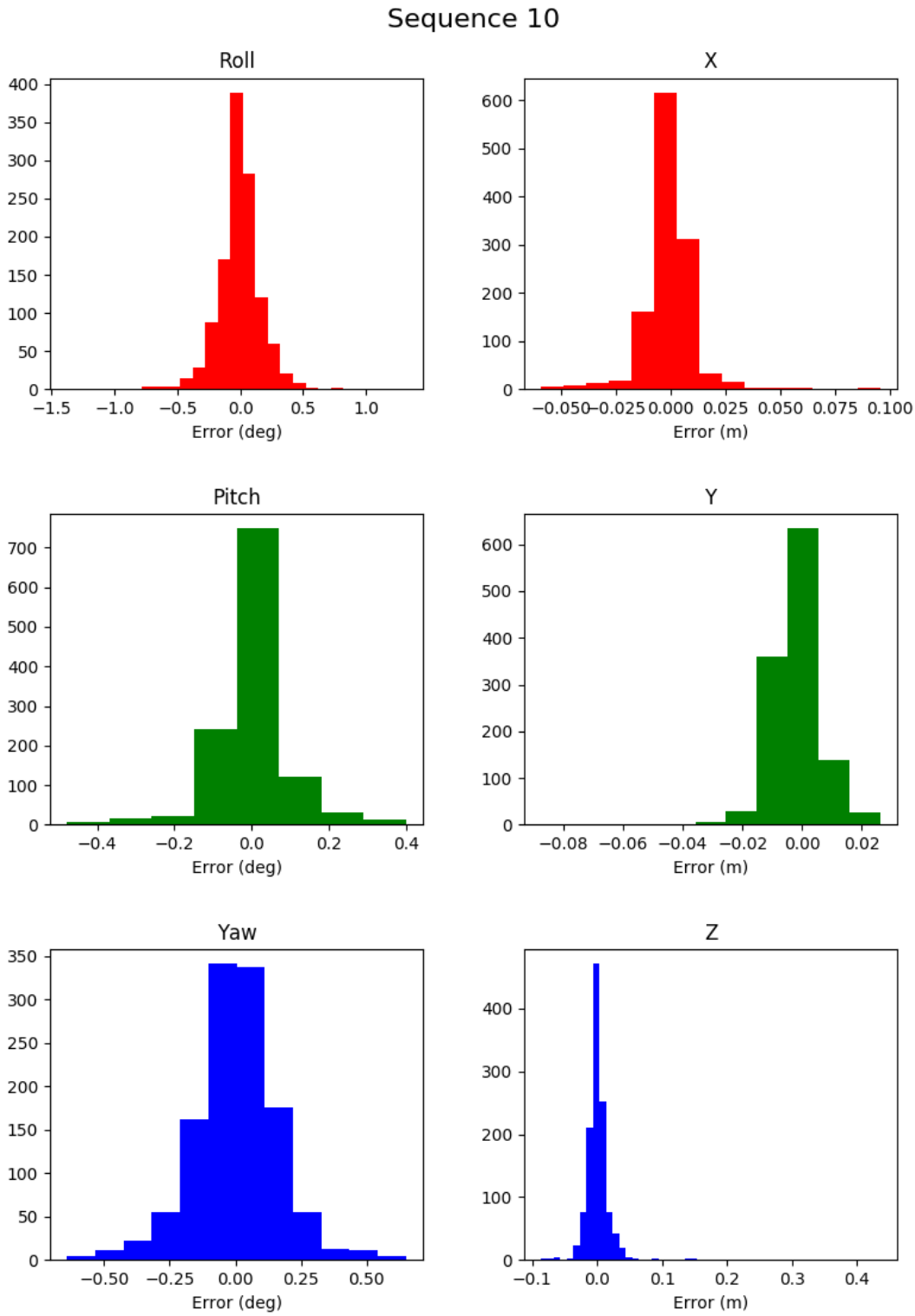
(a) Trajectory



(b) KITTI Evaluation



(c) XYZ RPY Plot



(d) XYZ RPY Error Histograms

Figure 5.8: Sequence 10 Results

Averaging among the sequences the obtained rotation error and translation error of the KITTI sequences, we notice that rotational-wise the system is stable as the rotational error converges to a value of almost 0.008 degrees. See Figure 5.9. The transnational component on the other hand does not converge. The average error for the translation however is 1.853%.

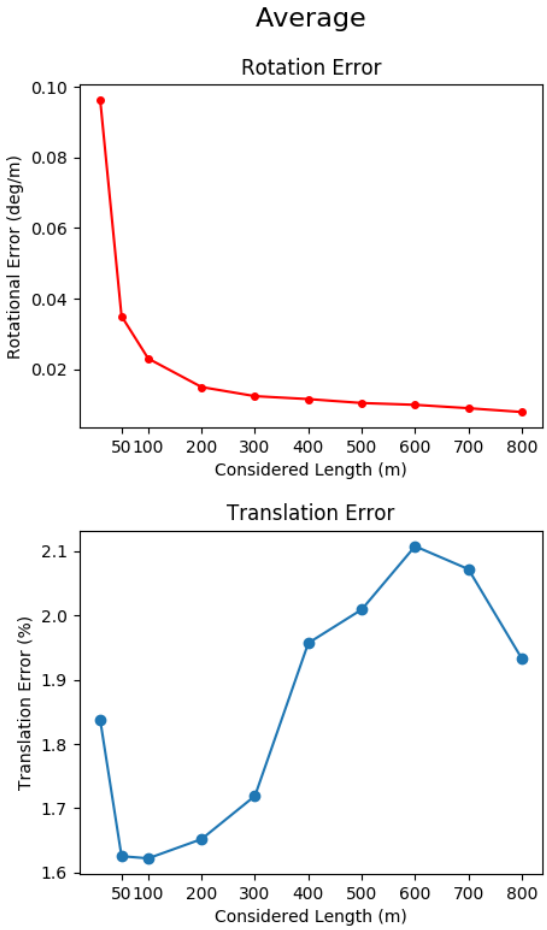


Figure 5.9: Average Error on KITTI datasets (sequences 00 to 10)

5.3 ARTEMIPS

In order to test the algorithm on the ARTEMIPS car, a test run was done on the test-track of the Cité de l'Automobile in Mulhouse - France.

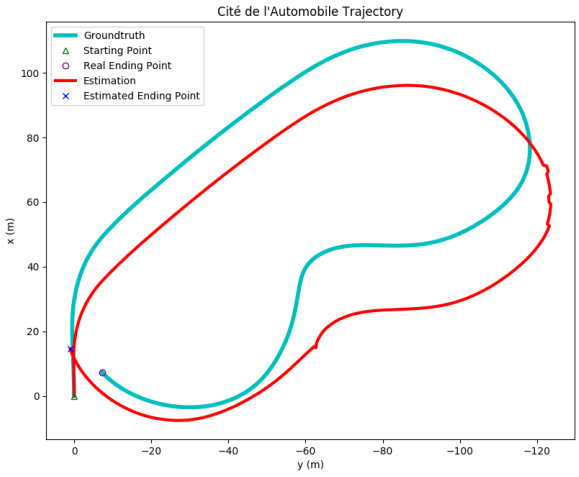
The results in this case weren't as good as the KITTI dataset.

The errors of the system on ARTEMIPS were higher for some reasons. One of the main reasons is the camera used. The camera used in the KITTI dataset was a wide-angle 1.4 megapixel camera. The camera used on the car is not optimized for vehicle use. The manufacturers recommend it for some applications such as

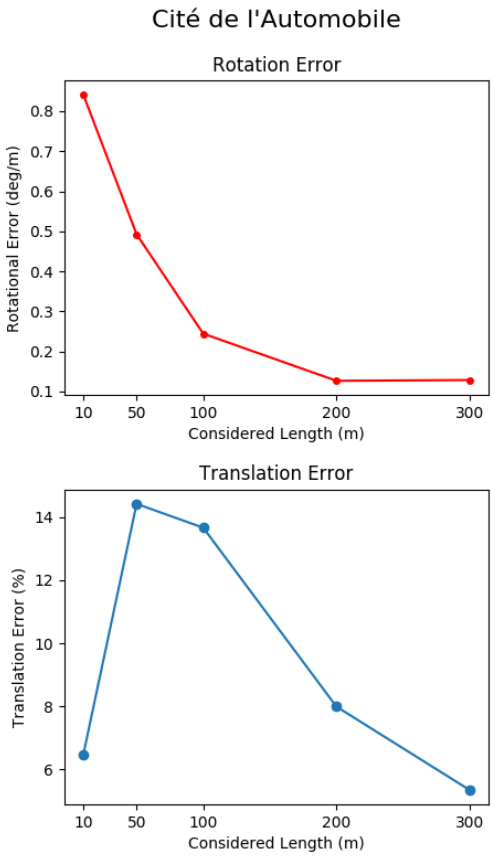
Figure 5.10: Windshield reflection

food inspection and semiconductor inspection. Also the need of a polarizing filter is needed to account for the reflection on the windshield due to the the sun (see Figure 5.10). Such reflections lead to features with non-changing pixel locations. These *false features* lead to estimation errors.

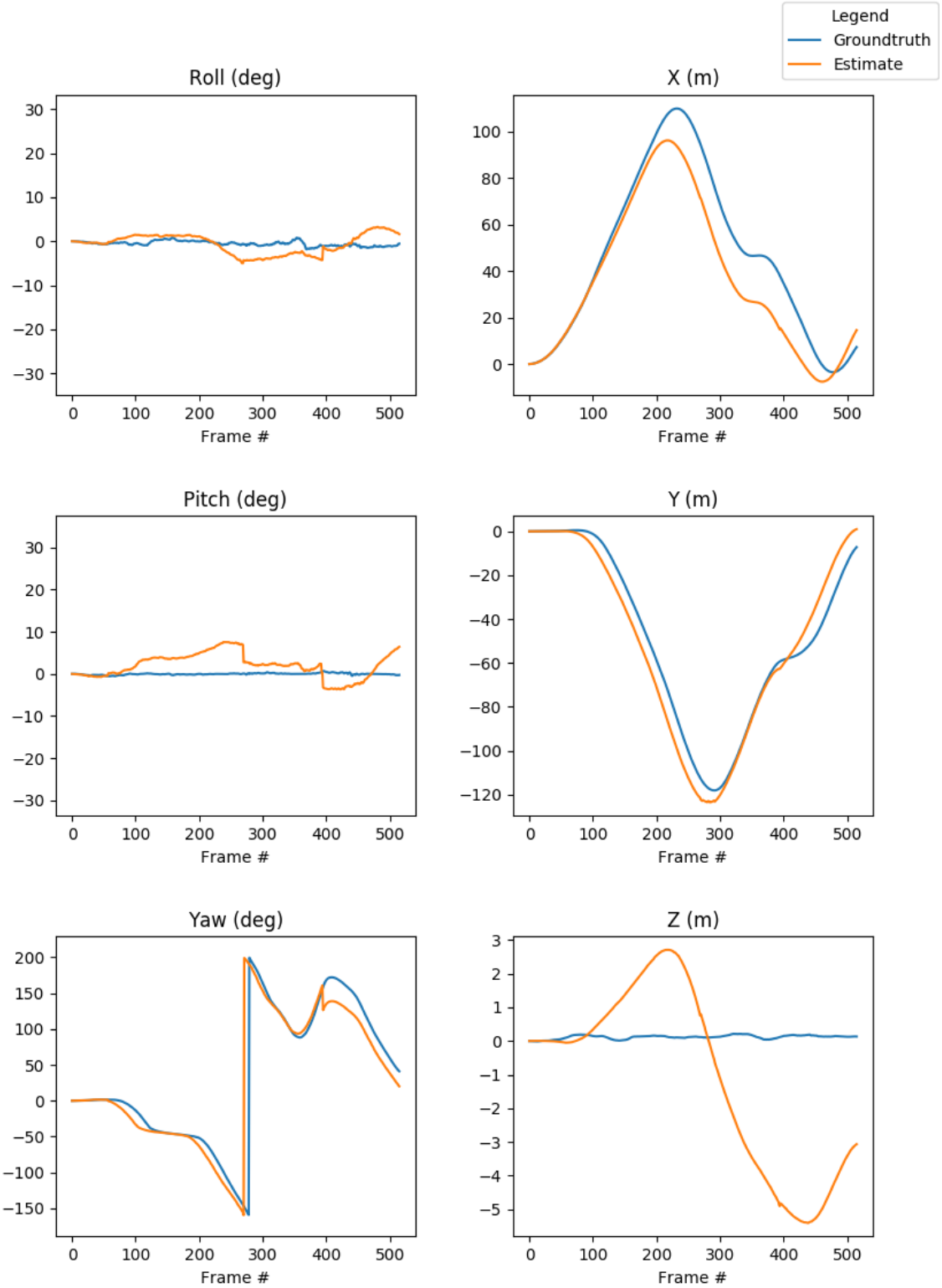
The algorithm also fails in the case of high speed turns. This was seen most evidently in sequence 06 (see Figure 5.6a) and during the test drive at the Cité d'Automobile (figure 5.11a).



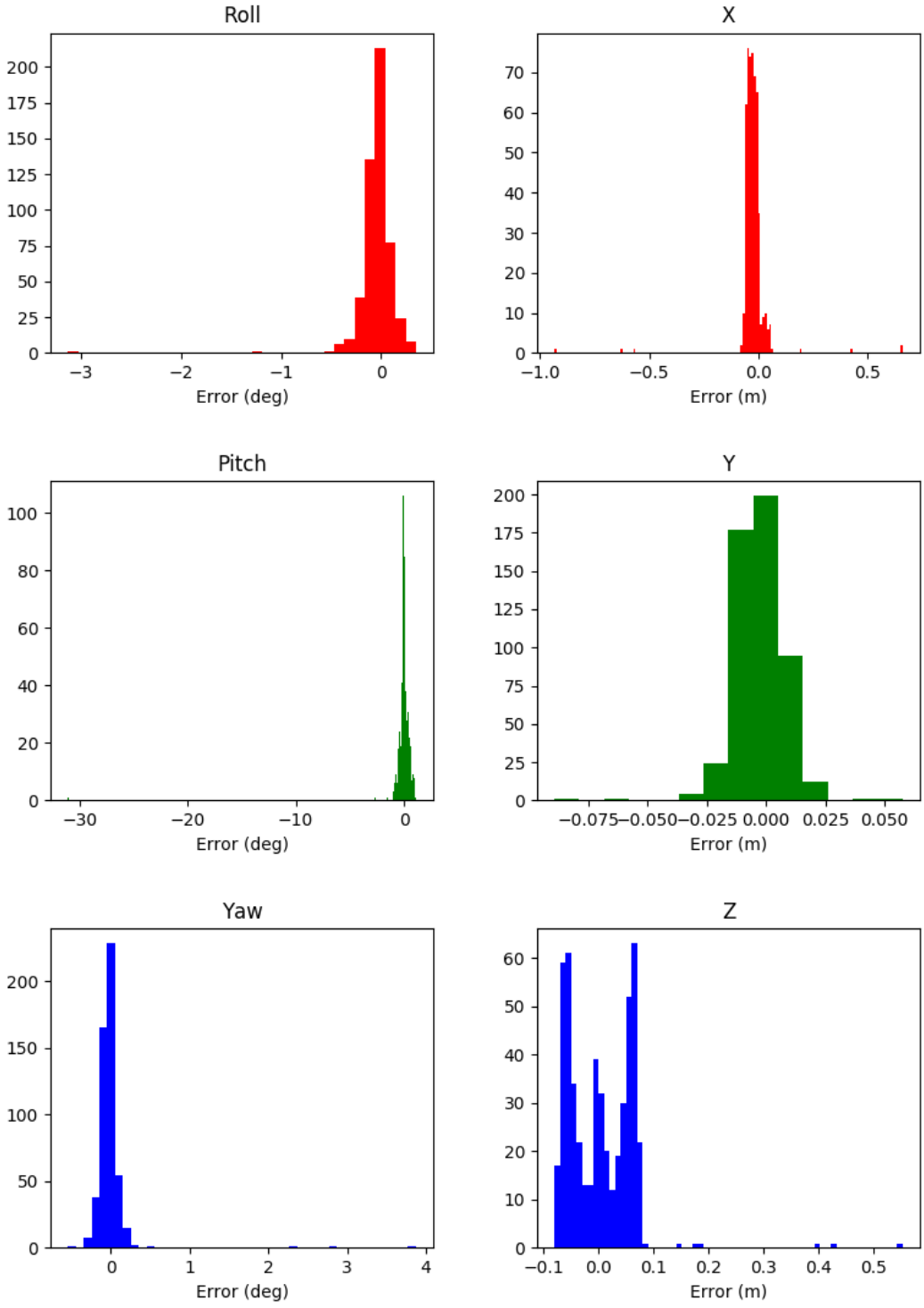
(a) Trajectory



(b) KITTI Evaluation



(c) XYZ RPY Plot



(d) XYZ RPY Error Histograms

Figure 5.11: Cité de l'Automobile Sequence Results

Chapter 6

Discussion

6.1 Recap

The aim of this project was to develop a monocular visual odometry system for ARTEMIPS, an self-driving car. This VO system will allow the vehicle to localize itself in the absence of GPS signal or its inaccuracy.

The developed system was built on the idea of presence of planar structures while driving, which affects the accuracy of estimation if essential matrix was used. The developed system is homography-based VO system which, in the case of non-planar scenes, allows the recovery of the essential matrix using the Parallax Beam paradigm.

6.2 Outlook and Future Work

The monocular VO system developed has provided satisfactory results. The system however still needs to be improved. As seen before, lighting conditions affect the outcome of our system. Image pre-processing can help mitigate their effect.

Moreover, other sensors such as GPS and IMU can be fused with the VO system. Also the use of the IMU can help address the scale ambiguity problem. As previously stated in the epipolar geometry section, the translation component extracted from the essential matrix is *up to a scale* meaning that it is a unit vector. Several solutions based on deep learning or prior knowledge exist.

A small test has been conducted to try to extract the scale from the homography. The decomposition of the homography matrix (as per OpenCV function) leads to a

non-normalized translation vector. The figure 6.1 shows the norm of the translation vectors of each estimation as a function of frame number. The yellow line shows the ground truth scale. The blue line shows the output of a rolling average of window of size 30 frames on the estimated scale extracted from the homography translation vector. The red line shows an exponential weighted moving average of span 30 frames applied to the estimated scale. Note that the estimated values have been multiplied by a scale of 2 which was set empirically. Further work can be done regarding scale estimation.

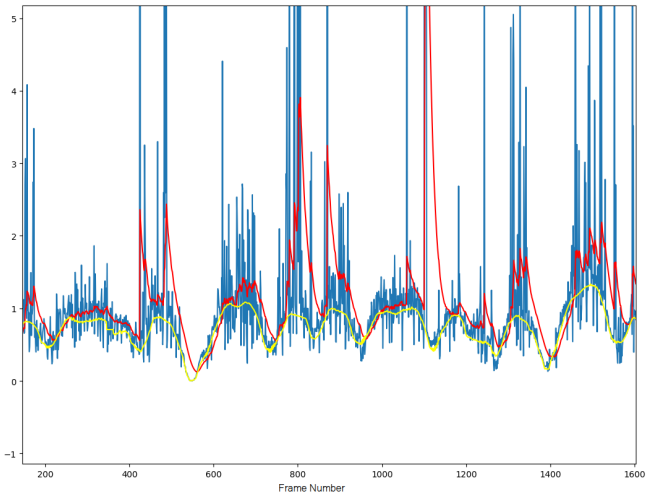


Figure 6.1: Homography Scale Estimation

Further improvements can be done by utilizing *Bundle Adjustment*. In VO, individual frame-by-frame transformations are concatenated to form the current pose of the vehicle [12]. Considering the presence of error in each transformation, we can say that the errors in the current pose depend on errors of previous transformations. As the current pose is formed, the error from past transformation propagates to the current pose and is accumulated. This accumulation is called *drift*. Bundle Adjustment is a procedure in which camera poses are optimized by taking into consideration previous camera poses.

Bibliography

[1] Rahul Kumar and Richa. *Autonomous Vehicle Market by Level of Automation (Level 3, Level 4, and Level 5) and Component (Hardware, Software, and Service) and Application (Civil, Robo Taxi, Self-driving Bus, Ride Share, Self-driving Truck, and Ride Hail) - Global Opportunity Analysis and Industry Forecast, 2019-2026*. 2018.

[2] Eric Krotkov. *Mobile robot localization using a single image*. 1989.

[3] E. Royer M. Lhuillier M. Dhome J.M. Lavest. "Monocular Vision for Mobile Robot Localization and Autonomous Navigation". In: *International Journal of Computer Vision* 74.3 (Sept. 2007), pp. 237–260.

[4] AutoPilot Review. *Ex-Uber Engineer Completes Coast-to-Coast Self-Driving Trip*.

[5] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2000).

[6] Y. Matsushita Z. Zhang and Yi Ma. "Camera calibration with lens distortion from low-rank textures". In: *In Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition* (2011), pp. 2321–2328.

[7] X. Armangue J. Salvi and J. Batlle. "A comparative review of camera calibrating methodswith accuracy evaluation". In: (2001).

[8] F. Fraundorfer D. Scaramuzza. "Visual Odometry: Part I - The First 30 Years and Fundamentals". In: *IEEE Robotics and Automation Magazine* 18 (2011).

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

[10] R. Raguram C. Wu Y.H. Jen E. Dunn B. Clipp S. Lazebnik J.M. Frahm P. Georgel D. Gallup T. Johnson and M. Pollefeys. "Building Rome on a cloudless day". In: *Proc. European Conf. Computer Vision* (2010), pp. 368–381.

[11] H. Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. 1980.

[12] D. Scaramuzza F. Fraundorfer. "Visual Odometry: Part II - Matching, Robustness, and Applications". In: *IEEE Robotics and Automation Magazine* 19 (2011).

[13] M. Muja and D. G. Lowe. "Scalable Nearest Neighbor Algorithms for High Dimensional Data". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 36 (2014).

[14] D. Nistér. "An Efficient Solution to the Five-Point Relative Pose Problem". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 26.6 (June 2004), pp. 756–777. ISSN: 0162-8828.

[15] Ezio Malis and Manuel Vargas. "Deeper understanding of the homography decomposition for vision-based control". In: (Jan. 2007).

[16] M. Rebert D. Monnin S. Bazeille C. Cudel. "Parallax beam: a vision-based motion estimation method robust to nearly planar scenes". In: (2019).

[17] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.*

[18] Ezio Malis and Manuel Vargas. *Deeper understanding of the homography decomposition for vision-based control.* Research Report RR-6303. INRIA, 2007, p. 90. URL: `https://hal.inria.fr/inria-00174036`.