Monocular Visual Odometry for Underwater Navigation

Erlend Nodeland Eriksen

Supervisor:
Annette Stahl

In cooperation with Blueye Robotics AS,
with supervisor:
Johannes Schrimpf

Norwegian University of Science and Technology
TTK4900 – Master thesis, Cybernetics and Robotics
Trondheim, January 2020

Abstract

In this thesis we propose a visual odometry algorithm for underwater navigation using a monocular camera. We cover the main mathematical concepts needed in order to estimate camera motion from consecutive images, as well as techniques that increase the accuracy of the motion estimates and reduce the computational burden of obtaining them. We then describe how to implement the proposed algorithm using these concepts and techniques. Finally, we evaluate the performance of the proposed algorithm on selected underwater video sequences, and we discuss what could be done in order to improve its robustness and computational performance further.

Abstrakt

I denne masteroppgaven foreslår vi en visuell odometri algorime for undervannsnavigasjon ved hjelp av et monokulært kamera. Vi dekker de matematiske konseptene som trengs for å estimere kamera bevegelse fra påfølgende bilder, i tillegg til teknikker som øker nøyaktigheten til bevegelses-estimatene og som reduserer den beregningsmessige byrden for å finne dem. Deretter beskriver vi hvordan man kan implementere den foreslåtte algoritmen ved hjelp av disse konseptene og teknikkene. Til slutt evaluerer vi ytelsen til algoritmen på utvalgte video serier, og vi diskuterer hva som kan gjøres for å forbedre robustheten og beregningsytelsen dens ytterligere.

Contents

List of Acronyms

CMOS = Complementary Metal–Oxide–Semiconductor

CPU = Central Processing Unit

DoF = Degrees of Freedom

DSP = Digital Signal Processor

EKF = Extended Kalman Filter

FPGA = Field-Programmable Gate Array

GPGPU = General Purpose Graphics Processing Unit

GPU = Graphics Processing Unit

IMU = Inertial Measurement Unit

KLT = Kanade-Lucas-Tomasi

LBP = Local Binary Patterns

MAD = Median Absolute Deviation

SIMD = Single Instruction, Multiple Data

SLAM = Simultaneous Localization And Mapping

SSD = Sum of Squared Differences

VO = Visual Odometry

List of Figures

Chapter 1: Introduction

This chapter will describe the context of the thesis.

Section 1.1 gives the reason for why the subject matter of the thesis was chosen to be what it is. It sets up the problem that we want to solve. In section 1.2 we describe the goals that we want to accomplish throughout the thesis. Section 1.3 describes the main innovation that has been done. Section 1.4 gives an overview of the thesis.

1.1 Motivation

Visual odometry has been explored extensively for over water applications like UAV navigation. However, many popular visual odometry algorithms have poor performance underwater[50] because of light attenuation effects, marine snow etc. In addition to this, many popular VO algorithms make use of computationally intensive operations like descriptor based feature matching or dense image alignment, requiring the use of powerful computing resources.

As is mentioned in section 2.3, most of the existing underwater VO algorithms make use of stereo cameras. This disqualifies them for use on platforms that only feature a monocular camera.

On the subject of monocular cameras, the last few years have seen a growth in the market for underwater drones, which are camera equipped ROVs that are smaller, cheaper and easier to use than more traditional ROVs. These drones do not have the same abundance in sensors and computing power that large research and industry ROVs have, so they need to consider VO solutions that require less resources, both in terms of sensors and computing power. They rarely have support for stereo cameras, and they could benefit from visual odometry in for instance marine inspection, which is a common use case.

All this considered, we see the need for a monocular VO algorithm which is robust against the problems introduced by underwater scenes as well as being computationally simple.

1.2 Aim of Study

We present a monocular visual odometry algorithm based on alignment of image and descriptor patches. The algorithm is a combination of the frontend of the semi-direct VO algorithm SVO[10], with the added modification of aligning geometrically differentiable descriptors called bitplanes[2]. The output of the algorithm is a live pose estimate and a trajectory which is up to scale, meaning that the scale of the trajectory is not part of the estimate.

Throughout the thesis, we will endeavour to explain and show experimentally how the proposed algorithm is robust to problems introduced by underwater scenes. We will also show failure scenarios and how the algorithm can be improved.

We limit the scope of the thesis by focusing on the theory and choices made for the front-end of the VO algorithm, ie. the image alignment and the steps leading up to it. We leave the backend, or mapping part of the algorithm mostly untouched. Note that this choice is solely for the purpose of limiting the scope of the thesis, and it doesn't mean there are no choices for the backend that affect the underwater performance of the algorithm.

1.3 Contribution

The addition of bitplanes to the SVO frontend represents the main moment of innovation in the proposed algorithm. The computational and conceptual simplicity of sparse image alignment implicitly satisifies one of the requirements stated in the motivation section. Robustness to underwater distortion effects is the other requirement, and as we will demonstrate, it is satisfied to some degree by the use of bitplanes for image alignment.

For evaluation, the proposed algorithm is implemented in the C++ programming language for GNU/Linux based operating systems. The C++ library Eigen is used for linear algebra operations, along with the Eigen compatible Lie group implementation called Sophus. The FAST implementation from Edward Rostens[44][45] computer vision library libCVD is used for corner detection.

Online visualization is done using OpenGL, and image capture is done using the V4l2 driver along with libjpeg-turbo for image decoding, unless hardware decoding is available. In addition, the popular video streaming framework GStreamer is used for remote visualization when executing on embedded platforms. GNU Octave, which is a clone of the Matlab programming language, is used for offline plotting and analysis.

1.4 Outline

Relevant literature is presented in chapter 2. We also give a brief overview of different approaches to visual odometry.

Chapter 3 covers the theoretical background for the methods used in the proposed algorithm. Most of the chapter is dedicated to explaining the tools and concepts needed in order to use non-linear optimization to find camera motion.

Chapter 4 goes through each step of the proposed algorithm. The expressions for the partial derivatives needed for image-alignment are derived in section 4.4.1.

Chapter 5 evaluates the performance of the proposed algorithm. We show plots of trajectory estimates and residuals generated from small pertubations in the motion estimate.

Chapter 6 concludes the thesis. It sums up the work done and the discoveries that have been made.

Chapter 2: Literature Review

In this chapter we review relevant existing literature.

In section 2.1 we discuss the main highlights of the evolution of visual odometry. In section 2.2 we give an overview of the main ways of distinguishing visual odometry methods. Section 2.3 sums up the brief evolution of visual odometry methods specifically taylored to underwater environments.

2.1 Evolution of Visual Odometry Methods

Early computer based visual odometry methods typically made use of corner detection algorithms like the Harris corner detector[25]. After detection, the corners were typically tracked in consecutive frames with algorithms like KLT[48].

Later methods used robust feature descriptors like SURF[5] and ORB[46] in order to find feature correspondences.

In order to estimate the pose of the camera, early methods often used filtering as in EKF-SLAM[38]. Later, more and more methods, including ORB-SLAM[39] and PTAM[28], have been using bundle adjustment, also called "smoothing", instead of filtering.

PTAM[28] also introduced a separation and decoupling of a frontend and backend in visual odometry methods, where the frontend is responsible for feature tracking and motion estimation while the backend is responsible for depth estimation or for refining the feature map and pose. The backend runs in a separate thread from the frontend and is decoupled from hard real time contraints. This separation has proven useful, and is employed in multiple popular methods today, like SVO[10].

While early visual odometry methods were mostly indirect, meaning that they extract features like corners or lines from the images, there have been multiple "direct" methods in recent years, like DTAM[40], that operate directly on the image intensities themselves.

2.2 Taxonomy of Visual Odometry Methods

2.2.1 Sparse and Dense Methods

Visual odometry methods are divided into sparse and dense methods.

Dense methods use all of the image in their computations. These methods have high computational cost because of the large amount of data, and they typically need massively parallell implementations in order to run in real time. Dense methods can potentially be used for dense reconstruction of scenes.

Sparse methods only consider parts of the image, reducing the computational cost. Sparse methods usually tries to find parts of the image that are easy to track.

2.2.2 Indirect Methods

Indirect methods extract interest points called features from salient image regions, discarding the rest of the image. The features are locations in the image with a high intensity gradient, typically corners or lines.

The idea behind this approach is that the features will contain most of the valuable information in the image since they typically have a more distinct look compared to the image regions that are not detected as features. Thus a set of features form a consise representation of image regions that are easy to find again in consequtive frames in order to determine the camera motion.

The first step in indirect methods is feature detection, for which there exists a number of algorithms including FAST[44]. Next a lot of indirect methods will compute a feature descriptor, like for instance BRIEF[6], for each detected feature. These descriptors are typically binary vectors that encode a description of the image region around the feature that is supposed to be invariant to conditions like change in lighting as well as rotation in some cases.

When a second frame is received, indirect methods will try to find the location of the features extraced in the last frame in the new frame. This can either be done through optical flow like in KLT tracking[48], or through feature correspondences using feature descriptors. Finding feature correspondences is a difficult process, and outlier rejection methods like RANSAC[21] are needed to handle false correspondences. After feature correspondences are found, the motion can be calculated using epipolar geometry.

It is also possible to find the motion through minimizing the sum of squared differences (SSD) of the reprojection error of the feature matches. The reprojection error of matching pair of feature observations, where the newest observation was observed in the newest frame and the previous observation was observed in a previous frame, is the geometric error corresponding to the distance between the newest observation and the previous observation projected into the newest frame. This is shown in equation (2.2.1), where $\boldsymbol{\theta}$ is a parameterization of the motion between the images, $\boldsymbol{y_i}$ is the newest observation of a feature, $\boldsymbol{x_i}$ is a previous observation of the same feature, w is the reprojection function from equation (3.4.10), and d is the distance between the feature $\boldsymbol{x_i}$ and the camera.

$$E(p) = \sum_i ||\boldsymbol{y_i} - w(\boldsymbol{x_i}, \boldsymbol{\theta}, d)||^2 \tag{2.2.1}$$

2.2.3 Direct Methods

Direct methods operate directly on the image intensities without going through the feature extraction process of indirect methods. Therefore they typically process information that indirect methods discard, like low gradient image regions. This can potentially help increase the accuracy of the visual odometry, but it also means that direct methods usually process more data than indirect methods, increasing the computational cost of the algorithms.

In order to find the motion between a reference image (or alternatively a patch in the reference image referred to as a template) and the current image, direct methods usually minimize the "photometric error" between the images. The photometric error is defined as the sum of squared differences between an image I and a template image T transformated by a warp. It is shown in equation (2.2.2), where T and I are the image functions respectively for the reference and the current image, w is the warp function, $\boldsymbol{\theta}$ is a parameterization of the motion between the images, and $\boldsymbol{x}$ is an image location in T.

$$E(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \Big[I(\omega(\boldsymbol{x}, \boldsymbol{\theta})) - T(\boldsymbol{x}) \Big]^2 \tag{2.2.2}$$

To reduce the amount of information to process, sparse direct methods only process the image regions that are salient according to some metric. For instance Chistensen and Hebert[9] extracts edges from a canny[7] edge detector. Also, SVO[10] extracts image regions using FAST[44][45] and processes the sparse image patches around the features directly afterwards, so it could fit into this category. However SVO is instead described as semi-direct because after the direct image alignment step, it proceeds to do indirect bundle adjustment.

Some direct methods don't use raw image intensities[1]. In place of the image intensities, they use different kinds of geometrically differentiable binary descriptors. We discuss such a method in section 3.7.

2.3 Underwater Visual Odometry Methods

The use of visual odometry for underwater navigation has been researched to an increasing degree in the last few years as an alternative to expensive EKF based solutions that combine inertial measurement units, Doppler Velocity Logs and sonars. Many methods target accurate mapping of sea-floors or caves since this is demanded by industry and since cameras potentially offer more information than sonars and at lower cost.

When it comes to underwater navigation, sonar based methods have traditionally been preferred because they do not suffer from the visual degradation observed in underwater images. However, the information from sonars are difficult to analyze compared to images, and at close range, sonar is inaccurate[43].

Eusice et al.[16] were among the first to experiment with underwater visual odometry, using an indirect approach in conjunction with inertial sensors. Their approach was soon extended to be used on stereo cameras[26]. Several stereo camera based methods were developed[33]. In general, the difficulties introduced by underwater vision lead most VO-methods to opt for stereo cameras, leaving the literature for monocular solutions relatively sparse up until recently.

As for more recent research, Weidner et al.[50] found that state of the art methods commonly used in the air do not perform satisfactory under water. They instead used stereo cameras and strategic illumination to solve some of the problems introduced by turbid water.

Interestingly, Nawaf et al.[37] use a neural network in their indirect VO-algorithm in order to estimate the uncertainty in their pose estimate since this is complicated to model in an underwater context due to water degradation. Silveira et al.[47] also make use of neural networks. They use them in an attempt to mimic biological perception by having sensor data and position estimates map into neurons.

Ozog et al.[42] performs SLAM around a ship hull by assuming the hull surface to be locally planar. This assumption is also used for hull inspection in Kim and Eustice[27], in conjunction with an odometry approach which switches between a homography and an epipolar geometry based model, depending on the hull geometry.

Direct VO methods typically suffer more from the distortion and degradation effects experienced underwater because they operate on direct light intensities[36]. This is likely why there are few direct VO methods developed especially for underwater use.

Ferrera et al.[20] indicated that optical flow based tracking performs better than descriptor based methods underwater. The optical flow algorithm they used was the KLT (Kanade-Lucas-Tomasi) feature tracker, which is direct in nature even though it is commonly used in indirect methods.

Chapter 3: Theory

In this chapter we present the main theoretical background needed in order to understand the proposed solution to the monocular visual odometry problem for underwater scenes.

3.1 Feature Detection

Feature detection is the process of extracting salient image locations from an image. A good feature detection algorithm will find features that are easy to track in subsequent image frames because they stand out in contrast to their surroundings.

Feature detection is also referred to interchangeably as corner detection because corners usually make decent features.

3.1.1 FAST

The FAST feature detection algorithm[44][45] is popular because of its computational efficiency and theoretical simplicity.

FAST considers a set of 16 "test" pixels forming a circle around a feature candidate, as illustrated in figure 3.1. The basic principle is to check if there is a subset of at least N adjacent pixels from this set that have consistently larger or lower intensity than the candidate pixel. If such a subset is found, the candidate is chosen as a corner. N is typically set to be around 12 in the basic version of the algorithm. Also, a pixel intensity is considered lower or higher than the candidate pixel intensity only if it differs by more than a chosen threshold.

When choosing N=12, it is possible to use a high speed test to exclude non-corners. This test only considers 4 evenly spaced test pixels among the circle, for instance pixel 1, 5, 9 and 13 in figure 3.1. In order for the candidate to be chosen a corner, at least 3 of the 4 evenly distributed test pixels need to have consistently higher or lower intensity than the candidate. If this is not the case, the candidate can be dropped without further evaluaion of the test pixels.

Unfortunately, the high speed test can not easily be generalized to choices of N which are less than 12. Also, the choice of evenly distributed pixels and the order of their evaluation are probably going to be suboptimal in real applications, since they are chosen arbitrarily. To address these issues, the developers of FAST introduced a machine learning approach[45] in order to at all times evaluate the test pixels that give the most information about whether the candidate is a corner or not.

3.1.2 FAST-Score / Non-Maximum Suppression

In order to avoid having multiple overlapping or very close features, it is usual to remove all but the best features in local neighbourhoods.

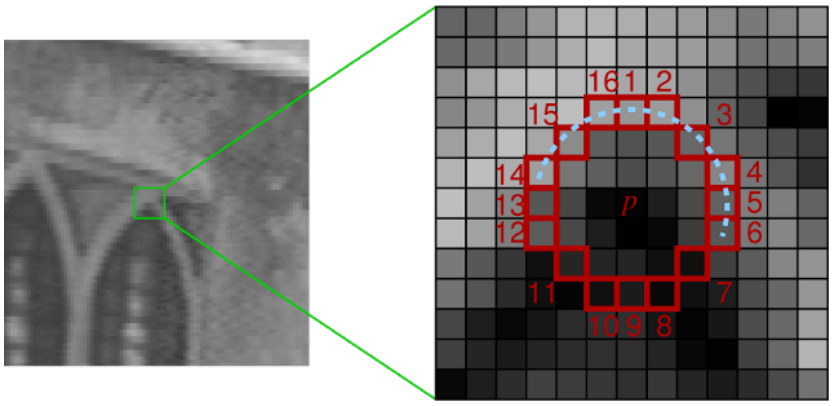


Figure 3.1: FAST in action. The Red boxes are the 16 test pixels surrounding the candidate p in the middle. The light blue half circle marks a subset of adjacent test pixels which all have larger intensity than p.

Whether a feature is good or bad is judged based on its FAST-score. The FAST-score can be defined in multiple ways, but it is common to set it to the sum of absolute differences between the intensities of the test pixels and the candidate.

3.2 3D Geometry

This section defines a mathematical representation of posisitions and motion as well as corresponding operations needed in our visual odometry approach.

3.2.1 Position

We define a position in 3d space by three real numbers, each equal to the position along each of our coordinate systems axes:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.2.1}$$

3.2.2 Translation

Translation is defined as element-wise addition:

$$\boldsymbol{x} + \boldsymbol{t} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix} \tag{3.2.2}$$

3.2.3 3D Rotation Representation

Three-dimensional Rotations can be parametrized in multiple ways. Notable representations are Euler angles, quaternions, angle-axis and rotation matrices[13]. In this thesis we will focus on Lie representations[14][4] of rotation and transformation as described in section 3.3.

Euler Angles

Euler angles is a compact way of representing a rotation. They consist of three angles. Rotation by euler angles is performed by consecutively rotating around each of the principal axis by each of the angles. This rotation can be performed by mapping each euler angle rotation to a rotation matrix using basic trigonometry. The order in which the angles are to be applied needs to be specified.

Calculations involving euler angles suffer from a problem called gimbal lock. This means that if we represent our estimate of consecutive camera orientations with euler angles and our camera rotates through a singularity of the euler angle representation, the estimate will loose a degree of freedom.

Angle-Axis / Rotation Vector

The angle-axis representation represents 3D-rotations using two quantities: The unit vector $\boldsymbol{a}$ indicates the direction of the axis of rotation, and the angle θ indicates the magnitude of the rotation around $\boldsymbol{a}$.

The representation is predicated on Eulers rotation throrem, which says that any sequence of rotations around multiple axis is equivalent to a single rotation around a fixed axis. Eulers rotaton theorem can be used to take three rotations around the three principal axis, and compose them so we get a single rotation around the axis given by $\boldsymbol{a}$.

We can multiply angle and axis to represent the rotation using only one quantity, called the rotation vector: $\boldsymbol{\omega} = \omega \boldsymbol{a}$. This representation is equivalent to the $\mathfrak{so}(3)$ representation which we will discuss in section 3.3.5.

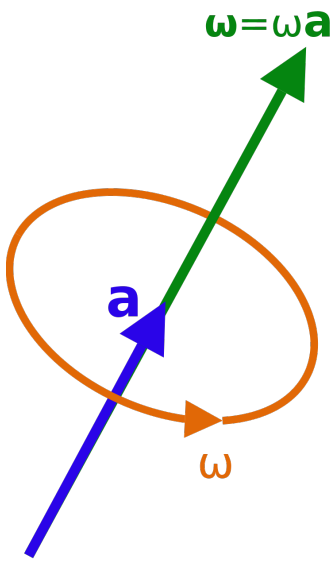


Figure 3.2: Illustration of the axis-angle and rotation vector representations of rotation

Rotation Matrix

A 3D rotation matrix is a three by three orthogonal matrix with unitary determinant.

The orthogonal requirement means that the set of columns and the set of rows in a rotation matrix each consist of orthonormal vectors, meaning that they are vectors that are both orthogonal to each other and that all have an absolute size of one.

The unitary determinant requirement disqualifies reflections, which are orthogonal matrices with determinant -1. Reflections are transformations that reflect sets of points through a plane, failing to preserve the relative orientation of the points.

The 3D rotation matrix representation is equivalent to the $SO(3)$ representation, which we define in section 3.3.4.

3.2.4 Homogenous Coordinates

Homogenous coordinates are also called projective coordinates because they are useful in situations where coordinates loose one degree of freedom through projection.

For a 3d point $\boldsymbol{x} = [x, y, z]^T$ we define the homogenous representation $\tilde{\boldsymbol{x}}$ of $\boldsymbol{x}$ by adding another coordinate equal to one:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \tilde{\boldsymbol{x}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2.3}$$

Vector geometry works similarly for homogenous coordinates as for ordinary coordinates. However, in homogenous coordinates we define the equivalence relation given by equation (3.2.4):

$$\tilde{\boldsymbol{x}} \sim \tilde{\boldsymbol{y}} \Leftrightarrow \tilde{\boldsymbol{x}} = \lambda \tilde{\boldsymbol{y}} \tag{3.2.4}$$

This means that all points that differ only by a scale factor λ in homogenous coordinates are considered equal. This leads naturally to a conversion process from homogenous to normal coordinates where we divide by the last homogenous coordinate before removing it:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} \sim \frac{1}{\tilde{w}} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ \tilde{z}/\tilde{w} \\ 1 \end{bmatrix} \Rightarrow \boldsymbol{x} = \begin{bmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ \tilde{z}/\tilde{w} \end{bmatrix} \tag{3.2.5}$$

3.2.5 Skew-Symmetric Matrices

Skew-symmetric matrices are matrices A for which $A^T = -A$. They are implicitly square matrices from this requirement. For a 3d vector $\boldsymbol{x}$ we can define a skew-symmetric matrix $[\boldsymbol{x}]_\times \in \mathbb{R}^{3x3}$ parametrized by $\boldsymbol{x}$:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow [\boldsymbol{x}]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{3.2.6}$$

The skew-symmetric matrix can be used in order to turn the vector cross product into matrix-vector multiplication:

$$\boldsymbol{x} \times \boldsymbol{y} = [\boldsymbol{x}]_\times \boldsymbol{y} \tag{3.2.7}$$

We can also calculate the skew-symmetric matrix parametrized by the cross product of two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^3$ in terms of the skew-symmetric matrices $[\boldsymbol{x}]_\times, [\boldsymbol{y}]_\times$:

$$[\boldsymbol{x} \times \boldsymbol{y}]_\times = [\boldsymbol{x}]_\times [\boldsymbol{y}]_\times - [\boldsymbol{y}]_\times [\boldsymbol{x}]_\times \tag{3.2.8}$$

This expression is referred to as the commutator of $[\boldsymbol{x}]_\times$ and $[\boldsymbol{y}]_\times$.

It is possible to reduce the cube of a skew-symmetric matrix:

$$[\boldsymbol{x}]_\times^3 = -(\boldsymbol{x}^T \boldsymbol{x}) \cdot [\boldsymbol{x}]_\times \tag{3.2.9}$$

This means that skew-symmetric matrices with arbitrary odd and even exponents, $2i+1, 2i+2 \in \mathbb{N}$ can be reduced as follows:

$$[\boldsymbol{x}]_\times^{2i+1} = -(1)^i \theta^{2i} [\boldsymbol{x}]_\times \tag{3.2.10}$$
$$[\boldsymbol{x}]_\times^{2i+2} = -(1)^i \theta^{2i} [\boldsymbol{x}]_\times^{2i} \tag{3.2.11}$$

Where θ is given by equation (3.2.12):

$$\theta = \sqrt{\boldsymbol{x}^T \boldsymbol{x}} \tag{3.2.12}$$

If $\boldsymbol{x}$ has size one, $\boldsymbol{x}^T \boldsymbol{x} = 1$, then we can reduce the square of a skew-symmetric matrix:

$$[\boldsymbol{x}]_\times^2 = \boldsymbol{x}\boldsymbol{x}^T - I, \ \boldsymbol{x}^T \boldsymbol{x} = 1 \tag{3.2.13}$$

3.3 Lie Groups and Lie Algebras

3.3.1 Lie Group Definition

Lie groups[14][4] are groups that are also smooth differential manifolds. The group requirement means that they consist of a set, G paired with a operator, $\bullet$ which together obey the four axioms for group theory given $x, y, z \in G$:

Closure:

$$x \bullet y \in G \tag{3.3.1}$$

Associativity:

$$(x \bullet y) \bullet z = x \bullet (y \bullet z) \tag{3.3.2}$$

Identity:

$$\exists e \forall x : x \bullet e = e \bullet x = x \tag{3.3.3}$$

Inverse:

$$\forall x \exists x^{-1} : x \bullet x^{-1} = x^{-1} \bullet x = e \tag{3.3.4}$$

As for the smooth differential manifold requirement for Lie Groups, this means that they locally resemble a vector space enough to allow one to do calculus using a differential structure which is globally consistent over the manifold.

3.3.2 Lie Algebra Definition

A Lie Algebra is a vector space $\mathfrak{g}$ with a non-associative operation $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ that satisfies the following axioms for $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathfrak{g}$ and $a, b \in \mathbb{F}$, where $\mathbb{F}$ is the field of $\mathfrak{g}$:

Bilinearity:

$$[a\boldsymbol{x} + b\boldsymbol{y}, \boldsymbol{z}] = a[\boldsymbol{x}, \boldsymbol{z}] + b[\boldsymbol{y}, \boldsymbol{z}] \tag{3.3.5}$$
$$[\boldsymbol{z}, a\boldsymbol{x} + b\boldsymbol{y}] = a[\boldsymbol{z}, \boldsymbol{x}] + b[\boldsymbol{z}, \boldsymbol{y}] \tag{3.3.6}$$

Alternativity:

$$[\boldsymbol{x}, \boldsymbol{x}] = 0 \tag{3.3.7}$$

The Jacobi Identity:

$$[\boldsymbol{x}, [\boldsymbol{y}, \boldsymbol{z}]] + [\boldsymbol{z}, [\boldsymbol{x}, \boldsymbol{y}]] + [\boldsymbol{y}, [\boldsymbol{z}, \boldsymbol{x}]] = 0 \tag{3.3.8}$$

A notable property arising from these axioms is anticommutativity:

Anticommutativity:

$$[\boldsymbol{x}, \boldsymbol{y}] = -[\boldsymbol{y}, \boldsymbol{x}] \tag{3.3.9}$$

For this thesis, we will be using the field of real numbers, $\mathbb{F} = \mathbb{R}$.

3.3.3 Lie Algebra derived from a Lie Group

The Lie group-Lie algebra correspondence allows us to study Lie Groups in terms of Lie algebras, which is advantageous since Lie algebras are linear, unlike Lie groups. The correspondance relies on Lie's third theorem[15], the homomorphisms theorem and the subgroups-subalgebras theorem[35].

For a given Lie Group G we can define its corresponding Lie Algebra $\mathfrak{g}$ as the tangent space around the identity, e of G. This tangent space is by definition the space of possible differential pertubations around e, and it can be parametrized using the basis elements $G_1, ..., G_k$, where k are the degrees of freedom of G. These elements are called the generators of $\mathfrak{g}$, and with them we can write the element $[\boldsymbol{x}] \in \mathfrak{g}$ as a linear combination:

$$[\boldsymbol{x}] = \sum_{i=1}^{k} x_i G_i, \quad \boldsymbol{x} = [x_1, ..., x_k]^T \in \mathfrak{g} \tag{3.3.10}$$

There are multiple corresponding pairs of Lie groups and Lie algebras that are useful in visual odometry, including $SO(3)$ with $\mathfrak{so}(3)$, $SE(3)$ with $\mathfrak{se}(3)$, and $SIM(3)$ with $\mathfrak{sim}(3)$. For the rest of the thesis, we will limit ourselves to $SO(3)$ with $\mathfrak{so}(3)$ and $SE(3)$ with $\mathfrak{se}(3)$.

3.3.4 $SO(3)$

We define the group $SO(3)$ as the set of 3D rotation matrices composed over matrix multiplication. Rotation matrices are matrices $C \in \mathbb{R}^{3\times3}$ that follow the following axioms given the identity matrix I:

Orthogonality:

$$C^T C = C C^T = I \tag{3.3.11}$$

Unitary determinant:

$$|C| = 1 \tag{3.3.12}$$

Under this definition we can show that $SO(3)$ satisfies all the Lie axioms from equation (3.3.1)-(3.3.4). Closure of $SO(3)$ follows from Eulers rotation theorem, which says a compounding of rotations can be replaced by a single rotation. Associativity follows from the matrix multiplication rules, although we will omit the derivation here. As for the existence of an identity and inverse element, we can derive this directly from equation (3.3.11) by setting $e = I$, $x = C$ and $C^{-1} = C^T$.

3.3.5 $\mathfrak{so}(3)$

We define the set of the Lie algebra $\mathfrak{so}(3)$ as the set of skew-symmetric matrices $[\boldsymbol{\omega}]_\times,\ \boldsymbol{\omega} \in \mathbb{R}^3$, where $[\cdot]_\times$ is the skew-symmetric matrix parameterization given in section 3.2.5. Then we define the operation $[\cdot,\cdot]_\times\ :\ \mathfrak{so}(3), \mathfrak{so}(3) \to \mathfrak{so}(3)$ by using the skew-symmetric representation of cross-products from equation (3.2.8):

$$[[\boldsymbol{\omega}_1]_\times, [\boldsymbol{\omega}_2]_\times]_\times = [\boldsymbol{\omega}_1 \times \boldsymbol{\omega}_2]_\times = [\boldsymbol{\omega}_1]_\times [\boldsymbol{\omega}_2]_\times - [\boldsymbol{\omega}_2]_\times [\boldsymbol{\omega}_1]_\times \tag{3.3.13}$$

Since the skew symmetric matrix representation $[\cdot]_\times$ is a unique parameterization so that $[\boldsymbol{\omega}_1]_\times = [\boldsymbol{\omega}_2]_\times \Leftrightarrow \boldsymbol{\omega}_1 = \boldsymbol{\omega}_2$, this means that our Lie algebra bracket operator given in (3.3.13) inherits the properties of the cross-product $[\boldsymbol{\omega}_1, \boldsymbol{\omega}_2] = \boldsymbol{\omega}_1 \times \boldsymbol{\omega}_2$. It follows that the axioms from equations (3.3.5)-(3.3.2) are satisfied since they are all basic properties of the cross product.

Since $\mathfrak{so}(3)$ is a Lie algebra, it is by definition a tangent space at the identity of its corresponding Lie group. The corresponding Lie group of $\mathfrak{so}(3)$ is $SO(3)$, which is evident if we observe the derivatives of rotation around each axis at the identity matrix $I \in SO(3)$, which are given by G_1, G_2, G_3 in equation (3.3.14).

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix},\ G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix},\ G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.3.14}$$

By definition G_1, G_2, G_3 span the tangent space of $SO(3)$ at the identity matrix, and from the definition of the skew-symmetric matrix given in section 3.2.5 we see that they also span the set of skew-symmetric matrices, ie. $\mathfrak{so}(3)$. This means that $\mathfrak{so}(3)$ is the tangent space of $SO(3)$ at the identity matrix, and it is thus the Lie Algebra associated with $SO(3)$. The matrices G_1, G_2, G_3 are generators of the vector space $\mathfrak{so}(3)$, which means that we can represent all elements $[\boldsymbol{\omega}]_\times \in \mathfrak{so}(3)$ as linear combinations of $\boldsymbol{\omega} \in \mathbb{R}^3$ and G_1, G_2, G_3 as in equation (3.3.10):

$$[\boldsymbol{\omega}]_\times = \omega_1 G_1 + \omega_2 G_2 + \omega_3 G_3 \tag{3.3.15}$$

Sometimes we write $\boldsymbol{\omega} \in \mathfrak{so}(3)$ for ease of notation. Here the parameters $\omega_1, \omega_2, \omega_3$ are viewed as a representation of the skew-symmetric matrix $[\boldsymbol{\omega}]_\times$.

3.3.6 Map from $\mathfrak{so}(3)$ to $SO(3)$

The map between elements in $\mathfrak{so}(3)$ and elements in $SO(3)$ is given by matrix exponentiation, as defined in equation (3.3.16):

$$\exp(A) = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + ... = \sum_{i=0}^{\infty} \frac{1}{i!}A^i,\ A \in \mathbb{R}^{n \times n} \tag{3.3.16}$$

In order to see why the exponential map relates elements in $\mathfrak{so}(3)$ to elements in $SO(3)$, it is useful to derive and analyze a closed form expression. Using the cube reduction of skew-symmentric matrices

from equation (3.2.10) and the Taylor expansion of sine and cosine, we arrive at the Rodrigues formula for skew-symmetric matrix exponentiation:

$$\exp([\boldsymbol{\omega}]_\times) = I + [\boldsymbol{\omega}]_\times + \frac{1}{2!}[\boldsymbol{\omega}]_\times^2 + \frac{1}{3!}[\boldsymbol{\omega}]_\times^3 + ... \tag{3.3.17}$$

$$= I + \sum_{i=0}^{\infty}\Big(\frac{[\boldsymbol{\omega}]_\times^{2i+1}}{(2i+1)!} + \frac{[\boldsymbol{\omega}]_\times^{2i+2}}{(2i+2)!}\Big) \tag{3.3.18}$$

$$= I + \Big(\sum_{i=0}^{\infty}\frac{(-1)^i\omega^{2i}}{(2i+1)!}\Big)[\boldsymbol{\omega}]_\times + \Big(\sum_{i=0}^{\infty}\frac{(-1)^i\omega^{2i}}{(2i+2)!}\Big)[\boldsymbol{\omega}]_\times^2 \tag{3.3.19}$$

$$= I + \Big(1 - \frac{\omega^2}{3!} + \frac{\omega^4}{5!} - ...\Big)[\boldsymbol{\omega}]_\times + \Big(\frac{1}{2!} - \frac{\omega^2}{4!} + \frac{\omega^4}{6!} - ...\Big)[\boldsymbol{\omega}]_\times^2 \tag{3.3.20}$$

$$= I + \Big(\frac{sin(\omega)}{\omega}\Big)[\boldsymbol{\omega}]_\times + \Big(\frac{1 - cos(\omega)}{\omega^2}\Big)[\boldsymbol{\omega}]_\times^2,\ \omega = \sqrt{\boldsymbol{\omega}^T\boldsymbol{\omega}} \tag{3.3.21}$$

It is possible to show that the rotation represented by $\exp([\boldsymbol{\omega}]_\times)$ is a rotation by $\omega = \sqrt{\boldsymbol{\omega}^T\boldsymbol{\omega}}$ radians around the axis given by $\boldsymbol{\omega}$. In fact Rodrigues formula is used when rotating by a rotation vector, as defined in section 3.2.3, so $\mathfrak{so}(3)$ elements are equivalent to rotation vectors. Rodrigues formula shows that any $[\boldsymbol{\omega}]_\times = [\omega\boldsymbol{a}]_\times \in \mathfrak{so}(3)$ generates a valid rotation $\exp([\boldsymbol{\omega}]_\times) \in SO(3)$.

Rodrigues formula also shows that we will get the same exponential $\exp([\boldsymbol{\omega}]_\times)$ for any $(\sqrt{\boldsymbol{\omega}^T\boldsymbol{\omega}} + 2\pi n)\boldsymbol{\omega},\ n \in \mathbb{Z}$ because of the trigonometric functions. Thus the exponential map from $\mathfrak{so}(3)$ to $SO(3)$ is non-injective, meaning that we can generate some element in $SO(3)$ from multiple elements in $\mathfrak{se}(3)$. If we limit the rotation angle, $|\omega| < \pi$ of the elements in $\mathfrak{so}(3)$, then we can guarantee that they will all generate different rotations through the exponential map, making the map injective.

Now, in order to show that the exponential map is a complete bijection between $\mathfrak{so}(3)$ and $SO(3)$, it is necessary to show that every rotation in $SO(3)$ can be generated by an element in $\mathfrak{so}(3)$. In order to do this, we should find the inverse of the exponential map.

Inverse of Rodrigues Formula

We represent the element $\boldsymbol{\omega}$ in its angle-axis form by putting $\boldsymbol{\omega} = \omega\boldsymbol{a},\ \boldsymbol{a}^T\boldsymbol{a} = 1$. We can then use the sqew square-reduction formula (3.2.13) to obtain a different form of the Rodrigues formula which is more suited for inverting:

$$\exp([\boldsymbol{\omega}]_\times) = \exp([\omega\boldsymbol{a}]_\times) = \cos\omega I + (1 - \cos\omega)\boldsymbol{a}\boldsymbol{a}^T + \sin\omega[\boldsymbol{a}]_\times \tag{3.3.22}$$

The trace, $tr(\cdot)$ of a matrix is the sum of its diagonal elements. By evaluating the trace of the rotation matrix, we obtain a closed form expression for the angle, ω given by the rotation matrix, $C = \exp([\boldsymbol{\omega}]_\times$:

$$tr(C) = tr(\cos\omega I + (1 - \cos\omega)\boldsymbol{a}\boldsymbol{a}^T + \sin\omega[\boldsymbol{a}]_\times) \tag{3.3.23}$$

$$= \cos\omega \cdot tr(I) + (1 - \cos\omega) \cdot tr(\boldsymbol{a}\boldsymbol{a}^T) + \sin\omega \cdot tr([\boldsymbol{a}]_\times) \tag{3.3.24}$$

$$= \cos\omega \cdot 3 + (1 - \cos\omega) \cdot 1 + \sin\omega \cdot 0 \tag{3.3.25}$$

$$= 2\cos\omega + 1 \tag{3.3.26}$$

$$\Rightarrow \omega = \cos^{-1}\Big(\frac{tr(C) - 1}{2}\Big) + 2\pi n,\ n \in \mathbb{Z} \tag{3.3.27}$$

Here we choose n so that $|\omega| < \pi$. We must also make sure that we pick the right sign for ω, since $\cos -\omega = \cos \omega$. In practice, we pick a sign and then see if we get the correct rotation C when going the other way after having obtained the solution $\boldsymbol{\omega}$. If we get the wrong rotation, $\exp([\boldsymbol{\omega}]_\times) \neq C$, we switch the sign of ω.

In order to obtain the vector $\boldsymbol{a}$, we observe that a rotation of a vector around its own axis changes nothing: $C\boldsymbol{a} = \boldsymbol{a}$. This means that $\boldsymbol{a}$ is an eigenvector of C with eigenvalue 1, and we can solve the corresponding eigenproblem to find $\boldsymbol{a}$. We can use the formula $\log(C) = \dfrac{\omega}{2\sin\omega} \cdot (C - C^T)$, for which we will omit derivation, and pick the skew symmetric parameters $\boldsymbol{\omega}$ from the resulting matrix according to the definition of the skew symmetric matrix (3.2.6).

Using the method above, we are able to find a corresponding element $\boldsymbol{\omega} = \omega\boldsymbol{a} \in \mathfrak{so}(3)$ for any rotation $C \in SO(3)$, and together with the injective formulation of the Rodrigues formula, we have therefore shown that the exponential map is a bijective map between $\mathfrak{so}(3)$ and $SO(3)$.

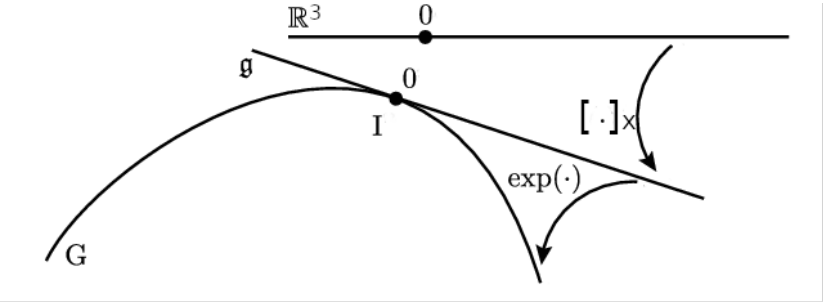


Figure 3.3: A conceptualized representation of the spaces $\mathbb{R}^3$, $\mathfrak{g} = \mathfrak{so}(3)$, and $G = SO(3)$. The line representation of $\mathbb{R}^3$ and the Lie algebra $\mathfrak{g}$ highlights the fact that they are vector spaces and are thus linear under addition and scalar multiplication, as opposed to the Lie group, G, which is a differentiable manifold, and thus merely resembles a vector space locally.

3.3.7 Derivative of $SO(3)$ rotation

It is also interesting to calculate the derivative of a rotation $\boldsymbol{y} = C \cdot \boldsymbol{x},\ \boldsymbol{x} \in \mathbb{R}^3$ with respect to the Lie algebra element $[\boldsymbol{\omega}]_\times \in \mathfrak{so}(3),\ \exp([\boldsymbol{\omega}]_\times) = C$. We do this by modeling the infinitesimal pertubation from the rotation C as a rotation $\exp([\boldsymbol{\Delta\omega}]_\times)$, pertubated through composure. Here $\boldsymbol{\Delta\omega} \in \mathbb{R}^3$ is an infinitesimal pertubation from $\boldsymbol{\omega} \in \mathbb{R}^3$. We also use the fact that the derivatives, $\partial \exp([\boldsymbol{\omega}]_\times) \,/\, \partial\omega_i,\ i = 1, 2, 3$ at $\boldsymbol{\omega} = 0$ are the generators G_1, G_2, G_3 of $\mathfrak{so}(3)$ as given in equation (3.3.14).

$$\boldsymbol{y} = C \cdot \boldsymbol{x} = \exp([\boldsymbol{\omega}]_\times) \cdot \boldsymbol{x} \tag{3.3.28}$$

$$\Rightarrow \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\omega}} = \frac{\partial}{\partial \boldsymbol{\Delta\omega}}\bigg|_{\boldsymbol{\Delta\omega}=0} \Big(\exp([\boldsymbol{\Delta\omega}]_\times) \cdot C \Big) \cdot \boldsymbol{x} \tag{3.3.29}$$

$$= \frac{\partial}{\partial \boldsymbol{\Delta\omega}}\bigg|_{\boldsymbol{\Delta\omega}=0} \Big(\exp([\boldsymbol{\Delta\omega}]_\times) \Big) \cdot (C \cdot \boldsymbol{x}) \tag{3.3.30}$$

$$= \frac{\partial}{\partial \boldsymbol{\Delta\omega}}\bigg|_{\boldsymbol{\Delta\omega}=0} \Big(\exp([\boldsymbol{\Delta\omega}]_\times) \Big) \cdot \boldsymbol{y} \tag{3.3.31}$$

$$= (G_1\boldsymbol{y}|G_2\boldsymbol{y}|G_3\boldsymbol{y}) \tag{3.3.32}$$

$$= -[\boldsymbol{y}]_\times \tag{3.3.33}$$

$$= \begin{bmatrix} 0 & y_3 & -y_2 \\ -y_3 & 0 & y_1 \\ y_2 & -y_1 & 0 \end{bmatrix} \tag{3.3.34}$$

So the derivative of $\boldsymbol{y} = C \cdot \boldsymbol{x},\ C = \exp([\boldsymbol{\omega}]_\times)$ with respect to $\boldsymbol{\omega}$ is $-[\boldsymbol{y}]_\times$.

3.3.8 $SE(3)$

We define the Euclidean group $E(3)$ as the set of transformations of a Euclidean space $\mathbb{E}^3$ (for instance $\mathbb{R}^3$) that preserve the Euclidean distance between elements. Such transformations are called isometries, and they consist of all translations, rotations and reflections on $\mathbb{E}^3$ as well as any composition of these. We then define the special Euclidean group $SE(3)$ as the set of direct isometries on $\mathbb{E}^3$, meaning the set of isometries that preserve orientation. Direct isometries are different from general, or indirect isometries in that they consist only of translations and rotations, and not reflections. Using the Euclidean space given by the set homogenous points $\tilde{\boldsymbol{x}},\ \boldsymbol{x} \in \mathbb{R}^3$ as described in section 3.2.4, we can represent elements $T \in SE(3)$ as matrices $T \in \mathbb{R}^{4x4}$ given by equation (3.3.35).

$$T = \begin{bmatrix} C & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix},\ C \in SO(3),\ \boldsymbol{t} \in \mathbb{R}^3 \tag{3.3.35}$$

The upper left 3×3 sub-matrix of T is an element $C \in SO(3)$, representing the rotation part of the transformation. The vector $t \in \mathbb{R}^3$ represents the translation part of the transformation.

In order to apply the transformation T to a vector $\boldsymbol{x} = [x, y, z]^T$ given by its homogenous representation $\tilde{\boldsymbol{x}} = [x, y, z, 1]^T$, we use matrix-vector multiplication:

$$\tilde{\boldsymbol{y}} = T \cdot \tilde{\boldsymbol{x}} = \begin{bmatrix} C & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.3.36}$$

The result of this multiplication is a homogenous representation $\tilde{\boldsymbol{y}}$ of the transformed point $\boldsymbol{y} \in \mathbb{R}^3$. The fourth parameter of the homogenous representation of $\tilde{\boldsymbol{x}}$ is left untouched by the transformation because the bottom row of T is identical to the bottom row of I^4. This means that by setting this fourth parameter to 1 and applying equation (3.2.5), we can immediately extract the coordinates of the transformed vector $\boldsymbol{y}$ as the first three coordinates of $\tilde{\boldsymbol{y}}$:

$$\tilde{\boldsymbol{y}} = T \cdot \tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \tilde{y}_3 \\ 1 \end{bmatrix} \Rightarrow \boldsymbol{y} = \begin{bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \tilde{y}_3 \end{bmatrix} \tag{3.3.37}$$

Like $SO(3)$, the group operation for $SE(3)$ is composure implemented through matrix multiplication:

$$T_1 \cdot T_2 = \begin{bmatrix} C_1 & \boldsymbol{t_1} \\ \boldsymbol{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} C_2 & \boldsymbol{t_2} \\ \boldsymbol{0} & 1 \end{bmatrix} = \begin{bmatrix} C_1 C_2 & C_1 \boldsymbol{t_2} + \boldsymbol{t_1} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{3.3.38}$$

We show that $SE(3)$ satisfies all the Lie axioms from equation (3.3.1)-(3.3.4). Closure follows directly from the concept of transformations: Since all compositions of translation and rotation is an element of $SE(3)$, we can't possibly compose two transformations $T_1, T_2 \in SE(3)$ such that $T_1 \cdot T_2 \notin SE(3)$. We can also observe that the expression at the right in equation (3.3.38) satisfies the definition (3.3.35) and is thus an element of $SE(3)$. Associativity follows from multiplying the matrices obtained from direct substition into section 3.3.1, although we will omit the derivation here. The inverse element for $SE(3)$ is given by equation (3.3.39):

$$T = \begin{bmatrix} C & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \;\Rightarrow\; T^{-1} = \begin{bmatrix} C^T & -C^T \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \tag{3.3.39}$$

We can check this by calculating $T \cdot T^{-1}$:

$$T \cdot T^{-1} = \begin{bmatrix} C & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} C^T & -C^T \boldsymbol{t} \\ 0 & 1 \end{bmatrix} = I^4 \tag{3.3.40}$$

Where we have used the orthogonality property (3.3.11) of $SO(3)$, $CC^T = C^T C = I^3$. The result is the identity matrix I^4, which serves as the identity element for $SE(3)$, fulfilling the last requirement for $SE(3)$ to be a Lie group.

3.3.9 $\mathfrak{se}(3)$

$\mathfrak{se}(3)$ is the lie algebra of $SE(3)$, and it is given by its generators $G_1, ..., G_6$:

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\; G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\; G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.3.41}$$

$$G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\; G_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\; G_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3.42}$$

The matrices G_4, G_5, G_6 are the differential rotations, and they are therefore similar to the generators in $\mathfrak{so}(3)$. G_1, G_2, G_3 are the differential translations. Together they generate the matrices $\boldsymbol{\xi}^\wedge$ as defined in equation (3.3.43):

$$\boldsymbol{\xi}^\wedge \equiv \xi_1 G_1 + ... + \xi_6 G_6 \in \mathfrak{se}(3) \tag{3.3.43}$$

The coordinates $\xi_1, ..., \xi_6$ of $\boldsymbol{\xi}$ is sometimes reffered to as the "twist" coordinates of its corresponding $SE(3)$ transformation T. If we divide $\boldsymbol{\xi}$ into the translational and rotational coordinates, $\boldsymbol{u}, \boldsymbol{\omega}$, we get an expression which is easier to work with:

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^6, \ \boldsymbol{u}, \boldsymbol{\omega} \in \mathbb{R}^3 \tag{3.3.44}$$

$$\Rightarrow \boldsymbol{\xi}^\wedge = u_1 G_1 + u_2 G_2 + u_3 G_3 + \omega_1 G_4 + \omega_2 G_5 + \omega_3 G_6 \tag{3.3.45}$$

$$= \begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{u} \\ \mathbf{0} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & u_1 \\ \omega_3 & 0 & -\omega_1 & u_2 \\ -\omega_2 & \omega_1 & 0 & u_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3.46}$$

This matrix expression, together with the same bracket operator given in equation (3.3.13), can be used to show that $\mathfrak{se}(3)$ fills the requirements (3.3.5)-(3.3.2) for being a lie algebra. We will omit the proofs here though. For ease of notation, we write $\boldsymbol{\xi} \in \mathfrak{se}(3)$.

As in $\mathfrak{so}(3)$, the commutator $[A, B] = AB - BA$ is used as the lie bracket for $\mathfrak{se}(3)$.

3.3.10 Map from $\mathfrak{se}(3)$ to $SE(3)$

As in $SO(3)$ and $\mathfrak{so}(3)$, the map from $\mathfrak{se}(3)$ to $SE(3)$ is given by matrix exponentiation. We derive a closed form expression with the same techniques used in the derivation of the Rodrigues formula, equation (3.3.21):

$$\exp(\boldsymbol{\xi}^\wedge) = \exp\Big(\begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{u} \\ \mathbf{0} & 0 \end{bmatrix} \Big) \tag{3.3.47}$$

$$= I + \Big(\begin{bmatrix} [\boldsymbol{\omega}]_\times & \boldsymbol{u} \\ \mathbf{0} & 0 \end{bmatrix} \Big) + \frac{1}{2!} \Big(\begin{bmatrix} [\boldsymbol{\omega}]_\times^2 & [\boldsymbol{\omega}]_\times \boldsymbol{u} \\ \mathbf{0} & 0 \end{bmatrix} \Big) + \frac{1}{3!} \Big(\begin{bmatrix} [\boldsymbol{\omega}]_\times^3 & [\boldsymbol{\omega}]_\times^2 \boldsymbol{u} \\ \mathbf{0} & 0 \end{bmatrix} \Big) + ... \tag{3.3.48}$$

$$= \begin{bmatrix} \exp([\boldsymbol{\omega}]_\times) & V\boldsymbol{u} \\ \mathbf{0} & 1 \end{bmatrix}, \ V = I + \frac{1}{2!}[\boldsymbol{\omega}]_\times + \frac{1}{3!}[\boldsymbol{\omega}]_\times^2 + ... \tag{3.3.49}$$

Here the top left sub-matrix, $\exp([\boldsymbol{\omega}]_\times)$ is given by Rodrigues formula (3.3.21).

The matrix V is called the left jacobian of $SO(3)$, and it appears in multiple situations when dealing with lie groups. We derive a closed form expression for V:

$$V = \sum_{i=0}^{\infty} \frac{1}{i+1} \Big([\boldsymbol{\omega}]_\times \Big)^i \tag{3.3.50}$$

$$= I + \sum_{i=0}^{\infty} \Big(\frac{[\boldsymbol{\omega}]_\times^{2i+1}}{(2i+2)!} + \frac{[\boldsymbol{\omega}]_\times^{2i+2}}{(2i+3)!} \Big) \tag{3.3.51}$$

$$= I + \Big(\sum_{i=0}^{\infty} \frac{(-1)^i \omega^{2i}}{(2i+2)!} \Big) [\boldsymbol{\omega}]_\times + \Big(\sum_{i=0}^{\infty} \frac{(-1)^i \omega^{2i}}{(2i+3)!} \Big) [\boldsymbol{\omega}]_\times^2 \tag{3.3.52}$$

$$= I + \Big(\frac{1}{2!} - \frac{\omega^2}{4!} + \frac{\omega^4}{6!} + ... \Big) [\boldsymbol{\omega}]_\times + \Big(\frac{1}{3!} - \frac{\omega^2}{5!} + \frac{\omega^4}{7!} + ... \Big) [\boldsymbol{\omega}]_\times^2 \tag{3.3.53}$$

$$= I + \Big(\frac{1 - \cos\omega}{\omega^2} \Big) [\boldsymbol{\omega}]_\times + \Big(\frac{\omega - \sin\omega}{\omega^3} \Big) [\boldsymbol{\omega}]_\times^2, \ \omega = \sqrt{\boldsymbol{\omega}^T \boldsymbol{\omega}} \tag{3.3.54}$$

The inverse, V^{-1} is given by:

$$V^{-1} = I - \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{1}{\omega^2}\Big(1 - \frac{\omega}{2}\frac{\sin\omega}{1-\cos\omega}\Big)[\boldsymbol{\omega}]_\times^2 \tag{3.3.55}$$

In order to obtain the logarithm, $\log(T) \in \mathfrak{se}(3)$, $T \in SE(3)$, we can use the $SO(3)$ logarithm described in section 3.3.6 on the upper left sub-matrix C of $T = \begin{bmatrix} C & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix}$, and then calculate the translation part $\boldsymbol{u}$ of the twist coordinates according to $\boldsymbol{u} = V^{-1}\boldsymbol{t}$.

Positive definiteness of VV^T

Another useful property for V is that VV^T is positive definite, meaning that $\boldsymbol{x}^T VV^T\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \mathbf{0}$. We can show this by defining γ and $\boldsymbol{a}$ according to equation (3.3.56) and then expanding the expression:

$$\gamma = 2\frac{1-\cos\omega}{\omega^2} > 0, \quad \boldsymbol{a} = \frac{\boldsymbol{\omega}}{\omega} \tag{3.3.56}$$

$$\boldsymbol{x}^T VV^T\boldsymbol{x} = \boldsymbol{x}^T(\gamma I + (1-\gamma)\boldsymbol{a}\boldsymbol{a}^T)\boldsymbol{x}^T = \boldsymbol{x}^T(\boldsymbol{a}\boldsymbol{a}^T - \gamma[\boldsymbol{a}_\times][\boldsymbol{a}_\times])\boldsymbol{x}^T \tag{3.3.57}$$

$$= \boldsymbol{x}^T\boldsymbol{a}\boldsymbol{a}^T\boldsymbol{x}^T + \gamma([\boldsymbol{a}_\times]\boldsymbol{x})^T([\boldsymbol{a}_\times]\boldsymbol{x}) \tag{3.3.58}$$

$$= (\boldsymbol{a}^T\boldsymbol{x})^T(\boldsymbol{a}^T\boldsymbol{x}) + 2\gamma([\boldsymbol{a}_\times]\boldsymbol{x})^T([\boldsymbol{a}_\times]\boldsymbol{x}) > 0 \tag{3.3.59}$$

The last inequality holds because γ as defined in equation (3.3.56) is strictly positive, while the other terms in equation (3.3.59) are positive semi-definite.

Since V is invertible, we also have:

$$(V^{-1})^T(V^{-1}) > 0 \tag{3.3.60}$$

3.3.11 Derivative of $SE(3)$ transformation

We obtain the derivative of a $SE(3)$ transformation.

A transformation T acts on a three-dimensional vector $\boldsymbol{x}$ according to:

$$\boldsymbol{y} = (C|\boldsymbol{t}) \cdot \Big(\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} \Big) = C\boldsymbol{x} + \boldsymbol{t},\ T = \begin{bmatrix} C & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} = \exp(\boldsymbol{\xi}^\wedge) \tag{3.3.61}$$

By the same argument as in equation (3.3.28), we obtain the result:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\xi}} = (G_1\boldsymbol{y} \mid G_2\boldsymbol{y} \mid G_3\boldsymbol{y} \mid G_4\boldsymbol{y} \mid G_5\boldsymbol{y} \mid G_6\boldsymbol{y}) = (I \mid -[\boldsymbol{y}]_\times) \tag{3.3.62}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & y_3 & -y_2 \\ 0 & 1 & 0 & -y_3 & 0 & y_1 \\ 0 & 0 & 1 & y_2 & -y_1 & 0 \end{bmatrix} \tag{3.3.63}$$

3.4 Image Formation and Camera Mathematics

Here, we formulate a mathematical model for image formation. We also explain the major error sources affecting the image both from the environment and from the image formation process. Lastly, we describe geometry relating images taken from different perspectives.

3.4.1 Image Formation

We define a monochrome image mathematically as a map from image locations to image intensities:

$$I : \Omega \to \mathbb{R},\ \Omega \subset \mathbb{R}^2 \tag{3.4.1}$$

$$I(\mathbf{x}),\ \mathbf{x} \in \Omega \tag{3.4.2}$$

In practice, images are often captured using CMOS or CCD image sensors. These sensors perform quantization by dividing their image sensor area into small light-sensitive patches called pixels. The pixels are exposed for a given time period, called the exposure time. The intensity registered at a certain pixel is the cumulative amount of light hitting the pixel during the exposure time. This intensity is also quantized. It is common to use one byte to encode each intensity value, making the intensities take the form of integers from 0 to 255.

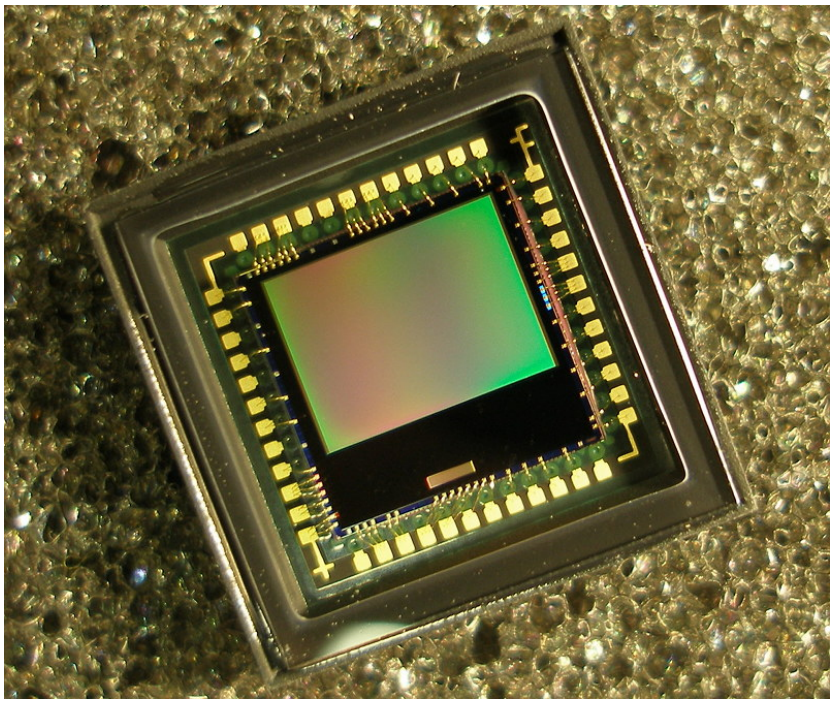


Figure 3.4: CMOS image sensor

At what time the pixel intensities get exposed is dependent on the shutter type. With global shutter, all pixels are activated at the same time, making global shutter cameras a good choice for visual odometry applications. Rolling shutter cameras on the other hand read the pixels row by row over a short time period. If there is relative motion between the camera and the scene during this time period, it can cause distortions in the image. Some odometry algorithms take this phenomenon into

account by adding a time parameter $t \in \mathbb{R}$ to the image map, representing the time of exposure for a specific pixel[30]:

$$I : \Omega \times \mathbb{R} \to \mathbb{R},\ \Omega \subset \mathbb{R}^2 \tag{3.4.3}$$

$$I(\mathbf{x}, t),\ \mathbf{x} \in \Omega,\ t \in \mathbb{R} \tag{3.4.4}$$

3.4.2 Underwater Image Distortion

When trying to track scenes in underwater images, we should be aware of certain distortion effects that reduce the visibility of the scene.

Veiling Light / Backscatter

When an underwater scene is illuminated, some of the light will reach the water between the underwater camera and the scene it tries to capture, and a portion of the light will be scattered towards the camera. This light is called veiling light since it appears to come from the scene, but is really just the color of the water and small particles like micro-organisms and algae contained in the water.

The veiling light appears like a haze, reducing the contrast in the scene.

Veiling light is also called backscatter. It can be produced by artificial light sources, like LED lights as well as natural light sources, like the sun.

Light Attenuation

Heavy light scattering and absorption leads to poor light attenuation in water. This means that every meter, a portion of the light travelling from an underwater structure towards the camera will be scattered away or absorbed. The structures visibility and saliency is therefore reduced exponentially with larger viewing distances.

Light attenuatoin and veiling light can be compensated for to some degree using dehazing algorithms[23]. The performance of this kind of algorithm can potentially be increased by image depth estimates, since both the veiling light and the attenuation increases with the distance that the light passes before reaching the camera.

Marine Snow

Marine snow often appear n underwater images as small spots with high light intensity. It mostly consists of organic material falling towards the upper layers of the water column. If the illumination source of the scene is close to the camera and with the same direction (like for instance a built in camera flash), then marine snow close to the lens will be lit up, and it will appear bright white.

Too much marine snow can potentially represent a large challenge for odometry applications, since it will be hard to see enough of the structure we wish to track. An odometry algorithm may even start tracking the snow instead of the structure, which would lead the position estimate of the algorithm to drift with the water.

Some algorithms use explicit modeling of marine snow to remove it from underwater images.[17][29]

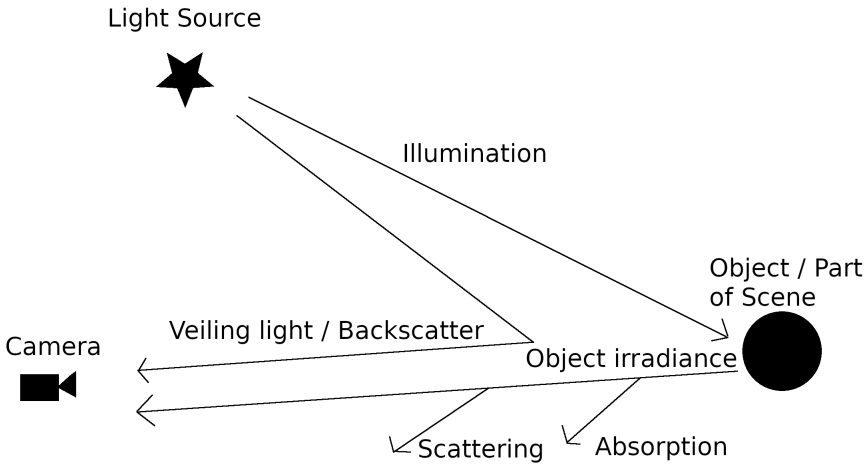


Figure 3.5: Illustration of how light is scattered and absorbed underwater on its way to the camera.

Occlusion

Some underwater scenes are dynamic due to fauna. For instance if an underwater scene is actively illuminated, then fish may be attracted to the light source, crowding the scene. Occlusions make it difficult to track stationary points consistently over long periods of time.

Some underwater VO methods consider retracking mechanisms that allow them to recover KLT feature tracking of features that gets occluded for a short time period[20].

3.4.3 Pinhole Camera Model

We present the pinhole projection model of a camera and define functions for deprojecting something observed in one camera view and projecting it into another camera view.

Projection

A camera projection model defines how 3d points in a scene is projected onto the two-dimensional image surface of an image sensor. The pinhole model is best described through the intrinsic camera matrix K given by equation (3.4.5):

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4.5}$$

K is parametrized by four variables, all given in pixels. f_x and f_y are the focal lengths in x and y direction, and c_x and c_y are the coordinates of the image center. These parameters are called "intrinsic" parameters, meaning that they contain information that is internal to the camera system. They can thus be estimated through calculations using the geometry of the camera system, although the practical way of estimating the parameters is through a calibration process using the pinhole model in combination with images of a known scene, like a chess board pattern or similar.

The projection $\boldsymbol{x} = [x_1, x_2]^T$ (in pixels) of a 3d point $\boldsymbol{p} = [p_x, p_y, p_z]^T$ onto an image sensor is given by equation (3.4.6):

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \frac{1}{p_z} K \boldsymbol{p} = \frac{1}{p_z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{3.4.6}$$

Later, we will refer to this projection through the function $\boldsymbol{\pi}$:

$$\boldsymbol{\pi}(\boldsymbol{p}) = \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.4.7}$$

Where x_1, x_2 are obtained from equation (3.4.6).

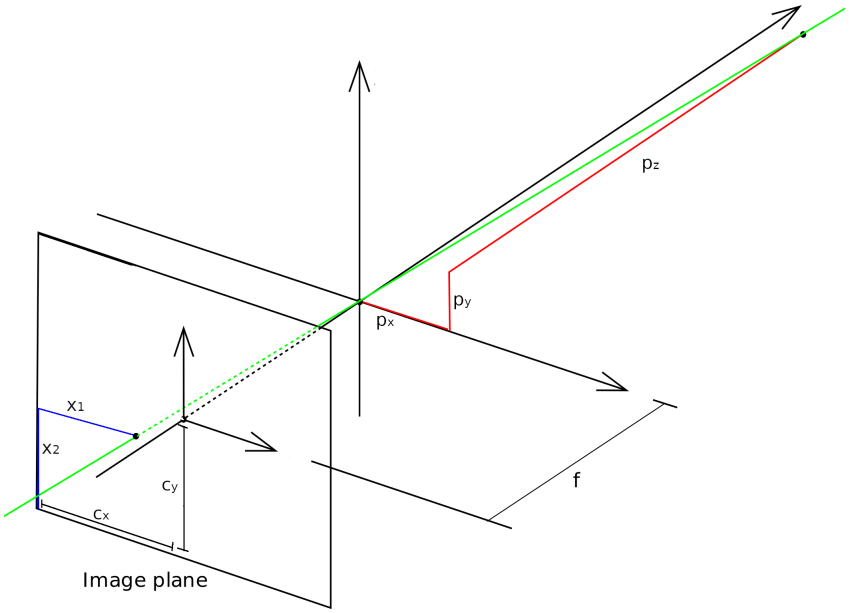


Figure 3.6: Geometric illustration of the pinhole model.

Deprojection

Using the same model, deprojection is given by:

$$\frac{1}{p_z} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \begin{bmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \tag{3.4.8}$$

In order to resolve the scale of $\boldsymbol{x}$ in equation (3.4.8) we need to know the depth d since this information is lost in the projection $\boldsymbol{y}$. We define the deprojection function $\boldsymbol{\pi}^{-1}$:

$$\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \boldsymbol{\pi}^{-1}(\boldsymbol{x}, d) \tag{3.4.9}$$

Where the depth d is used to resolve the scale ambiguity in equation (3.4.8) using the identity $p_z = d$.

We also write $\boldsymbol{\pi}^{-1}(\boldsymbol{x}) = \boldsymbol{\pi}^{-1}(\boldsymbol{x}, d)$, as a shorthand where we assume that the depth d is implicitly accounted for.

Reprojection

The functions $\boldsymbol{\pi}, \boldsymbol{\pi}^{-1}$ encode the intrinsics of our camera model, unless we want to include extra intrinsic models like lens distortion and the vignetting effect[51]. In order to reproject an image location from one image to another, we need to also consider extrinsic camera parameters, i.e. the relative pose $T \in SE(3)$ betweeen the two cameras. For a pose representation $\boldsymbol{\theta}$, we parametrize T by $T(\boldsymbol{\theta})$. We can then define the warp function w for an image location $\boldsymbol{x} = [x_1, x_2]^T$:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = w(\boldsymbol{x}, \boldsymbol{\theta}, d) = \boldsymbol{\pi}(T(\boldsymbol{\theta})\boldsymbol{\pi}^{-1}(\boldsymbol{x})) \tag{3.4.10}$$

For a shorter expression, we use the notation $w(\boldsymbol{x}, \boldsymbol{\theta}) = w(\boldsymbol{x}, \boldsymbol{\theta}, d)$, where we assume that the image depth d is implicitly given along with $\boldsymbol{x}$.

For ease of notation, we assume that all three dimensional vectors that are transformed by a transformation matrix are implicitly converted to their homogenous representation according to equation (3.2.3). Conversely, we also assume that the homogenous vector output from the transformation T in equation (3.4.10) is implicitly dehomogenized according to equation (3.2.5).

3.5 Lucas Kanade

The Lucas Kanade problem[3] is to minimize the photometric error E, which is the sum of squared differences between a template T and an image I transformated by a warp, with respect to the parameterization of the warp. This is captured in equation (3.5.1), where T and I are the image functions respectively for the template and the current image, w is the warp function from equation (3.4.10), $\boldsymbol{\theta}$ is the chosen parameterization of the warp, and $\{\boldsymbol{x}\}$ denotes selected image locations in T.

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta})) - T(\boldsymbol{x}) \Big]^2 \tag{3.5.1}$$

This is a non-linear optimization problem, as a result of the image functions I and T being as good as guaranteed to be non-linear when representing photos. More specifically, it is a non-linear least squares problem, of which there are multiple optimization strategies. We consider the Gauss-Newton algorithm.

3.5.1 Gauss Newton

One of the more popular strategies for solving the non-linear least squares problem is called Gauss-Newton. In Gauss-Newton, the residuals of the photometric error are linearized at each iteration of the algorithm, using a first order taylor expansion as shown in equation (3.5.2):

$$\begin{aligned} E(\boldsymbol{\theta}, \boldsymbol{\Delta\theta}) &= \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta} + \boldsymbol{\Delta\theta})) - T(\boldsymbol{x}) \Big]^2 \\ &= \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta})) + \nabla I \frac{\partial w}{\partial \boldsymbol{\theta}} \boldsymbol{\Delta\theta} - T(\boldsymbol{x}) \Big]^2 \end{aligned} \tag{3.5.2}$$

Where we have written $\frac{\partial w(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial w}{\partial \boldsymbol{\theta}}$ for ease of notation.

We differentiate the linearized error function with respect to the parameter update $\boldsymbol{\Delta\theta}$:

$$\frac{\partial E}{\partial \boldsymbol{\Delta\theta}} = 2 \sum_x \Big[\nabla I \frac{\partial w}{\partial \boldsymbol{\theta}}\Big]^T \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta})) + \nabla I \frac{\partial w}{\partial \boldsymbol{\theta}} \boldsymbol{\Delta\theta} - T(\boldsymbol{x})\Big] \tag{3.5.3}$$

Then we set this expression to zero and solve for $\boldsymbol{\Delta\theta}$ in equation (3.5.4), effectively finding the stationary point for the linear-residual photomeric error function at the current iteration.

$$\frac{\partial E}{\partial \boldsymbol{\Delta\theta}} = 0$$

$$\Rightarrow \sum_x \Big[\nabla I \frac{\partial w}{\partial \boldsymbol{\theta}}\Big]^T \Big[\nabla I \frac{\partial w}{\partial \boldsymbol{\theta}}\Big] \boldsymbol{\Delta\theta} = \sum_x \Big[\nabla I \frac{\partial w}{\partial \boldsymbol{\theta}}\Big]^T \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta})) - T(\boldsymbol{x})\Big] \tag{3.5.4}$$

This stationary point $\boldsymbol{\Delta\theta}$ represents the parameter update that we expect to minimize the error function $E(\boldsymbol{\theta} + \boldsymbol{\Delta\theta})$ the most, given the linearization around $\boldsymbol{\theta}$. Thus $\boldsymbol{\theta}$ should be updated according to equation (3.5.5) before moving to the next iteration.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\Delta\theta} \tag{3.5.5}$$

The equations 3.5.4 and 3.5.5 are the normal equations for an iteration of the standard Gauss-Newton based Lucas Kanade algorithm. The algorithm should keep iterating until the update $\boldsymbol{\Delta\theta}$ is sufficiently low, giving a final solution $w(\boldsymbol{\theta})$ for the warp between the current image I and the template T.

We can write equation (3.5.4) as:

$$J^T J \boldsymbol{\Delta\theta} = -J^T r(\boldsymbol{\theta}) \tag{3.5.6}$$

with the total jacobian J and the total residual function r given by:

$$J = \sum_x \Big[\nabla I \frac{\partial w}{\partial \boldsymbol{\theta}}\Big], \tag{3.5.7}$$

$$r(\boldsymbol{\theta}) = \sum_x \Big[I(w(\boldsymbol{x}, \boldsymbol{\theta})) - T(\boldsymbol{x})\Big] \tag{3.5.8}$$

The solution is given by:

$$\boldsymbol{\Delta\theta} = -(J^T J)^{-1} J^T r(\boldsymbol{\theta}) \tag{3.5.9}$$

In order to solve equation (3.5.9), J and $J^T J$ need to be evaluated at each iteration because they depend on $\nabla I(w(\boldsymbol{x}, \boldsymbol{\theta}))$ and $\frac{\partial w}{\partial \boldsymbol{\theta}}$, which both depend on the parameterization $\boldsymbol{\theta}$, which changes on each iteration. We can reduce the computational burden of the algorithm significantly if we are able to relax this requirement and only evaluate the SSD of the residuals, $r(\boldsymbol{\theta})$ at each iteration before solving equation (3.5.9). Fortunately this is possible through the use of the inverse compositional Gauss-Newton algorithm as detailed in section 3.5.2 and [3]. From now on, we refer to the algorithm described above as the forward additive Gauss-Newton, to differentate it from the inverse compositional Gauss-Newton.

3.5.2 Inverse Compositional Gauss-Newton

The inverse compositional Gauss-Newton algorithm switches the roles of the template and image around so that the pertubation $\boldsymbol{\Delta\theta}$ is applied to T. It also formulates the update in terms of composition of warps instead of addition of warp parameters.

The inverse compositional photometric error is given by:

$$E(\boldsymbol{\theta}, \boldsymbol{\Delta\theta}) = \sum_x \Big[T(w(\boldsymbol{x}, \boldsymbol{\Delta\theta})) - I(w(\boldsymbol{x}, \boldsymbol{\theta}))\Big]^2 \tag{3.5.10}$$

As in section 3.5.1, we linearize the residuals along the pertubation $\boldsymbol{\Delta\theta}$ from the current warp parameter $\boldsymbol{\theta}$:

$$E(\boldsymbol{\theta}, \boldsymbol{\Delta\theta}) = \sum_x \Big[T(\boldsymbol{x}) + \nabla T \frac{\partial w}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\mathbf{0}} \boldsymbol{\Delta\theta} - I(w(\boldsymbol{x}, \boldsymbol{\theta}))\Big]^2 \tag{3.5.11}$$

We differentiate equation (3.5.11) and set it to zero, ending up with equation (3.5.12):

$$J^T J \boldsymbol{\Delta\theta} = -J^T r(\boldsymbol{\theta}) \tag{3.5.12}$$

with

$$J = \sum_x \Big[\nabla T \frac{\partial w}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\mathbf{0}}\Big], \tag{3.5.13}$$

$$r(\boldsymbol{\theta}) = \sum_x \Big[T(\boldsymbol{x}) - I(w(\boldsymbol{x}, \boldsymbol{\theta}))\Big] \tag{3.5.14}$$

The inverse compositional Gauss-Newton is completed by the inverse compositional update given by equation (3.5.15):

$$w(\boldsymbol{x}, \boldsymbol{\theta})) \leftarrow w(\boldsymbol{x}, \boldsymbol{\theta}) \circ w(\boldsymbol{x}, \boldsymbol{\Delta\theta})^{-1} \tag{3.5.15}$$

Where the $\circ$ operator denotes composition:

$$w(\boldsymbol{x}, \boldsymbol{\theta})) \circ w^(\boldsymbol{x}, \boldsymbol{\Delta\theta}) {-}1 \equiv w(w(\boldsymbol{x}, \boldsymbol{\Delta\theta})^{-1}, \boldsymbol{\theta})) \tag{3.5.16}$$

We see from equation (3.5.12) and (3.5.13) that there are two changes from equation (3.5.6) and (3.5.7). The first change is that $\dfrac{\partial w(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ has been switched to $\dfrac{\partial w(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\mathbf{0}}$ in the expression for the Jacobian J. The second change is that we have switched the role of T and I so that we evaluate $\nabla T(\boldsymbol{x})$ instead of $\nabla I(\boldsymbol{x}, \boldsymbol{\theta})$. This means that J and $J^T J$ do not depend on $\boldsymbol{\theta}$, and we can precompute them before the first iteration of Gauss-Newton, based on the template T.

3.5.3 Choice of Warp Parameter

The warp $w(\boldsymbol{x}, \boldsymbol{\theta})$ in Lucas-Kanade (3.5.1) can be parametrized in different ways. The choice of parametrization determines what kind of information you end up with after solving the problem. The canonical Lucas-Kanade problem simply puts:

$$w(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{x} + \boldsymbol{\theta},\ \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^2 \tag{3.5.17}$$

Here $\boldsymbol{\theta}$ represents the offset in pixels between the two image regions, which is referred to as optical flow.

Direct VO methods rely on estimating the 6DoF pose between the images, so they parametrize the warp using some kind of pose representation. For instance we can use the $\mathfrak{se}(3)$ pose representation as per section 3.3.9:

$$w(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\pi}(\exp(\boldsymbol{\theta})\boldsymbol{\pi}^{-1}(\boldsymbol{x})),\ \boldsymbol{x} \in \mathbb{R}^2,\ \boldsymbol{\theta} \in \mathfrak{se}(3) \tag{3.5.18}$$

Where $\boldsymbol{\pi}$ is the projection function as defined in equation (3.4.7), $\boldsymbol{\pi}^{-1}$ is the deprojection function, and where $\boldsymbol{x}$ represents the image coordinates of an image location. Here $\boldsymbol{\theta}$ takes the form of a six dimensional vector which parametrizes the $\mathfrak{se}(3)$ space like $\boldsymbol{\xi}$ in equation (3.3.44).

3.5.4 Sampling of Warped Image

Aside from the inverse compositional Gauss-Newton algorithm (3.5.2), additional computational efficiency can be gained by ignoring the warping of the image I when calculating the residual r in equation (3.5.12). Instead of warping the entire image around the pixels $\boldsymbol{x}$, we assume small motion $\boldsymbol{\theta}$ and use the photo consistency assumption. This allows us to evaluate the non-template part of the residuals by simply sampling the original image I at the image location specified by $w(\boldsymbol{x}, \boldsymbol{\theta})$.

Bi-linear sampling is commonly used for its simplicity. For GPGPU-based Lucas-Kanade implementations there are fast, built-in mechanisms for sampling textures on GPUs.

3.5.5 Outlier Suppression

The least squares based optimization algorithms presented so far have an important problem when used on real data. Abnormally large residual values heavily influence the solution because of the quadratic nature of the residuals in Lucas-Kanade (3.5.1). These kind of residuals are outliers caused by observations that can't be accounted for in our simplified model, like occlusion, moving objects or unaccounted image distortion.

In general, there are two strategies for dealing with outliers. The first is to model the effects causing the outliers and estimate their parameters so they can be removed from the problem. For example, in the case of underwater odometry, it is possible to detect and remove marine snow from the image[17][29].

The second possibility is to incorporate a mathematical kind of outlier suppression directly into the error function. In order to reduce the contribution of outliers, the quadratic term from equation (3.5.1) is replaced with a robust cost expression given by multiplication with the weighting function ω:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}} \omega(r_{\boldsymbol{x}}(\boldsymbol{\theta}))r_{\boldsymbol{x}}(\boldsymbol{\theta})^2 \tag{3.5.19}$$

Where we have substituted $r_{\boldsymbol{x}}(\boldsymbol{\theta}) = I(\omega(\boldsymbol{x}, \boldsymbol{\theta})) - T(\boldsymbol{x})$. The minimizer of the error is given by setting the derivative of equation (3.5.19) to zero:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{\boldsymbol{x}} \frac{\partial r_{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \omega(r_{\boldsymbol{x}}(\boldsymbol{\theta})) r_{\boldsymbol{x}}(\boldsymbol{\theta}) = 0 \tag{3.5.20}$$

This expression is similar to the minimizer of the normal non-weighted error function:

$$\frac{\partial \sum_{\boldsymbol{x}} r_{\boldsymbol{x}}(\boldsymbol{\theta})^2}{\partial \boldsymbol{\theta}} = \sum_{\boldsymbol{x}} \frac{\partial r_{\boldsymbol{x}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} r_{\boldsymbol{x}}(\boldsymbol{\theta}) = 0 \tag{3.5.21}$$

The only difference between equation (3.5.20) and equation (3.5.21) is the extra factor $\omega(r_{\boldsymbol{x}}(\boldsymbol{\theta}))$. This means that if we choose $\omega(r_{\boldsymbol{x}}(\boldsymbol{\theta}))$ so that it is convex around the zero point at $r_{\boldsymbol{x}}(\boldsymbol{\theta}) = 0$, then the minimizer for the weighted error function will be equivalent to the minimizer for the non-weighted error function.

For the iterative formulation, we treat the weights as constant scaling factors for the residuals in our Gauss-Newton formulation. We denote the constant weight factors as $\omega_{\boldsymbol{x}} = \omega(r_{\boldsymbol{x}}(\boldsymbol{\theta}))$. The weights $\omega_{\boldsymbol{x}}$ need to be recalculated after each iteration. We can incorporate the weights through a diagonal matrix W with the weights $\omega_{\boldsymbol{x}}$ along the diagonal. Since the weight matrix W is treated as constant for a given iteration, it is straight forward to derive the normal equation corresponding to equation (3.5.6):

$$J^T W J \boldsymbol{\Delta\theta} = -J^T W r(\boldsymbol{\theta}) \tag{3.5.22}$$

Multiple weight functions are discussed in the literature. We consider the Huber loss function, which suppresses the contributions of outliers by making the error function linear in residuals that are larger than a limit h. The Huber weights are given by $\omega^h(x)$:

$$\omega^h(r_{\boldsymbol{x}}) = \begin{cases} \frac{1}{2} & \text{if } |r_{\boldsymbol{x}}| \leq h \\ \frac{h}{|r_{\boldsymbol{x}}|} - \frac{h^2}{2r_{\boldsymbol{x}}^2} & \text{if } |r_{\boldsymbol{x}}| > h \end{cases} \tag{3.5.23}$$

The complete Huber loss function ρ^h is the product of the weights and the squared residuals $r_{\boldsymbol{x}}^2$:

$$\rho^h(r_{\boldsymbol{x}}) = \omega^h(r_{\boldsymbol{x}}) r_{\boldsymbol{x}}^2 = \begin{cases} \frac{1}{2} r_{\boldsymbol{x}}^2 & \text{if } |r_{\boldsymbol{x}}| \leq h \\ h|r_{\boldsymbol{x}}| - \frac{1}{2}h^2 & \text{if } |r_{\boldsymbol{x}}| > h \end{cases} \tag{3.5.24}$$

The huber loss function is both continuous and derivative, as can be seen in figure 3.7. It is convex around $r_{\boldsymbol{x}} = 0$, so it doesn't introduce change the local minimum of the error function.

In order to choose a good value for h, we need to have a good scale estimate of the inlier residuals. Of course, we don't know which residuals are inliers or outliers, so we need to filter them.

We can calculate a robust estimate of the standard deviation of inlier residuals by using the MAD scale estimator:

$$\sigma_{MAD}(\{r_{\boldsymbol{x}}\}) = k \cdot median(\{r_{\boldsymbol{x}}\}), \quad k = \frac{1}{Q(3/4)} \approx 1.4826 \tag{3.5.25}$$

Where Q denotes the inverse cumulative function for normal distributions.

We now choose $h = 1.345\sigma_{MAD}(\{r_{\boldsymbol{x}}\})$, where 1.345 is a tuning constant tuned to produce 95% efficiency when there are no outliers and still offer protection when there are outliers[22].

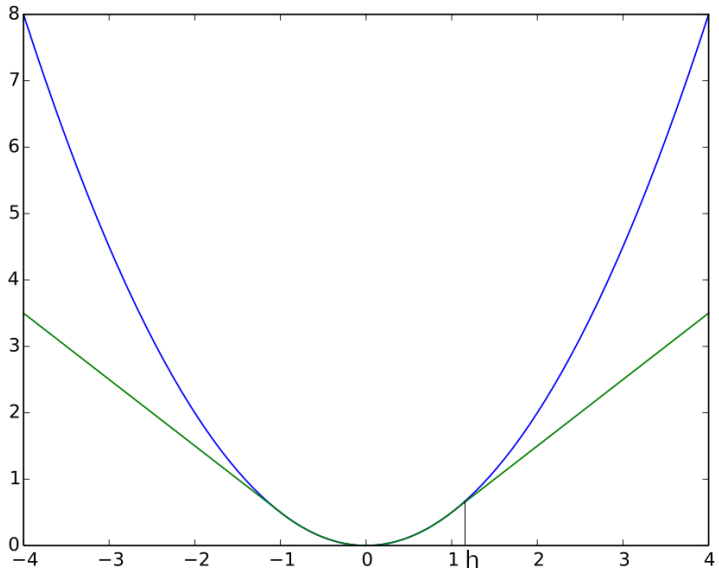


Figure 3.7: Illustration of the Huber loss of the residuals. The blue graph represents the quadratic residual $r_{\boldsymbol{x}}^2$ used in ordinary least squares, while the green graph represents the Huber loss $\rho^h(r_{\boldsymbol{x}})$.

3.5.6 Bayesian derivation of Lucas Kanade

The non-linear Lucas-Kanade problem was derived from photometric error minimization initially in section 3.5. Here, we derive it from a Bayesian perspective instead. This will help us to reason about how we can incorporate various prior information into our model.

We first formulate the problem as a Maximum a Posteriori (MAP) problem, where we maximize the posteriori probability of the parameters, $\boldsymbol{\theta}$ given the observations $\hat{\boldsymbol{f}} = [\hat{f}_1, ...\hat{f}_n]$:

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{\theta}|\hat{\boldsymbol{f}}) \tag{3.5.26}$$

We apply Bayes rule to equation (3.5.26) and obtain:

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; \frac{p(\hat{\boldsymbol{f}}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\hat{\boldsymbol{f}})} \tag{3.5.27}$$

Since $p(\hat{\boldsymbol{f}})$ does no depend on $\boldsymbol{\theta}$, we can drop it from the problem. Assuming no prior information about the parameters $\boldsymbol{\theta}$, we set $p(\boldsymbol{\theta})$ to the uniform distribution:

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; p(\hat{\boldsymbol{f}}|\boldsymbol{\theta}) \tag{3.5.28}$$

For now we assume all observations $\hat{f}_i$ are independent with identical distribution, which means we can rewrite their total probability distribution $\hat{\boldsymbol{f}}$ as the product of each observation:

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \prod_{i}^{n} p(\hat{f}_i|\boldsymbol{\theta}) \tag{3.5.29}$$

Then we switch to minimizing the negative log-likelihood, reducing multiplicative terms to sums and potential terms to multiplicative terms:

$$\boldsymbol{\theta}_{MAP} = \arg\min_{\boldsymbol{\theta}} \sum_{i}^{n} -\log(p(\hat{f}_i|\boldsymbol{\theta})) \tag{3.5.30}$$

We now assume that $p(\hat{f}_i)$ has a normal distribution with mean equal to $f(x_i, \boldsymbol{\theta})$ and standard deviation σ. The function $f(x_i, \boldsymbol{\theta})$ reflects the fact that an obervation $\hat{f}_i$ depends on what we observe, which in the case of visual odometry is typically landmarks, represented by x_i. However, $\hat{f}_i$ also depends on the parameters $\boldsymbol{\theta}$ that we are trying to estimate. If not, $\hat{f}_i$ wouldn't tell us anything about the parameters, and the observations would be a waste.

The normal distribution assumption gives a new expression for our problem:

$$\boldsymbol{\theta}_{MAP} = \arg\min_{\boldsymbol{\theta}} \sum_{i}^{n} \frac{1}{2\sigma^2}\Big(\hat{f}_i - f(x_i, \boldsymbol{\theta})\Big)^2 = \arg\min_{\boldsymbol{\theta}} \sum_{i}^{n} \frac{1}{2\sigma^2}\Big(r_i(\boldsymbol{\theta})\Big)^2 \tag{3.5.31}$$

The variance factor $\frac{1}{2\sigma^2}$ doesn't affect the solution, so it can be dropped. By comparing the resulting problem to equation (3.5.1), we can see that it is equivalent to the Lucas Kanade problem derived from photometric error minimization, which means that our Bayesian derivation is complete.

3.6 Motion Prior

The trajectory of the camera pose parameter $\boldsymbol{\theta}$ has restrictions related to the physics of the camera or the camera platform. Information about these physics can therefore be used to gain information about the solution of Lucas-Kanade, leading to a more efficient Gauss Newton implementation. We refer to this information as a "motion prior".

Going back to equation (3.5.27) we keep the MAP problem structure by assiming a non-uniform distribution $p(\boldsymbol{\theta})$, indicating that we have some information about what we expect the solution $\boldsymbol{\theta}$ to look like, ie. the motion prior.

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; p(\hat{\boldsymbol{f}}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{3.6.1}$$

Equation (3.5.29) now becomes:

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} \; \Big(\prod_{i}^{n} p(\hat{f}_i|\boldsymbol{\theta})\Big)p(\boldsymbol{\theta}) \tag{3.6.2}$$

And equation (3.5.30) becomes:

$$\boldsymbol{\theta}_{MAP} = \arg\min_{\boldsymbol{\theta}} \; \Big(\sum_{i}^{n} -log(p(\hat{f}_i|\boldsymbol{\theta}))\Big) - log(p(\boldsymbol{\theta})) \tag{3.6.3}$$

We assume that $p(\boldsymbol{\theta})$ is distributed according to the normal distribution, with our motion prior encoded into the mean $\boldsymbol{\theta}_{mp}$ and the covariance matrix Σ_0:

$$\boldsymbol{\theta}_{MAP} = \arg\min_{\boldsymbol{\theta}} \Big(\sum_{i}^{n} \frac{1}{2\sigma^2} \Big(r_i(\boldsymbol{\theta}) \Big)^2 \Big) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_{mp})^T \Sigma_0^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta}_{mp}) \tag{3.6.4}$$

Finally, we write equation (3.6.4) more concisely and subsitute the relative covariance $\Sigma = \frac{1}{\sigma^2}\Sigma_0$:

$$\boldsymbol{\theta}_{MAP} = \arg\min_{\boldsymbol{\theta}} \Big(\sum_{i}^{n} \Big(r_i(\boldsymbol{\theta}) \Big)^2 \Big) + (\boldsymbol{\theta} - \boldsymbol{\theta}_{mp})^T \Sigma^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta}_{mp}) \tag{3.6.5}$$

The choice of Σ depends on the nature of the motion prior. However, regardless of the choice it is important to make Σ^{-1} positive definite, so that $\boldsymbol{x}^T \Sigma^{-1} \boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq 0$. This guarantees that the motion prior term in the cost function given by equation (3.6.5) is positive, and adding terms that are not positive to a non-linear cost function can reduce the convexity of the problem.

3.6.1 Motion Prior in Forward Additive Gauss-Newton

We derive the iterative solution to the MAP problem (3.6.5) using the forward additive Gauss-Newton method. In order to formulate the iterative Gauss-Newton problem, we substitute $\boldsymbol{\theta} = \boldsymbol{\theta}_k + \boldsymbol{\Delta\theta}$, which is derived from the forward additive normal equation (3.5.5):

$$\boldsymbol{\Delta\theta} = \arg\min_{\boldsymbol{\Delta\theta}} \Big(\sum_{i}^{n} \Big(r_i(\boldsymbol{\theta_k + \Delta\theta}) \Big)^2 \Big) + (\boldsymbol{\theta}_k + \boldsymbol{\Delta\theta} - \boldsymbol{\theta}_{mp})^T \Sigma^{-1} (\boldsymbol{\theta}_k + \boldsymbol{\Delta\theta} - \boldsymbol{\theta}_{mp}) \tag{3.6.6}$$

$$= \arg\min_{\boldsymbol{\Delta\theta}} \Big(\sum_{i}^{n} \Big(r_i(\boldsymbol{\theta_k + \Delta\theta}) \Big)^2 \Big) + (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k))^T \Sigma^{-1} (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k)) \tag{3.6.7}$$

To find the minimizer of the problem, we first linearize the residuals:

$$\boldsymbol{\Delta\theta} = \arg\min_{\boldsymbol{\Delta\theta}} \Big(\sum_{i}^{n} \Big(r_i(\boldsymbol{\theta}_k) + J_i(\boldsymbol{\theta}_k)\boldsymbol{\Delta\theta} \Big)^2 \Big) + (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k))^T \Sigma^{-1} (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k)) \tag{3.6.8}$$

Where J_i is the jacobian of the residual:

$$J_i(\boldsymbol{\theta}_k) = \left. \frac{\partial r_i}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta} = \boldsymbol{\theta}_k} \tag{3.6.9}$$

To obtain the first normal equation for the Gauss Newton algorithm, we differentiate equation (3.6.8), set it to zero and rearrange:

$$2 \Big(\sum_{i}^{n} J_i(\boldsymbol{\theta}_k)^T \Big) \Big(\sum_{i}^{n} \Big(r_i(\boldsymbol{\theta}_k) + J_i(\boldsymbol{\theta}_k)\boldsymbol{\Delta\theta} \Big) \Big) + 2\Sigma^{-1} (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k)) = \mathbf{0} \tag{3.6.10}$$

$$\Rightarrow \Big(J(\boldsymbol{\theta}_k)^T J(\boldsymbol{\theta}_k) + \Sigma^{-1} \Big) \boldsymbol{\Delta\theta} = -J(\boldsymbol{\theta}_k)^T \boldsymbol{r}(\boldsymbol{\theta}_k) + \Sigma^{-1} (\boldsymbol{\theta}_{mp} - \boldsymbol{\theta}_k) \tag{3.6.11}$$

Where J and r are defined according to:

$$J = \sum_{i}^{n} J_i, \quad r = \sum_{i}^{n} r_i \tag{3.6.12}$$

In the first normal equation (3.6.11), we have now shown how to include a motion prior into the forward additive Gauss-Newton algorithm.

The second normal equation is the forward additive update:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k + \boldsymbol{\Delta\theta} \tag{3.6.13}$$

3.6.2 Motion Prior In Inverse Compositional Gauss-Newton

When deriving the iterative, forward additive Gauss-Newton algorithm in section 3.6.1, we substituted the forward additive update $\boldsymbol{\theta} = \boldsymbol{\theta}_k + \boldsymbol{\Delta\theta}$ directly into equation (3.6.5). It is not as straight forward to derive the inverse compositional algorithm.

The inverse compositional update equation is given by:

$$w(\boldsymbol{x}, \boldsymbol{\theta}) \leftarrow w(\boldsymbol{x}, \boldsymbol{\theta}) \circ w(\boldsymbol{x}, \boldsymbol{\Delta\theta})^{-1} \tag{3.6.14}$$

However, in contrast to the forward additive algorithm, the inverse compositional error function is not obtained by simply inserting the update equation into the residual expression. Instead it splits up the motion update and the current motion estimate and applies them respectively to the template and the current image term in the residuals as per equation (3.5.10).

This is readily replicated for the photometric error part of equation (3.6.5). However, the motion prior term can't be formulated in the same way. We need to make a substitution based on the inverse compositional update in equation (3.6.14), as we did in the forward additive derivation. The candidate substitution is given by:

$$\boldsymbol{\theta} = w_{\boldsymbol{\theta}}(\boldsymbol{x}, w(\boldsymbol{x}, \boldsymbol{\theta}_k) \circ w(\boldsymbol{x}, \boldsymbol{\Delta\theta})^{-1}) \tag{3.6.15}$$

Where $w_{\boldsymbol{\theta}}$ denotes the inverse of w in the $\boldsymbol{\theta}$ field, so that we have:

$$w_{\boldsymbol{\theta}}(\boldsymbol{x}, w(\boldsymbol{x}, \boldsymbol{\theta})) = \boldsymbol{\theta} \tag{3.6.16}$$

The expression (3.6.15) is difficult to handle, so we approximate it using an expression which is similar to the one in equation (3.6.6):

$$w_{\boldsymbol{\theta}}(\boldsymbol{x}, w(\boldsymbol{x}, \boldsymbol{\theta}_k) \circ w^{-1}(\boldsymbol{x}, \boldsymbol{\Delta\theta})) \approx \boldsymbol{\theta}_k - \boldsymbol{\Delta\theta} \tag{3.6.17}$$

We substitute into the motion prior term in equation (3.6.5):

$$(\boldsymbol{\theta}_k - \boldsymbol{\Delta\theta} - \boldsymbol{\theta}_{mp})^T \Sigma^{-1} (\boldsymbol{\theta}_k - \boldsymbol{\Delta\theta} - \boldsymbol{\theta}_{mp}) \tag{3.6.18}$$

$$= (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_k - \boldsymbol{\theta}_{mp}))^T \Sigma^{-1} (\boldsymbol{\Delta\theta} - (\boldsymbol{\theta}_k - \boldsymbol{\theta}_{mp})) \tag{3.6.19}$$

The stationary point of the inverse compositional error function with the additional motion prior term is given by:

$$\Big(J(\boldsymbol{\theta}_k)^T J(\boldsymbol{\theta}_k) + \Sigma^{-1}\Big)\boldsymbol{\Delta\theta} = -J(\boldsymbol{\theta}_k)^T \boldsymbol{r}(\boldsymbol{\theta}_k) + \Sigma^{-1}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{mp}) \tag{3.6.20}$$

Which has identical motion prior terms as in equation (3.6.11), except that $\boldsymbol{\theta}_{mp}$ and $\boldsymbol{\theta}_k$ have switched place.

Inverse Compositional Motion Prior Error

The error introduced by the approximation in equation (3.6.17) makes it so that the incorporation of the motion prior in the inverse compositional Gauss-Newton algorithm doesn't exactly penalize divergence of the motion estimate from the motion prior, but rather of an approximation of what we expect the motion estimate to be at the end of each iteration.

The nature of the error depends on the warp parametrization. As an example, we investigate the error using the lie warp parametrization:

$$w(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\pi}(\exp(\boldsymbol{\theta})\boldsymbol{\pi}^{-1}(\boldsymbol{x})),\ \boldsymbol{x} \in \mathbb{R}^2,\ \boldsymbol{\theta} \in \mathfrak{se}(3) \tag{3.6.21}$$

Using this warp parametrization, the approximation (3.6.17) simplifies to:

$$\log(\exp(\boldsymbol{\theta}_k^\wedge)\exp(-\boldsymbol{\Delta\theta}^\wedge)) \approx \boldsymbol{\theta}_k^\wedge - \boldsymbol{\Delta\theta}^\wedge \tag{3.6.22}$$

Where the log function refers to the map from $SE(3)$ to $\mathfrak{se}(3)$ which was derived in section 3.3.10.

This approximation is only fulfilled if the matrices $\boldsymbol{\theta}_k^\wedge$ and $-\boldsymbol{\Delta\theta}^\wedge$ commute. In general, equation (3.6.22) approximates the Baker–Campbell–Hausdorff formula:

$$X, Y \in \mathbb{R}^{nxn} \Rightarrow$$

$$\log(\exp(X)\exp(Y)) = X + Y + \frac{1}{2}\big[X, Y\big] + \frac{1}{12}\big[X, \big[X, Y\big]\big] - \frac{1}{12}\big[Y, \big[X, Y\big]\big] + ... \tag{3.6.23}$$

The error between the approximation (3.6.22), which assumes $\log(\exp(X)\exp(Y)) = X + Y$, and the Baker-Campbell-Hausdorff formula consists of the increasing orders of commutators of X and Y:

$$e(X, Y) = \frac{1}{2}\big[X, Y\big] + \frac{1}{12}\big[X, \big[X, Y\big]\big] - \frac{1}{12}\big[Y, \big[X, Y\big]\big] + ... \tag{3.6.24}$$

We should get an idea of how large this error can be. We define the matrix norm $||\cdot||$:

$$\big|\big|X\big|\big| = \max_{||\boldsymbol{v}||\neq 0} \frac{\big|\big|X\boldsymbol{v}\big|\big|}{\big|\big|\boldsymbol{v}\big|\big|} \tag{3.6.25}$$

Then the commutator norm is limited by:

$$\big|\big|\big[X, Y\big]\big|\big| = \big|\big|XY - YX\big|\big| \leq 2\big|\big|X\big|\big| \cdot \big|\big|Y\big|\big| \tag{3.6.26}$$

Inserting $X = \boldsymbol{\theta}_k^\wedge$ and $Y = -\boldsymbol{\Delta\theta}^\wedge$ into the error function (3.6.24), we get:

$$e(\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge) = \frac{1}{2}\big[\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge\big] + \frac{1}{12}\big[\boldsymbol{\theta}_k^\wedge, \big[\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge\big]\big] - \frac{1}{12}\big[-\boldsymbol{\Delta\theta}^\wedge, \big[\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge\big]\big] + ... \tag{3.6.27}$$

Taking the norm of both sides and applying equation (3.6.26), we get:

$$e(\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge) \leq \big|\big|\boldsymbol{\theta}_k^\wedge\big|\big| \cdot \big|\big|\boldsymbol{\Delta\theta}^\wedge\big|\big| + \frac{1}{3}\big|\big|\boldsymbol{\theta}_k^\wedge\big|\big|^2 \cdot \big|\big|\boldsymbol{\Delta\theta}^\wedge\big|\big| - \frac{1}{3}\big|\big|\boldsymbol{\theta}_k^\wedge\big|\big| \cdot \big|\big|\boldsymbol{\Delta\theta}^\wedge\big|\big|^2 + ... \tag{3.6.28}$$

$$= \big|\big|\boldsymbol{\Delta\theta}^\wedge\big|\big| \cdot \big|\big|\boldsymbol{\theta}_k^\wedge\big|\big| \cdot \Big(1 + \frac{1}{3}\big|\big|\boldsymbol{\theta}_k^\wedge\big|\big| - \frac{1}{3}\big|\big|\boldsymbol{\Delta\theta}^\wedge\big|\big| + ...\Big) \tag{3.6.29}$$

This expression gives us an idea of the size of the error. Especially, for small steps $\boldsymbol{\Delta\theta}^\wedge$ the error will be negligible, $e(\boldsymbol{\theta}_k^\wedge, -\boldsymbol{\Delta\theta}^\wedge) \ll \big|\big|\boldsymbol{\theta}_k^\wedge - \boldsymbol{\Delta\theta}^\wedge\big|\big|$.

3.6.3 Choice of Motion Prior

No Motion

A simple choice of motion prior is to set $\boldsymbol{\theta}_{mp} \equiv \mathbf{0}$. This reflects the insight that the velocity of a camera is constrained, and we should avoid considering solutions to the problem that involves unrealistically large changes in the camera pose.

The matrix Σ^{-1} in equation (3.6.11) can in this case be chosen as a scalar or diagonal matrix. The entries of Σ^{-1} should be tuned so that the motion prior has a reasonable impact on the total photometric error.

No Acceleration / Inertia Based

Alternatively, we can use the pose between the last image pair as the motion prior, $\boldsymbol{\theta}_{mp} \equiv \boldsymbol{\theta}_{k-1}$ (given that the parameters we are currently trying to find is the pose $\boldsymbol{\theta}_k$ between the current image pair). This corresponds to a inertia based, constant velocity model, where we only want to consider solutons that don't imply an unrealistically large change in the translational and rotational velocities.

Again, Σ^{-1} can be chosen as a diagonal matrix, where the entries represent the inertia of each motion parameter. If an actual inertial model is available for the camera or whatever inertial system the camera is mounted on, then this can be used instead. However most implementations prefer the simpler diagonal matrix or even a scalar matrix, with tunable entries.

Sensor Based

A third choice of motion prior is to use measurements from sensors like gyro sensors, accelerometers and water pressure / depth sensors. A challenge with this choice is that we need a precise inter-sensor calibration in terms of time synchronization and the relative pose between the sensors and the camera. Given that these requirements are fulfilled, we discuss the incorporation of sensor data from different sensors into our motion prior model.

In order to discuss sensor based motion priors we need to decide on the motion parametrization. We choose the $\mathfrak{se}(3)$ motion parametrization from equation (3.5.18).

Gyro Sensor

Gyro sensors measure angular rates in three dimensions $(\varphi_x, \varphi_y, \varphi_z)$. These rates can be composed and accumulated over the time period $(t_k - t_{k-1})$ between two frame captures in order to obtain a rotation vector representing the rotation of the camera platform between the two frames. Conversion between the axis of the gyro sensor and the camera is needed.

The rotation vector with accumulated gyro measurements is equivalent to a $\mathfrak{so}(3)$ rotation, so it serves as a motion prior to the rotational part of a $\mathfrak{se}(3)$ pose representation. We denote the gyro based motion prior as $\boldsymbol{\omega}_{mp}$:

$$\boldsymbol{\omega}_{mp} = R_{cam}^{\varphi} \int_{t_{k-1}}^{t_k} \begin{bmatrix} \varphi_x(t) \\ \varphi_y(t) \\ \varphi_z(t) \end{bmatrix} dt \tag{3.6.30}$$

Where R_{cam}^{φ} is the rotation between the gyro sensors axis and the camera axis. t_k is the capture time of the current frame and t_{k-1} is the capture time of the previous frame, given that we are trying to solve the image alignment problem for the current and the previous frame. $\varphi_x(t)$ is the gyro measurement of the rotational drift around the x-axis of the gyro sensor, etc.

In order to discuss the covariance matrix Σ for the sensor based motion prior, we split it into two three by three covariance matrices:

$$\Sigma = \begin{bmatrix} \Sigma_{\boldsymbol{u}} & 0_{3\times 3} \\ 0_{3\times 3} & \Sigma_{\boldsymbol{\omega}} \end{bmatrix} \tag{3.6.31}$$

Here $\Sigma_{\boldsymbol{u}}$ is the (auto-)covariance matrix of the translational part of the motion prior and $\Sigma_{\boldsymbol{\omega}}$ is the (auto-)covariance of the rotational part of the motion prior. We have assumed the cross-covariance matrices between the rotational and translational estimates to be zero, which we will see is an approximation.

Since the correspondence between the integrated gyro measurements and the rotation representation of the motion prior is the identity map, this means that $\Sigma_{\boldsymbol{\omega}}$ simply can be chosen as a diagonal or scalar matrix. If we want to create the motion prior only based on gyro measurements, we can ignore the translational part of the motion prior by choosing $\Sigma_{\boldsymbol{u}}$ to be a scalar matrix with large uncertainty.

Pressure / Depth Sensor

Pressure sensors measure the water depth of the camera platform. For a given image alignment problem with frames taken at t_k and t_{k-1}, we can use the difference in depth estimates d_p in order to get a motion prior for the translational part of the $\mathfrak{se}(3)$ pose between the frames. However, the incorporation is not as direct as it was for the gyro sensor.

First, we need to rotate the depth vector $\boldsymbol{d}_p = [0, 0, d_p]$ along the rotation R_{cam}^{NED} between the world coordinate system (North East Down) and the camera axis. Assuming small acceleration, we can use an accelerometer to estimate the direction of gravity, and then use this to resolve the two degrees of freedom that are needed in order to convert the depth vector into the coordinate system of the camera. The third degree of freedom, given by rotation in the North-East plane is not needed since the depth vector goes along the up-down axis in the world coordinate system. We call the resulting rotation estimate R_{cam}^{world}.

After rotating $\boldsymbol{d_p}$ by R_{cam}^{world}, we multiply it with the inverse of the left jacobian V of $SO(3)$ in order to obtain the translational part of the $\mathfrak{se}(3)$ motion prior as per section 3.3.10. The left jacobian V of $SO(3)$ is given in equation (3.3.50), and its inverse V^{-1} is given in equation (3.3.55).

We need an $\mathfrak{so}(3)$ estimate for the rotation between the frames in order to calculate V. This can be extracted from the rotational part $\boldsymbol{\omega}_{k-1}$ of the previous pose estimate $\boldsymbol{\theta}_{k-1}$, or in case gyro measurements are available, we can use the gyro based rotational motion prior $\boldsymbol{\omega}_{mp}$ from equation (3.6.30).

We denote the translational part of the depth based motion prior as $\boldsymbol{u}_{mp}$:

$$\boldsymbol{u}_{mp} = V^{-1} R_{cam}^{world} \boldsymbol{d}_p = V^{-1} R_{cam}^{world} \begin{bmatrix} 0 \\ 0 \\ d_p \end{bmatrix}, \tag{3.6.32}$$

$$V^{-1} = I - \frac{1}{2}[\boldsymbol{\omega}_{mp}]_\times + \frac{1}{\omega_{mp}^2}\Big(1 - \frac{\omega_{mp}}{2}\frac{\sin\omega_{mp}}{1-\cos\omega_{mp}}\Big)[\boldsymbol{\omega}_{mp}]_\times^2,\ \omega_{mp} = \sqrt{\boldsymbol{\omega}_{mp}^T\boldsymbol{\omega}_{mp}} \tag{3.6.33}$$

Where $\boldsymbol{\omega}_{mp}$ is the motion prior of the rotational part of the pose, obtained from accumulated gyro measurements or from the previous rotation estimate $\boldsymbol{\omega}_{k-1}$.

To represent the fact that we know only the depth d_p in the translational vector $[0, 0, d_p]$, we use the uncertainty matrix Σ_p encoding large uncertainties along the translational axis that are not estimated and a relatively low uncertainty in the water depth, or up down, direction:

$$\Sigma_p = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & .02 \end{bmatrix} \tag{3.6.34}$$

The entries should be tuned so that the confidence in the depth measurements is reasonably represented when the motion prior is incorporated into the photometric error cost function.

In order to obtain $\Sigma_{\boldsymbol{u}}$, we need to transform the uncertainty matrix Σ_p using the derivative of the map from equation (3.6.32):

$$\Sigma_{\boldsymbol{u}} = \Big(\frac{\partial \boldsymbol{u}_{mp}}{\partial \boldsymbol{d}_p}\Big)\Sigma_p\Big(\frac{\partial \boldsymbol{u}_{mp}}{\partial \boldsymbol{d}_p}\Big)^T \tag{3.6.35}$$

$$= \big(V^{-1}R_{cam}^{world}\big)\Sigma_p\big(V^{-1}R_{cam}^{world}\big)^T \tag{3.6.36}$$

Note that $\Sigma_{\boldsymbol{u}}$ is invertible because all of its matrix factors are invertible. Apart from V^{-1}, R_{cam}^{world} is invertible since it is a rotation matrix, and Σ_p is invertible since it is a diagonal matrix with non-zero trace. Together with a trivially invertible scalar matrix $\Sigma_{\boldsymbol{\omega}}$, this means we can insert the inverted complete motion prior covariance matrix Σ^{-1} from equation (3.6.31) into the normal equation (3.6.11), together with the motion prior $\boldsymbol{\theta}_{mp} = [\boldsymbol{u}_{mp}, \boldsymbol{\omega}_{mp}]^T$.

We can also show that $\Sigma_{\boldsymbol{u}}$ is positive definite, making Σ^{-1} also positive definite. The important observations to make is that $(V^{-1})^T(V^{-1})$ is positive definite, according to equation (3.3.60), and that $(R_{cam}^{world})(R_{cam}^{world})^T = I$ is also positive definite. Together with the diagonal and positive definite Σ_p, this is enough to show that $\Sigma_{\boldsymbol{u}}$ is positive definite. The positive definiteness of Σ^{-1} is important because it guarantees that the motion prior term in the cost function given by equation (3.6.5) is positive, and adding terms that are not positive to a non-linear cost function can reduce the convexity of the problem.

Since we used the rotational motion prior in order to calculate V in equation (3.6.33), the assumption of zero cross-covariance which was made in equation (3.6.31) is merely an approximation. If the uncertainty in the rotational motion prior is low enough, we can still discount the cross-covariances. If not, then the full cross-covariance matrices must be derived and calculated analogous to equation (3.6.35).

3.6.4 Initialization of Gauss-Newton

At the first iteration of Gauss-Newton we have the choice of what the initial value of the pose estimate should be. This choice can be made using the same motion information which is encoded in the motion prior discussed in the previous sections.

A good choice will initialize the estimate close to the final solution, which is the relative pose between the two frames considered in the image alignment problem. This will increase the chances that the algorithm starts in a convex area around the solution, which again increases the chances of converging to the correct solution. A good initial estimate will also lead to faster convergence.

The choice of initialization point usually comes down to the same choice as that of motion prior (3.6.3). Using the notation from equation (3.6.11), we can either put $\boldsymbol{\theta}_k \equiv \mathbf{0}$, starting at zero motion. Or we can put $\boldsymbol{\theta}_k \equiv \boldsymbol{\theta}_{k-1}$, starting at the pose between the previous two frames. Alternatively, we can use sensor-data like gyro, accelerometer and water pressure measurements in order to find out what we expect the solution to be.

3.7 Bitplanes

Bitplanes[1] are a type of geometrically differentiable feature descriptors. They can be used in Lucas-Kanade in order to gain the same kind of robustness to illumination changes, inaccuracies in camera parameters etc. that indirect VO methods have.

3.7.1 Bitplane Definition

The bitplane descriptor is the canonical LBP (Local Binary Patterns) descriptor[41]. For a given pixel, its bitplane descriptor is found by comparing its intensity with those of the eight neighbouring pixels. The descriptor is then given by an 8-bit number θ where the individual bits are mapped to the eight intensity checks / inequalities for each of the neighbours:

$$\phi(I, \boldsymbol{x}) = \sum_{i=1}^{8} 2^{i-1} \big[I(\boldsymbol{x} + \boldsymbol{\Delta x}_i) < I(\boldsymbol{x}) \big] \tag{3.7.1}$$

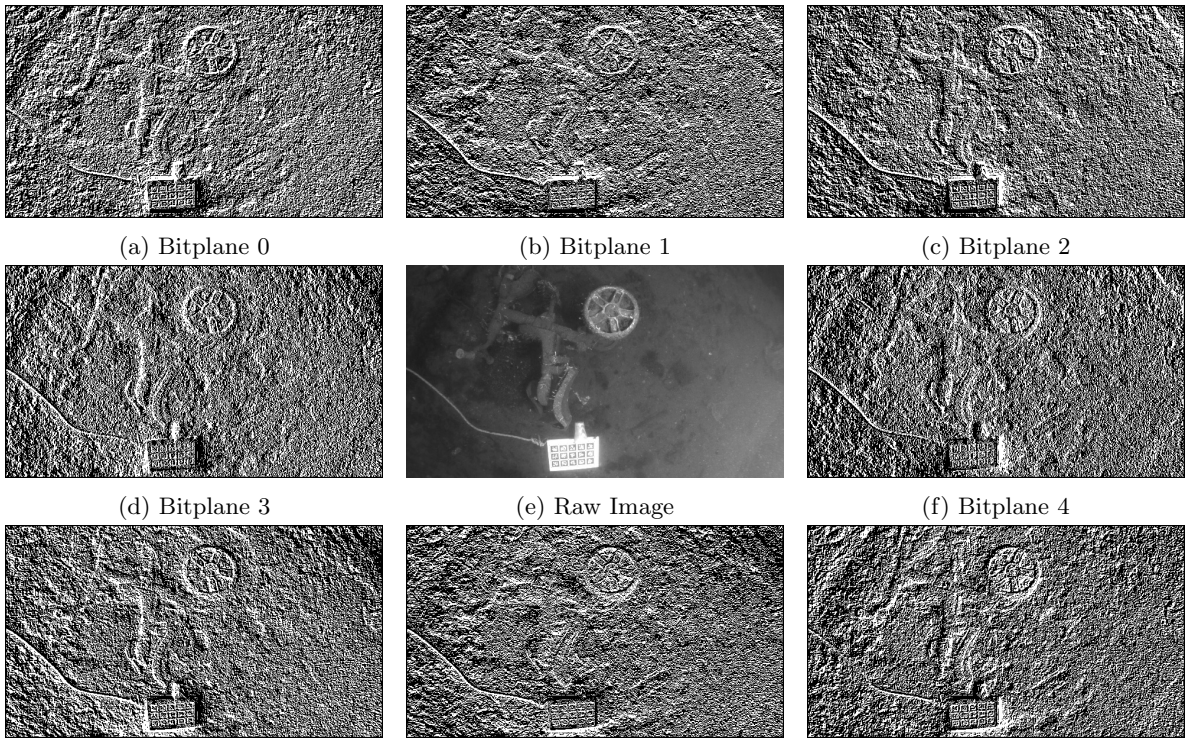
Where $\{\boldsymbol{\Delta x})_i\}$ is the set of the eight relative coordinate displacements between a pixel and its immediate neighbours. The exact order of the displacements $\{\boldsymbol{\Delta x})_i\}$, ie. if we go cock-wise or counter clock-wise through the neighbours etc, is inconsequential and can be chosen arbitrarily[1]. The choice of binary comparison operator $(\{>, \geq, <, \leq\}$ is also arbitrary.

The C/C++ code for computing the bitplane descriptor is given in listing (3.1).

Listing 3.1: Bitplane Descriptor Implementation

```
phi_I[x] =
    ((I[x - stride - 1] < I[x] ? 1 : 0) << 0) |
    ((I[x - stride    ] < I[x] ? 1 : 0) << 1) |
    ((I[x - stride + 1] < I[x] ? 1 : 0) << 2) |
    ((I[x          - 1] < I[x] ? 1 : 0) << 3) |
    ((I[x          + 1] < I[x] ? 1 : 0) << 4) |
    ((I[x + stride - 1] < I[x] ? 1 : 0) << 5) |
    ((I[x + stride    ] < I[x] ? 1 : 0) << 6) |
    ((I[x + stride + 1] < I[x] ? 1 : 0) << 7) ;
```

A bitplane representation is obtained by evaluating the bitplane descriptor for each pixel in an image and then store the descriptors in an array with similar placement to the original pixels. If the intensities



(a) Bitplane 0 (b) Bitplane 1 (c) Bitplane 2

(d) Bitplane 3 (e) Raw Image (f) Bitplane 4

(g) Bitplane 5 (h) Bitplane 6 (i) Bitplane 7

Figure 3.8: Visualization of the bitplanes for an image.
The number of each bitplane corresponds to the bit number in phi_I from listing 3.1.

of the original image are represented with 8-bit integers, the bitplane representation will take the same amount of memory as the image, and the descriptor (3.1) will be a direct map between the memory of the image and its resulting bitmap representation. In fact the listing (3.1) is in practice an image transform referred to as the census transform, and it can be implemented efficiently using parallell computation (3.8.1).

The bitplane representation also supports more complicated descriptors with longer words. This may improve the robustness of the image alignment further, at the cost of a less time and memory efficient implementation.

The bitplane representation can be thought of as a composition of eight different images, called bitplanes, with only one bit per pixel, corresponding to one of the intensity inequalities. In figure 3.8 we have calculated the bitplanes for an image. The bitplanes in figure 3.8 are arranged so that the pixel displacement of each bitplane corresponds to its placement in relation to the raw image. Note how each bitplane is basically an image derivative in the direction that goes away from the center of the raw image and through the center of the bitplane.

3.7.2 Lucas-Kanade with Bitplanes

Instead of operating on direct image intensities as in the original Lucas-Kanade based image alignment schemes presented in section 3.5, we can align bitplane representations of the original images:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \sum_{x} \Big[\phi(I, w(\boldsymbol{x}, \boldsymbol{\theta})) - \phi(T, \boldsymbol{x})\Big]^2 \tag{3.7.2}$$

However using the normal subtraction operator $(-)$ in equation (3.7.2) does not make sense. The most significant bits of the descriptors $\phi(I, w(\boldsymbol{x}, \boldsymbol{\theta}))$ and $\phi(T, \boldsymbol{x})$ will contribute exponentially more to the difference than the least significant bits, which is unreasonable since the order of the bits is arbitrary.

Instead we consider the Hamming distance operator $\delta_{hamming}$:

$$\delta_{hamming}(\phi_1, \phi_2) = \sum_{i=1}^{8} \Big[\frac{\phi_1 \odot 2^{i-1}}{2^{i-1}} \oplus \frac{\phi_2 \odot 2^{i-1}}{2^{i-1}} \Big] \tag{3.7.3}$$

Where $\odot$ denotes the bitwise AND operator and $\oplus$ denotes bitwise XOR.

The term $\frac{\phi_1 \odot 2^{i-1}}{2^{i-1}}$ is the ith bit in ϕ_1, and we denote it as ϕ_{1i} to get a simpler exression:

$$\delta_{hamming}(\phi_1, \phi_2) = \sum_{i=1}^{8} \Big[\phi_{1i} \oplus \phi_{2i} \Big] \tag{3.7.4}$$

We can also write the Hamming distance as a sum of l_2-norms:

$$\delta_{hamming}(\phi_1, \phi_2) = \sum_{i=1}^{8} \sqrt{(\phi_{1i} - \phi_{2i})^2} = \sum_{i=1}^{8} \Big|\Big| \phi_{1i} - \phi_{2i} \Big|\Big|_2 \tag{3.7.5}$$

Since the terms $(\phi_{1i} - \phi_{2i})$ are all either 1, 0 or -1, we can drop the square-root:

$$\delta_{hamming}(\phi_1, \phi_2) = \sum_{i=1}^{8} (\phi_{1i} - \phi_{2i})^2 \tag{3.7.6}$$

If we now take the terms $(\phi_{1i} - \phi_{2i})$ to be the residuals in Lucas-Kanade, we can rewrite the image alignment problem in terms of the hamming distance. In equation (3.7.8) we denote the ith bit of $\phi(I, \boldsymbol{x})$ as $\phi_i(I, \boldsymbol{x})$:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \sum_{x} \Big[\delta_{hamming}\big(\phi(I, w(\boldsymbol{x}, \boldsymbol{\theta})), \phi(T, \boldsymbol{x})\big) \Big] \tag{3.7.7}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{x} \sum_{i=1}^{8} \Big[\phi_i(I, w(\boldsymbol{x}, \boldsymbol{\theta})) - \phi_i(T, \boldsymbol{x}) \Big]^2 \tag{3.7.8}$$

This problem can be solved using multi-channel Gauss-Newton, where the eight bits of the bitplane descriptors are the eight channels. The solution is the usual Gauss-Newton update:

$$\boldsymbol{\Delta\theta} = -(J^T J)^{-1} J^T r(\boldsymbol{\theta}), \tag{3.7.9}$$

with

$$J = \sum_{x} \sum_{i=1}^{8} \Big[\nabla_w \phi_i(I, w(\boldsymbol{x}, \boldsymbol{\theta})) \frac{\partial w(\boldsymbol{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big], \tag{3.7.10}$$

$$r(\boldsymbol{\theta}) = \sum_{x} \sum_{i=1}^{8} \Big[\phi_i(I, w(\boldsymbol{x}, \boldsymbol{\theta})) - \phi_i(T, \boldsymbol{x}) \Big] \tag{3.7.11}$$

And the forward additional update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\Delta\theta} \tag{3.7.12}$$

Extension to inverse compositional Gauss-Newton is analogous to section 3.5.2.

3.8 Parallell Computation

Visual odometry techniques are by nature limited in performance by the amount of data they are able to process. In recent years it has not been possible to simply rely on the ever increasing single threaded cpu performance to handle increasing amounts of data. Instead more and more VO methods rely on parallell, or SIMD (Single Instruction, Multiple Data) computation.

3.8.1 Vectorization

Vectorization is a type of SIMD computing which allows computer programs to perform common instructions, like for instance arithmetic and logical operations, on multiple contiguous data elements simultaneously. It is implemented through special instruction sets, some of them with accompanying compiler intrinsics so that developers can make explicit use of the instructions without writing assembly code. When the relevant option flags are given to the compiler it may also perform automatic vectorization, so that SIMD instructions are inserted in suitable places.

Examples of instruction sets for vectorization are SSE2 for x86 processors and ARM NEON for ARM. These two instruction sets in particular have a large amount of similar operations to the point where you can define a partial one-to-one dictionary and translate betweeen the compiler intrinsics, allowing developers to write somewhat portable, explicit vectorized code.

3.8.2 GPGPU

GPGPU (General Purpose Graphics Processing Unit) is the use of a GPU for tasks that are traditionally handled by the CPU. GPUs are normally used to handle graphics processing, like for instance rendering and applying after effects on a scene in a computer game. However, their design makes them useful for different problems, as long as they are efficiently formulated as a SIMD problem.

GPGPU can be implemented through the use of traditional graphics libraries like OpenGL or Vulkan. However, there is software specifically meant for GPGPU which eases development and provides better code optimization. Popular GPGPU software modules includes Cuda, which is very popular among scientists for its performance and functionality but only works on NVIDIA GPUs, and OpenCL, which works on GPUs from a wide range of different vendors and also targets other kinds of computing units, like DSPs and FPGAs.

Chapter 4: Implementation

In this chapter we describe the proposed visual odometry approach. The goal of the algorithm is to estimate in real time the 6DoF pose, or egomotion, of a camera in relation to the environment or structure captured by the camera. Since the algorithm only relies on monocular vision, the translational part of the resulting pose estimate is up to scale, meaning that we don't estimate its scale.

We will describe the algorithm by going through the consequtive steps that newly captured images go through before they cause an update to the estimated camera pose.

4.1 Multi-Scale Image Pyramid

On receiving a new image, a pyramid representation of the image is created. The first level of the pyramid consists of the original image or a downscaled version of it. Then each of the subsequent levels is a half-sampled version of the previous level.

The multi-scale pyramid is created in order to be used for feature detection and image alignment.

Bi-linear subsampling is chosen for its computational simplicity compared to more complex sampling techniques like Gaussian blur. Computational efficiency is gained using explicit vectorization (3.8.1).

We choose a 4-level pyramid with level 1 having a resolution of 640x360 pixels and level 4 having a resolution of 80x45 pixels.

We also compute a corresponding pyramid using the bitplane representation from section 3.7. For each level of the bitplane pyramid we compute the bitplane representation of the corresponding level of the raw image pyramid.

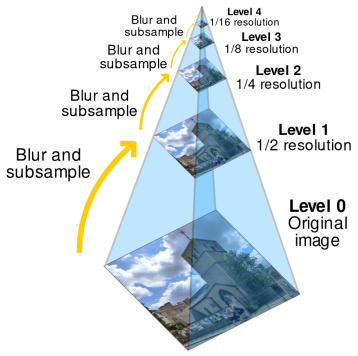


Figure 4.1: A five-level multi-scale image pyramid

4.2 Feature Detection

FAST as described in section 3.1 is used for feature detection because of its computational efficiency. Additional efficiency is again gained with vectorization (3.8.1).

The features define image regions that are subsequently tracked and used for image alignment. Image regions where we already have a depth estimate, or where we are currently trying to find the depth are not considered in the search for FAST corners.

In order to choose the most useful among overlapping or close features, we use non-maximal suppression based on a FAST score (3.1.2).

Features are detected at all levels of the image pyramid (4.1) so that we find corners with multiple scales. Features detected at lower resolutions correspond to corners that appeas large in the image, while features detected at higher resolutions correspond to small corners. Additional non-maximal suppression is done across scales in fixed rectangular image areas, so that we end up with at most one detected feature in each area.

In figure 4.2 we see multi-scale FAST features with non-maximal suppression. The test pixels correspond to the circles in the figure, highlighting the fact that FAST corners detected at larger resolutions correspond to larger image features. In figure 4.3 we have also drawn the areas used for non-maximal suppression.

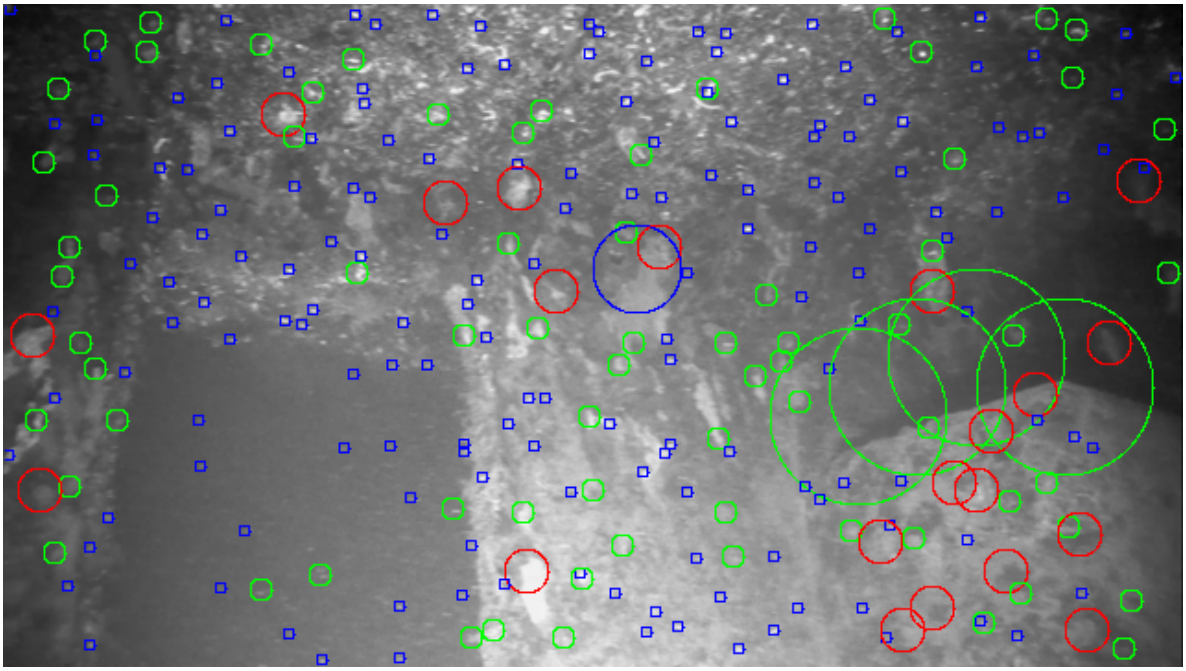


Figure 4.2: FAST corners detected at multiple levels with cross-scale non-maximum suppression. The circles correspond to the circles made by the test pixels, with the center being the corner pixel. FAST corners detected at lower resolution have larger circles because pixels correspond to larger geometries at lower resolution. The different sized circles have different colors so they are easier to disinguish.

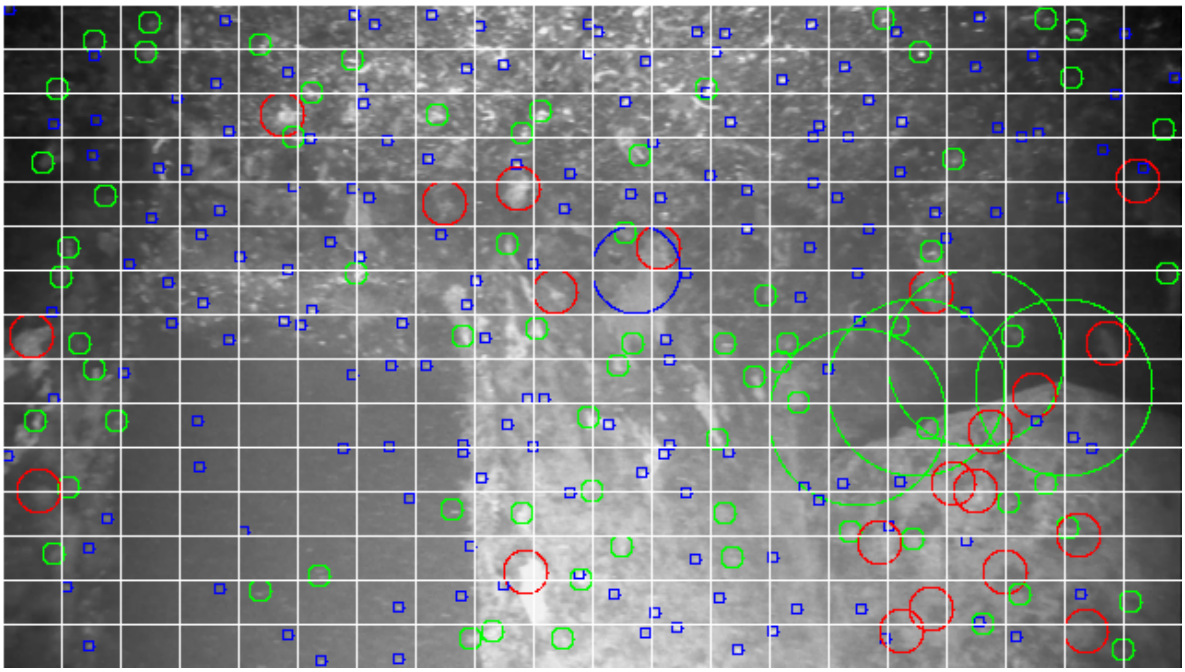


Figure 4.3: FAST corners detected at multiple levels. The white rectangles are the areas used for cross scale non-maximal suppression. Note that there is at most one corner in each rectangle.

4.3 Image Depth Estimation

Image depth refers to the distance between the camera and the features observed in the images taken by the camera. It is not to be confused with the water depth, which is the distance from the water surface to the camera. Image depth estimation is done in the VO backend, and it will only be discussed briefly because of this thesis focus on the VO frontend.

Depth estimation in monocular VO applications is often done using epipolar geometry. Given two images and a relative pose between them as estimated by a VO frontend, ie. using image alignment or similar, as well as an image region with matches in both images, we can estimate the depth of the image region using epipolar geometry.

In order to match image patches we use Lucas Kanade based optical flow.

The depth estimates generated from the epipolar matches of the same image region in consecutive frames are often noisy and they need to be filtered. For the approach described in this thesis we have opted for the bayesian inference filter described in Vogiatzis and Hernandez[49]. When enough epipolar matches in consecutive frames have been found for a given image region, the filters depth estimate will converge, and it is at this point that we begin to consider the image region for image alignment.

4.4 6DoF Image Alignment

Image alignment is needed in order to align a newly captured image with the last image so we can estimate the motion between them. This is the core of the implemented VO algorithm.

For a given pair of consecutive images we assume that we have a set of tracked image locations $\{\boldsymbol{x}\}$ in the old image for which the depth d has been found according to section 4.3. We align the images by solving the Lucas-Kanade problem (3.5).

We parametrize the warp in the same way as in equation (3.5.18), so that we get the full 6DoF pose between the images as output. When the solution has converged, we use the same 6DoF pose in order to transform the image locations $\{\boldsymbol{x}\}$ that were used for image alignment into the newest frame.

When solving the Lucas Kanade problem, we first solve the problem using the lowest resolution of the pyramid representation of each image. Then we move on to the next level of the pyramid, with twice the resolution, and we initialize the problem using the solution from the previous level. We stop when we have found the solution for the level with the highest resolution in the pyramid.

The reason for solving Lucas-Kanade at each consecutive level like this is because low resolution images typically give a more convex optimization problem which is easier to solve. This is shown later on in section 5.2. The solutions found at the lower resolution levels will only be rough estimates, but they will be useful in order to provide good initializations to the less convex problems at the higher resolutions. A good initialization will often lead to correct convergence even of a highly non-convex problem, since the initial estimate will be close to a convex valley around the correct solution.

At the lowest resolution levels, we opt for aligning bitplane representations instead of raw images as per section 3.7.2 in order to get an initial pose estimate which is robust to illumination changes etc. The higher resolution levels are aligned using normal image alignment. Later on, in section 5.2.1 we show why aligning bitplanes at the lower resolutions yields robustness against lighting changes and weakly textured scenes.

Section (4.4.1) will describe the Gauss-Newton implementation used for image alignment. The bitplane alignment of the lower resolution levels uses the same implementation with the modifications described in section 3.7.2.

4.4.1 Gauss Newton

We employ the inverse compositional Gauss-Newton algorithm from section 3.5.2.

We define patches of four by four pixels around each image location $\boldsymbol{x}$ in the previous image frame. The combination of these patches is chosen as the template T, while the newly captured image is chosen as I (at the current pyramid level). For each image location $\boldsymbol{x}$ in the template, we enumerate the 16 pixels in its patch according to:

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}, \ i = 1, ..., 16 \tag{4.4.1}$$

We also define the patch map Φ:

$$\Phi_i(\boldsymbol{x}) = \boldsymbol{x}_i \tag{4.4.2}$$

We choose the Lie algebra / rotation vector parameterization from equation (3.5.18):

$$w(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{\pi}(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x})) \tag{4.4.3}$$

Where $\boldsymbol{\xi}$ is the parameterization of the current pose estimate between the last and current frame, $\boldsymbol{\xi}^\wedge \in \mathfrak{se}(3)$, and where $\boldsymbol{\pi}$ is the projection function defined in equation (3.4.7).

We use the inverse compositional normal equations to solve the Lucas-Kanade problem:

$$J^T J \boldsymbol{\Delta\xi} = -J^T r(\boldsymbol{\xi}), \tag{4.4.4}$$

$$\exp(\boldsymbol{\xi}^\wedge) \leftarrow \exp(\boldsymbol{\xi}^\wedge)\exp(-\boldsymbol{\Delta\xi}^\wedge) \tag{4.4.5}$$

Where we have substituted our pose parametrization into equation (4.4.5).

We define the residuals of our Lucas Kanade problem in terms of image patches and the $\mathfrak{se}(3)$ parametrization:

$$r(\boldsymbol{\xi}) = \sum_{\boldsymbol{x}} \sum_{i=0}^{16} \Big[T(\boldsymbol{x}_i) - I(\Phi_i(\boldsymbol{\pi}(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x})))) \Big] \tag{4.4.6}$$

Note that we iterate through the patches Φ of each image location. The jacobian takes a similar form:

$$J = \sum_{\boldsymbol{x}} \sum_{i=0}^{16} \Big[\nabla T(\boldsymbol{x}_i) \frac{\partial \boldsymbol{\pi}(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x}))}{\partial \boldsymbol{\xi}} \Big|_{\boldsymbol{\xi}=\mathbf{0}} \Big] \tag{4.4.7}$$

We investiate the structure of the Jacobian by applying the chain rule to J:

$$J = \sum_{\boldsymbol{x}} \sum_{i=0}^{16} \Big[\nabla T(\boldsymbol{y}) \Big|_{\boldsymbol{y}=\Phi_i(\boldsymbol{\pi}(\exp(\mathbf{0}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x})))} \cdot \frac{\partial \boldsymbol{\pi}(\boldsymbol{q})}{\partial \boldsymbol{q}} \Big|_{\boldsymbol{q}=\exp(\mathbf{0}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x})} \cdot \frac{\partial(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x}))}{\partial \boldsymbol{\xi}} \Big|_{\boldsymbol{\xi}=\mathbf{0}} \Big] \tag{4.4.8}$$

We substitute $\boldsymbol{p} = \boldsymbol{\pi}^{-1}(\boldsymbol{x})$, with $\boldsymbol{p} = [p_x, p_y, p_z]^T$, and we proceed to expand the factors in equation (4.4.8) one by one, beginning at the right:

$$\left.\frac{\partial(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p})}{\partial\boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\mathbf{0}} = \left.(I \mid -[\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}]_\times)\right|_{\boldsymbol{\xi}=\mathbf{0}} = (I \mid -[\exp(\mathbf{0}^\wedge)\boldsymbol{p}]_\times) = (I \mid -[\boldsymbol{p}]_\times)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & p_z & -p_y \\ 0 & 1 & 0 & -p_z & 0 & p_x \\ 0 & 0 & 1 & p_y & -p_x & 0 \end{bmatrix} \tag{4.4.9}$$

Where we have used equation (3.3.62) to obtain the derivative of a transformation with respect to its Lie algebra representation.

The derivative of the projection function $\boldsymbol{\pi}$ is given by:

$$\left.\frac{\partial\boldsymbol{\pi}(\boldsymbol{q})}{\partial\boldsymbol{q}}\right|_{\boldsymbol{q}=\exp(\mathbf{0}^\wedge)\boldsymbol{p}} = \left.\frac{\partial\boldsymbol{\pi}(\boldsymbol{q})}{\partial\boldsymbol{q}}\right|_{\boldsymbol{q}=\boldsymbol{p}} = \begin{bmatrix} \frac{f_x}{p_z} & 0 & -\frac{f_x p_x}{p_z^2} \\ 0 & \frac{f_y}{p_z} & -\frac{f_y p_y}{p_z^2} \end{bmatrix} \tag{4.4.10}$$

Finally, the image gradient ∇T is given by:

$$\left.\nabla T(\boldsymbol{y})\right|_{\boldsymbol{y}=\Phi_i(\boldsymbol{\pi}(\exp(\mathbf{0}^\wedge)\boldsymbol{\pi}^{-1}(\boldsymbol{x})))} = \left.\nabla T(\boldsymbol{y})\right|_{\boldsymbol{y}=\boldsymbol{x}_i} = [T_{ix}, T_{iy}], \tag{4.4.11}$$

$$T_{ix} = \frac{\partial T(\boldsymbol{x}_i)}{\partial x_{i1}},\ T_{iy} = \frac{\partial T(\boldsymbol{x}_i)}{\partial x_{i2}}$$

We multiply the vector (4.4.11) and the matrices (4.4.10) and (4.4.9) to get the full jacobian:

$$J = \sum_x \sum_{i=0}^{16} \begin{bmatrix} \frac{T_{ix} f_x}{p_z} \\ \frac{T_{iy} f_y}{p_z} \\ -T_{ix}\frac{f_x p_x}{p_z^2} - T_{iy}\frac{f_y p_y}{p_z^2} \\ -T_{iy} f_y - p_y \frac{T_{ix} f_x p_x + T_{iy} f_y p_y}{p_z^2} \\ T_{ix} f_x - p_x \frac{T_{ix} f_x p_x + T_{iy} f_y p_y}{p_z^2} \\ \frac{T_{iy} f_y p_x}{p_z} - \frac{T_{ix} f_x p_y}{p_z} \end{bmatrix}^T \tag{4.4.12}$$

4.4.2 Iteratively Re-weighted Residuals

We use the Huber loss function from section 3.5.5 to suppress outliers in the residuals $r_{\boldsymbol{x}}(\boldsymbol{\xi}) = T(\boldsymbol{x}) - I(\boldsymbol{\pi}(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{x}))$. We put $W = diag([\omega_{\boldsymbol{x}}^h])$, where the weight elements $\omega_{\boldsymbol{x}}^h$ are calculated according to the huber weight function from equation (3.5.23), $\omega_{\boldsymbol{x}}^h = \omega^h(r_{\boldsymbol{x}}(\boldsymbol{\xi}))$. We choose $h = 1.345\sigma_{MAD}(\{r_{\boldsymbol{x}}\})$, where σ_{MAD} is the MAD scale estimator from equation (3.5.25).

We incorporate the diagonal weight matrix W in the normal equation equation (4.4.4) according to equation (3.5.22):

$$J^T W J \boldsymbol{\Delta\xi} = -J^T W r(\boldsymbol{\xi}) \tag{4.4.13}$$

We recalculate W after each iteration of Gauss Newton. Unfortunately, this means that the computational efficiency gained from using the inverse compositional Gauss Newton formulation to precompute the Hessian approximation $J^T J$, is now lost since we can't precompute W. However, the inverse compositional formulation still helps since we at least are able to precompute the Jacobian J.

4.4.3 Motion Prior

We add discrete time indices k to equation (4.4.5), where k is the index of the current frame:

$$\exp(\boldsymbol{\xi}_k^\wedge) \leftarrow \exp(\boldsymbol{\xi_k}^\wedge)\exp(-\boldsymbol{\Delta\xi}^\wedge) \tag{4.4.14}$$

Then we choose the inertia based motion prior from section 3.6.3, which consists of the pose estimated from the previous frame pair, $\boldsymbol{\xi}_{k-1}$. Using this motion prior, solutions which deviate too far from the pose estimated from the previous image pair will be penalized. We modify the iteratively re-weighted normal equation (4.4.13) to include the motion prior:

$$\left(J^T W J + \Sigma^{-1}\right)\boldsymbol{\Delta\xi} = -J^T W r(\boldsymbol{\xi}_k) + \Sigma^{-1}(\boldsymbol{\xi}_k^\wedge - \boldsymbol{\xi}_{k-1}^\wedge) \tag{4.4.15}$$

We set Σ^{-1} to be a scalar matrix:

$$\Sigma^{-1} \equiv \lambda I = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \tag{4.4.16}$$

Where the tuning parameter λ decides the relative contribution of the motion prior term to the photometric error.

4.5 Initialization

Before image alignment can be done, an initial estimate of the image depth in the scene must be done. This estimate is done according to Faugeras et al.[18]. We use the first two image frames and assume the scene is piecewise planar in order to estimate a homography that we then can extract the initial motion from. We use this motion estimate to triangulate the depths of the initial image patches.

The depth estimates do not include scale due to the monocular nature of the algorithm. Thus the scale for the rest of the depth estimates that the algorithm produces, as well as the trajectory, are decided by what scale we decide on in these first two views.

4.6 Visualization

The progress of the proposed algorithm can be visualized in multiple ways. For instance, in chapter 5 we discuss trajectory and residual plots, which provide valuable information about the algorithms progress. Figure (4.4) shows a visualization that highlights the movement of the aligned image patches (red) as well as the candidate patches (blue) that do not have a converged depth estimate yet, as per section 4.3.

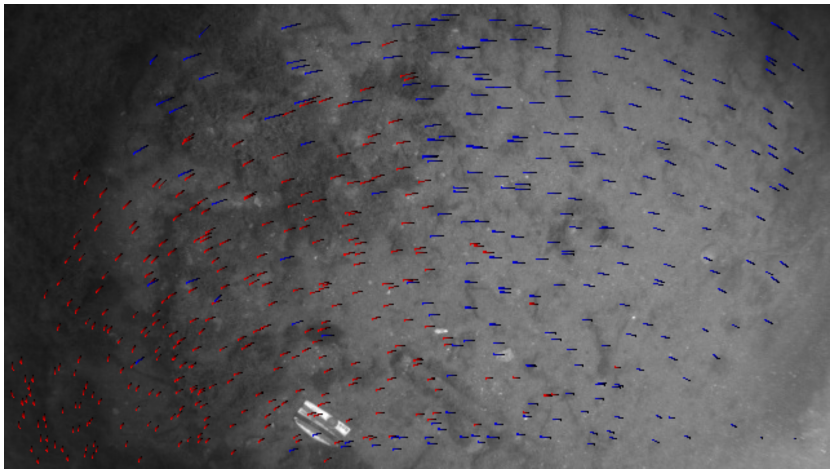


Figure 4.4: Visualization of the estimated transformation of aligned image patches (red) and the matches of candidate patches with, as of yet, unknown depth (blue). The visualization is from the execution of the implemented algorithm on the first Harbor sequence in the Aqualoc dataset[19].

Chapter 5: Results and Evaluation

In this chapter we showcase the functionality of the implemented algorithm as well as limitations and failure scenarios. We justify the use of multi-resolution image-alignment as well as bitplane alignment at the lower resolutions.

5.1 Trajectory Estimation

Only a few datasets for testing the trajectory estimation performance of underwater VO algorithms have been made available, including Aqualoc[19], the ACFR (Australian Centre for Field Robotics) Marine Robotics dataset and the Underwater caves sonar and vision data set[34].

Custom datasets were created using the Blueye Pioneer underwater drone. A third-party photogrammetry software called Agisoft Metashape was used in order to generate high precision trajectory estimates using the videos and sensor data captured from the drone. This kind of software benefits from being able to use the entire video at once, as well as not having to satify real time constraints. The estimates generated in Agisoft Metashape will serve as our ground truth when evaluating the proposed algorithms trajectory estimates.

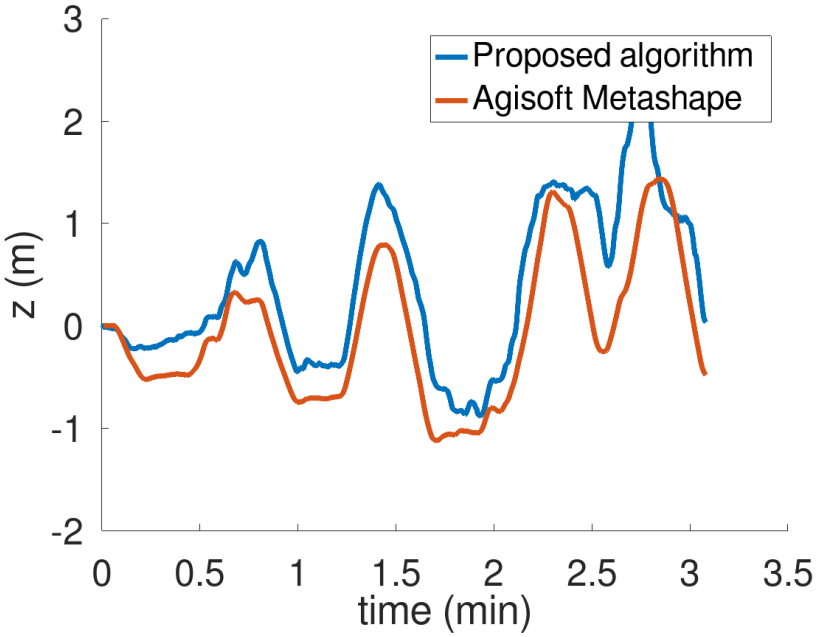
We refer to one of the custom datasets generated from data gathered by the Blueye Pioneer as the dock side series. It consists of a three minute long video of the side of a dock, filmed around 4 meters below the surface. The plots in figure 5.1 show the estimated trajectory using the proposed algorithm, as well as the estimate calculated in Agisoft Metashape.

From figure 5.1a we see that the water depth estimates for the dock side series are fairly consistent. However, observing the complete trajectory in figure 5.1b, we see that the proposed algorithm drifts a little with time.

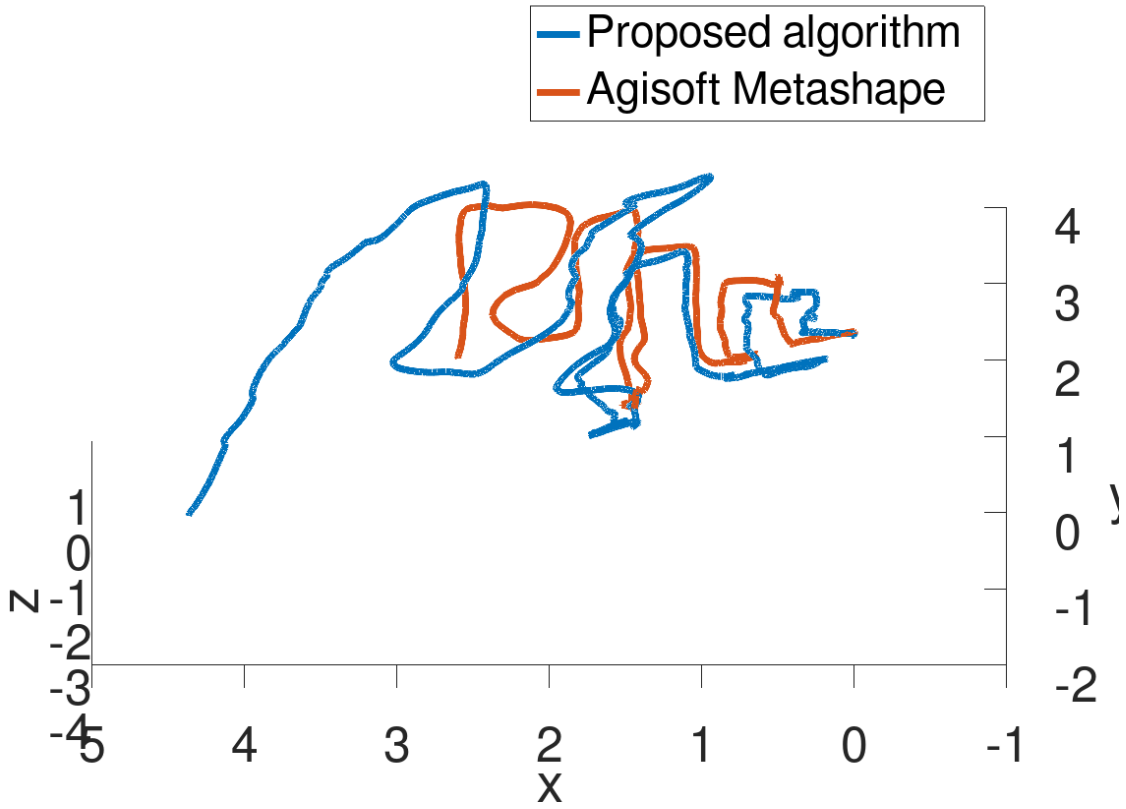
A video showing the progress of the proposed algorithm on the dock side sequence is available at https://youtu.be/oUHgMTFfRXk. The video shows the estimated trajectory of the camera as well as the video sequence itself.

The video sequence is overlayed with blue and red dots, as seen in the screenshot in figure 5.2. The blue spots indicate the position of candidate patches which are tracked using optical flow in order to estimate their depth in a bayesian inference filter as per section 4.3. The red spots correspond to the image patches that are being used for 6DoF image alignment as per section 4.4.

By comparing the live trajectory from the video with the apparant movement of the camera, it seems like the algorithm occasionally mistakes a part of the rotational movement for translation. This explains why the water depth estimates in figure 5.1a are more accurate than the xy coordinates in figure 5.1b, because the camera was mostly rotated in the xy-plane and not in the up-down / water depth direction.



(a) Estimated water depth from the dock side series, depth being reative to the initial depth of the camera.



(b) Estimated camera trajectory from the dock side series

Figure 5.1: Trajectory estimates for the dock side series captured on the Blueye Pioneer underwater drone.

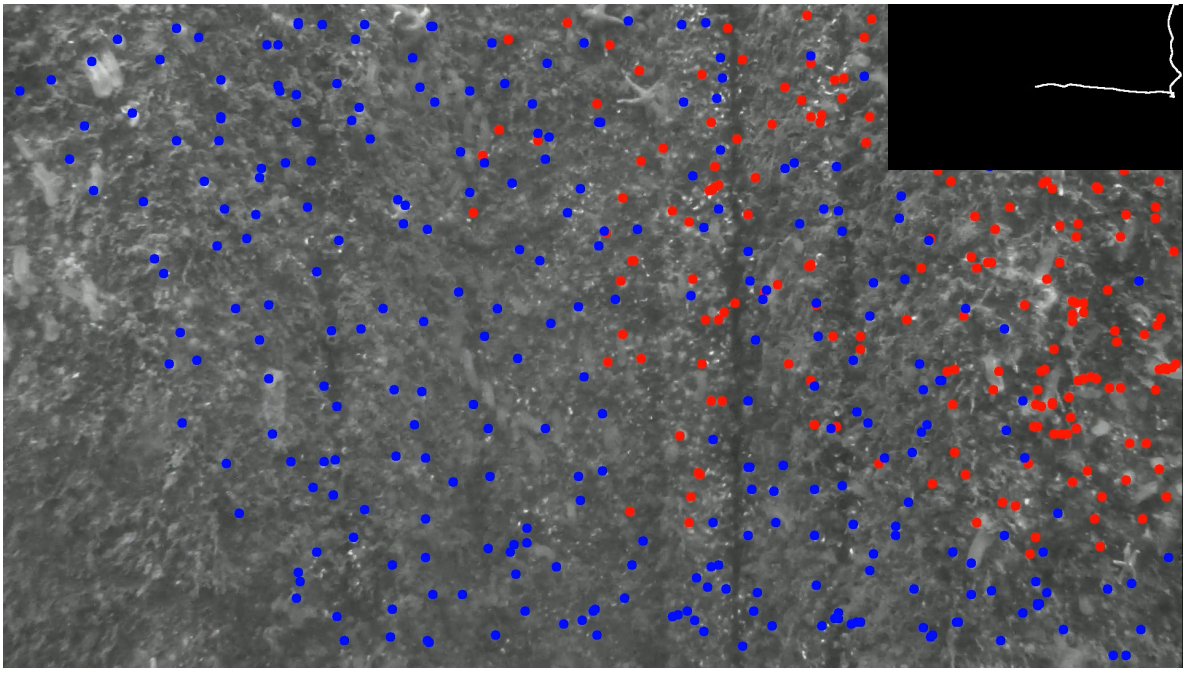
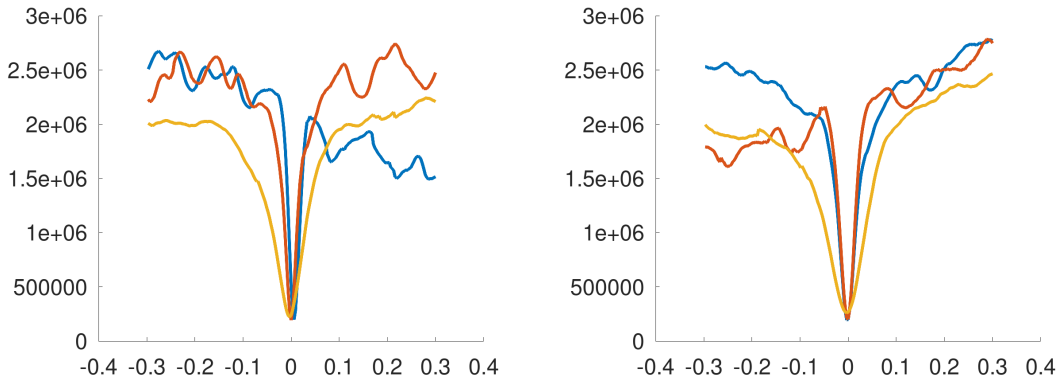


Figure 5.2: Screenshot from https://youtu.be/oUHgMTFfRXk showing the progress of the proposed algorithm on the "dock side" series. The blue and red dots correspond respectively to the image patches for which we are trying to find the depth / camera distance, and the image patches which are already being used for 6DoF image alignment.

5.2 Analysis of Residuals

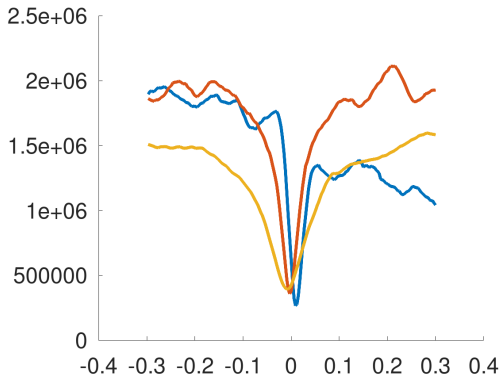
Figure (5.3) shows residuals for the implemented raw image alignment (4.4). It is apparant that the higher levels, which correspond to lower resolutions, have larger convex valleys around the converged estimate than the lower levels, which correspond to higher resolutions.

The corresponding bitplane based residuals are shown in figure 5.4. They have the same basic valley form as their raw image based counterparts, but they are narrower and less smooth, providing an image alignment problem which is more difficult to solve.

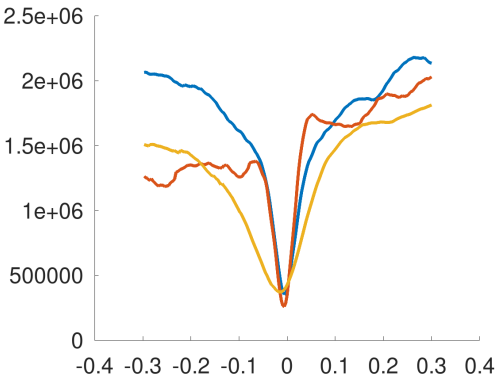


(a) Rotation residuals, level 1

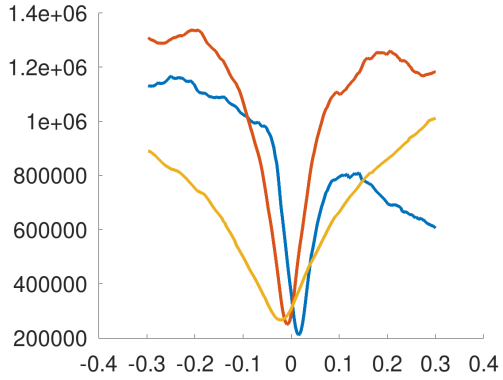
(b) Translation residuals, level 1



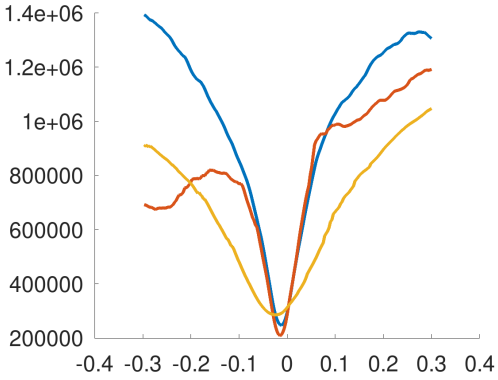
(c) Rotation residuals, level 2



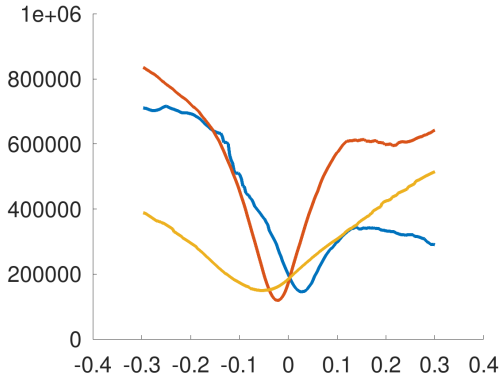
(d) Translation residuals, level 2



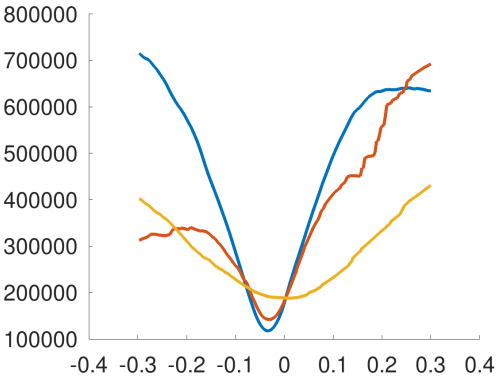
(e) Rotation residuals, level 3



(f) Translation residuals, level 3

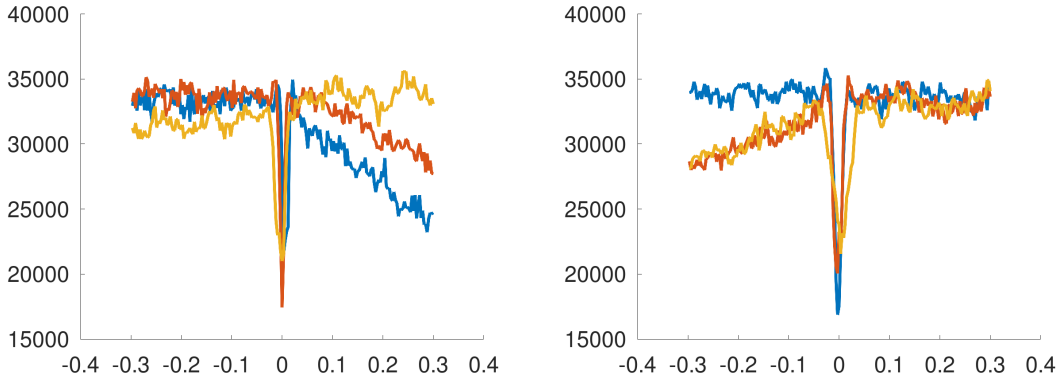


(g) Rotation residuals, level 4



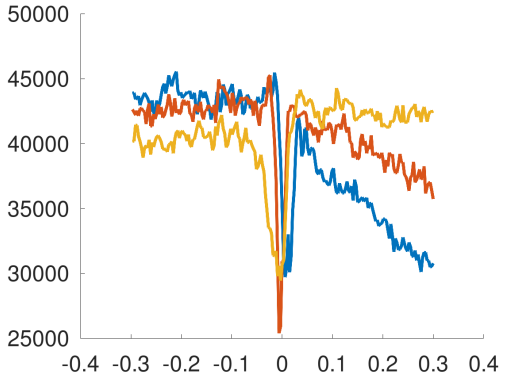
(h) Translation residuals, level 4

Figure 5.3: Raw image residuals.
The x-axis represents the size of the pertubation from the converged $\mathfrak{se}(3)$ pose estimate.

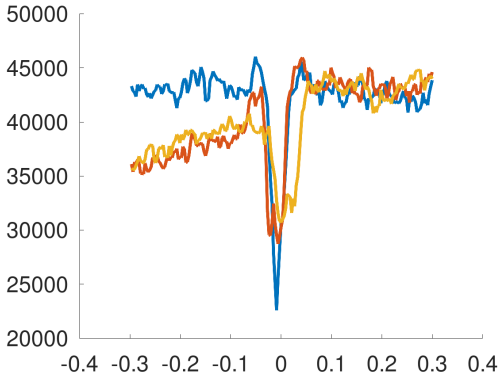


(a) Rotation residuals, level 1

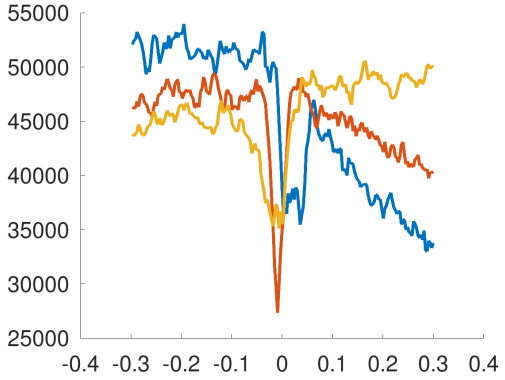
(b) Translation residuals, level 1



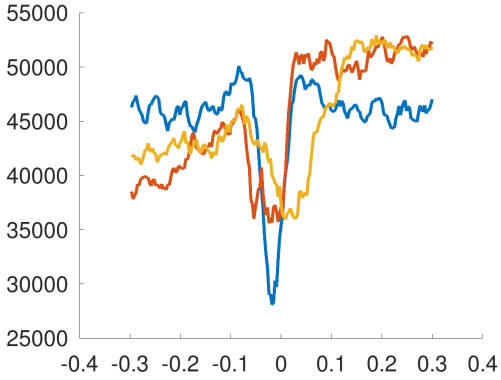
(c) Rotation residuals, level 2



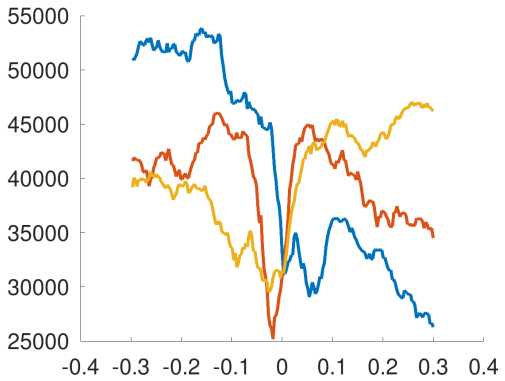
(d) Translation residuals, level 2



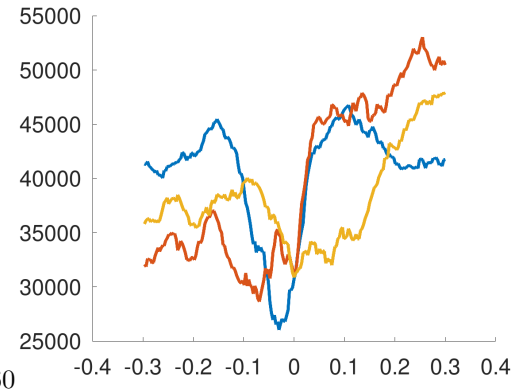
(e) Rotation residuals, level 3



(f) Translation residuals, level 3



(g) Rotation residuals, level 4



(h) Translation residuals, level 4

Figure 5.4: Bitplane residuals.
The x-axis represents the size of the pertubation from the converged $\mathfrak{se}(3)$ pose estimate.

5.2.1 Image Alignment Failure Scenario

The proposed algorithm aligns bitplane representations of the lower resolution multi-resolution pyramid levels as described in section 4.4. The reason for this choice is the failure of the image alignment operation in certain scenarios, which we will investigate here.

We pick a challenging frame in the dock side series gathered on the Blueye Pioneer drone. The frame is depicted in figure 5.5. It is in the middle of a camera exposure change, so the intensity over the entire image changes dramatically from frame to frame. We have calculated both the bitplane and raw image based residuals for the frame, while only using raw image based alignment for choosing the solution at all resolution levels.

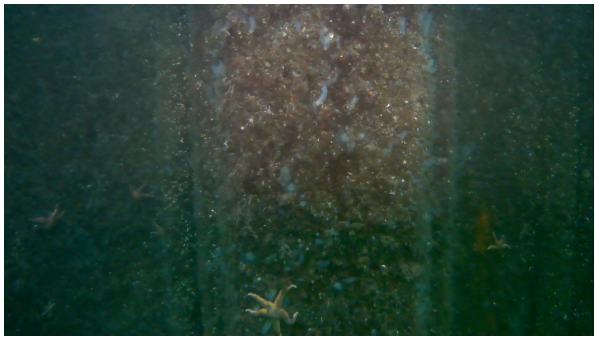


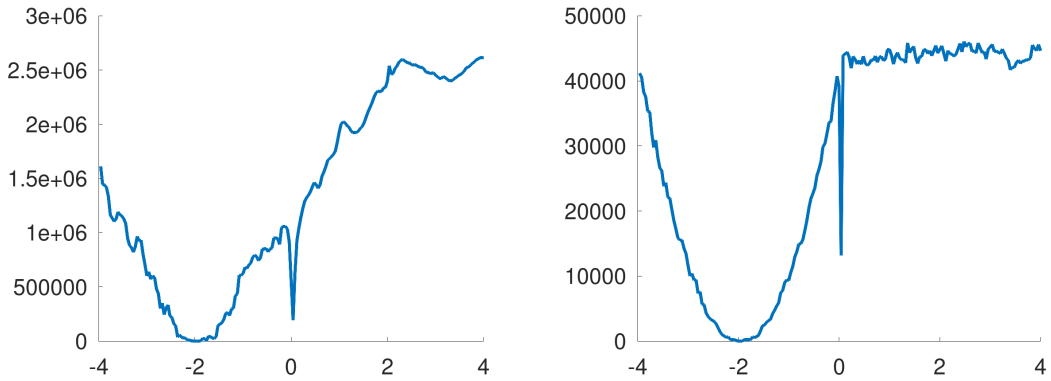
Figure 5.5: A frame from the dock side series which is in the middle of an illumination change generated by the initial exposure change in the camera.

From the raw image residuals on the left in figure 5.6 we see that there are two local minimas. One is at zero pertubation. This is the solution which the raw image based alignment has settled on. The other minima is at around -2. This turns out to be the correct solution, which means that the raw image alignment has failed significantly.

Comparing the raw image residuals with the bitplane based residuals at the right in figure 5.6, we see that the bitbplane residuals have a more clear minima. Especially on the lower resolution levels (level 3 and 4) the bitplane residuals make it much clearer what the correct solution is.

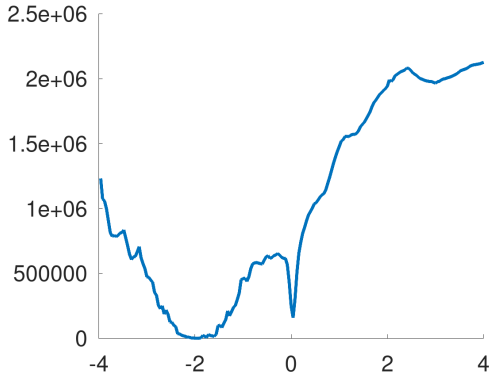
This is an extreme scenario, and it was chosen in order to highlight the robustness of bitplane based image alignment. Less extreme instances of the same issue were also observed consistently through the "Harbor" sequences in the Aqualoc dataset[19] due to the weakly textured seabed.

We found that for a four or five level multi-resolution pyramid, using bitplane alignment at the lowest two or three resolutions gave the most robust image alignment.

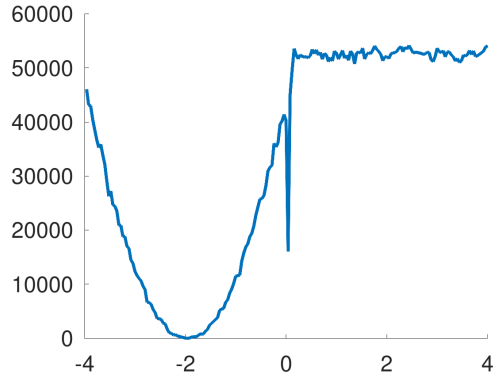


(a) Raw image residuals, level 1

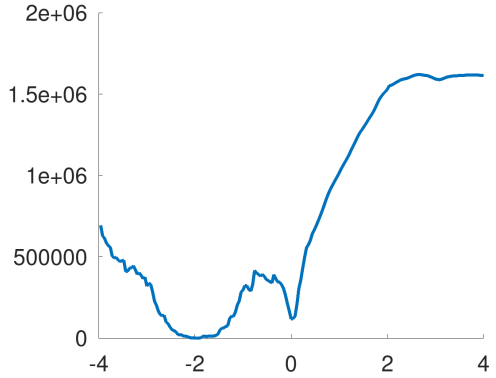
(b) Bitplane residuals, level 1



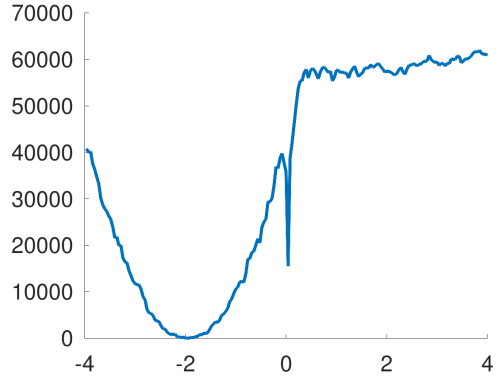
(c) Raw image residuals, level 2



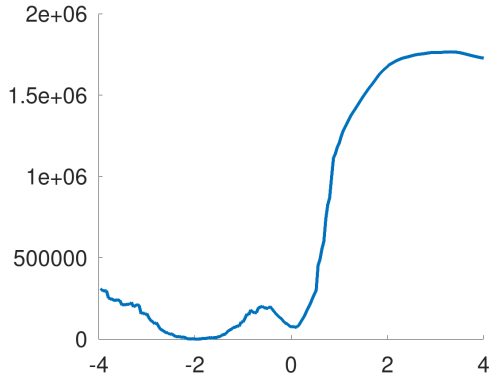
(d) Bitplane residuals, level 2



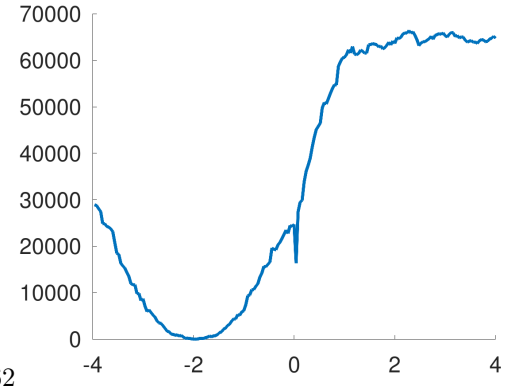
(e) Raw image residuals, level 3



(f) Bitplane residuals, level 3



(g) Raw image residuals, level 4



(h) Bitplane residuals, level 4

Figure 5.6: Residuals for one of the translation dimensions in the raw image alignment failure scenario. The x-axis represents the size of the pertubation from the converged $\mathfrak{se}(3)$ pose estimate.

5.3 Feature Detection Tuning

Feature detection is used in the proposed algorithm to split images into sparse regions. This comes with an element of uncertainty around how many features will be detected at each frame, and it becomes problematic in scenes with low texture.

This is evident in the first of the "Harbor" sequences in the Aqualoc dataset. A largely monotone seafloor makes it difficult to extract high quality features. Halfway through the sequence, the lack of features leads to divergence in the proposed algorithm.

In order to overcome the challenges of the sequence we remove the FAST-score threshold for the feature detection (section 4.2) and consider all feature candidates that are local maximas in terms of FAST-score. This gives us the trajectory estimate in figure 5.7. Here there is no divergence.

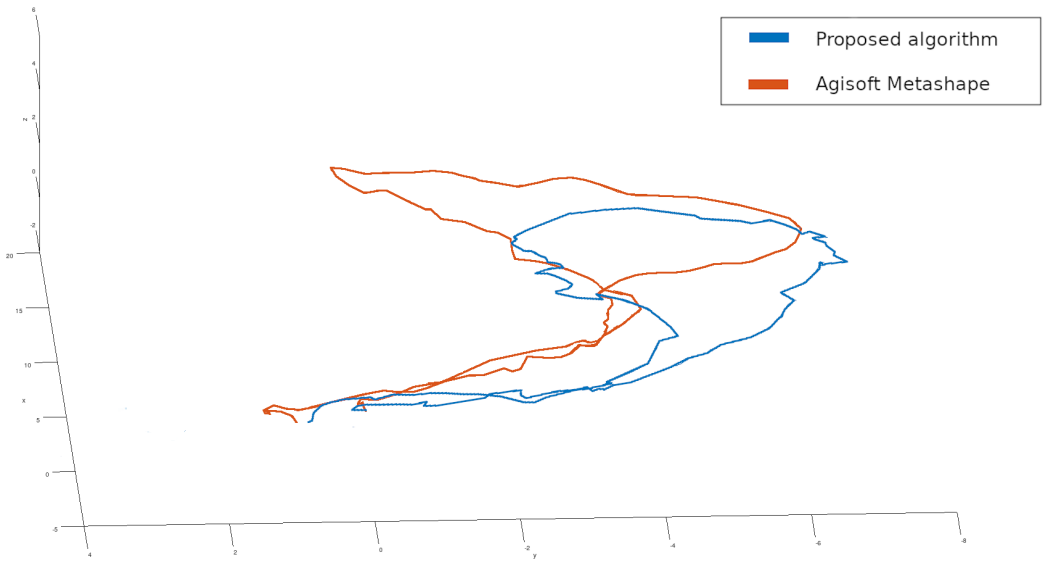


Figure 5.7: Trajectory estimates for "Harbor" sequence number 1 in the Aqualoc dataset[19].

Chapter 6: Conclusion and Further Work

Through this thesis we have described the theoretical background and implementation of a monocular visual odometry algorithm for real time underwater navigation. We have also evaluated the algorithms functionality and performance, and we have given justification for the design choices.

We found that the computationally efficient, sparse image alignment from Forster et al.[10] could be extended with bitplane alignment in order to gain illumination invariance and robustness to low-textured environments. Based on the residual plots, we found it best to use bitplane alignment for low resolution levels and raw image alignment for high resolution levels.

We also found that the FAST score detection threshold was better to be disposed with in certain scenarios in order to at least find some features in weakly textured areas.

In order to reduce the drift in trajectory estimate as observed in section 5.1, we could consider extending the visual odometry algorithm, at the cost of higher complexity. One extension would be to incorporate IMU and pressure sensor measurements and use the sensor based motion prior from section 3.6.3. This could however lead to instability if there are calibration issues and the measurements don't converge to a common consensus.

Another possibility would be to introduce keyframes, which are special frames kept in memory for multiple consecutive frames after they are first captured. New frames would be aligned with the previous keyframe instead of the previous frame, and the pose estimate would then hopefully drift slower due to the less frequent update rate of the reference frame. This would also come at the cost of a more challenging image alignment problem because the photo-consistency assumption would be less valid at higher perspective differences.

A more elaborate extension would be to implement bundle adjustment, aligning observations from multiple keyframes at once and refining their depth estimates along with the camera pose in one large optimization problem[8]. Bundle adjustment is considered state of the art for the purpose of reducing drift, especially scale drift, in monocular VO algorithms.

In section 5.3 we found that the FAST-score threshold normally used in feature detection can lead to divergence in low textured underwater sequences because of the lack of features. By removing the FAST-score threshold we guaranteed that enough features are considered for tracking.

This feature detection scheme is in practice similar to the way Alismail et al.[2] subsample pixels using a bitplane based saliency measure. By using bitplanes instead of FAST-features for feature detection, the computational burden of computing FAST-features could be avoided, and the overall complexity of the algorithms implementation would be reduced.

Other loss functions than the Huber loss implemented in section 4.4.2 could be investigated. In general, Huber loss is a t-distributon based loss function which exhibits relatively conservative outlier

suppression. We could design a custom t-distribution with more aggressive outlier suppression[11]. This could help to further reduce the effect of underwater-related image distortion.

The proposed algorithms implicit protection against marine snow could be expanded with explicit marine snow removal algorithms like Farhad et al.[17] or Koziarski et al.[29]. This is an example of the first strategy for dealing with outliers discussed in section 3.5.5. It could be extended into a more complete image enhancement stage where effects related to light attenuation and water photography in general are compensated for[32]. Interestingly, since light attenuation depends on the distance between scene and camera, as mentioned in section 3.4.2, image depth information can potentially be shared betweeen the proposed visual odometry algorithm and the image enhancement algorithm in order to compensate more efficiently for light attenuation.

Finally, although some of the design choices of the proposed algorithm are implicitly beneficial with respect to computational performance, further measures like GPGPU computation or similar could be considered in order to improve real-time performance on embedded devices. Device specific considerations should be taken because of the wide range of resources that are available on different devices.

Bibliography

[1] Hatem Alismail, Brett Browning, and Simon Lucey. "Bit-Planes: Dense Subpixel Alignment of Binary Descriptors". In: *CoRR* abs/1602.00307 (2016). arXiv: 1602.00307. URL: http://arxiv.org/abs/1602.00307.

[2] Hatem Alismail, Brett Browning, and Simon Lucey. "Direct Visual Odometry using Bit-Planes". In: *CoRR* abs/1604.00990 (2016). arXiv: 1604.00990. URL: http://arxiv.org/abs/1604.00990.

[3] Simon Baker and Iain Matthews. "Lucas-Kanade 20 Years On: A Unifying Framework". In: *International Journal of Computer Vision* 56.3 (Mar. 2004), pp. 221–255.

[4] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. DOI: 10.1017/9781316671528.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. ECCV'06. Graz, Austria: Springer-Verlag, 2006, pp. 404–417. DOI: 10.1007/11744023_32. URL: http://dx.doi.org/10.1007/11744023_32.

[6] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *ECCV*. 2010.

[7] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (Nov. 1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[8] Yu Chen, Yisong Chen, and Guoping Wang. *Bundle Adjustment Revisited*. 2019. arXiv: 1912.03858 [cs.CV].

[9] Kevin Christensen and Martial Hebert. "Edge-Direct Visual Odometry". In: *CoRR* abs/1906.04838 (2019). arXiv: 1906.04838. URL: http://arxiv.org/abs/1906.04838.

[10] Davide Scaramuzza Christian Forster Matia Pizzoli. *SVO: Fast Semi-Direct Monocular Visual Odometry*. https://www.ifi.uzh.ch/dam/jcr:e9b12a61-5dc8-48d2-a5f6-bd8ab49d1986/ICRA14_Forster.pdf. 2014.

[11] Mahmut Kutlukaya Christian Hennig. *SOME THOUGHTS ABOUT THE DESIGN OF LOSS-FUNCTIONS*. http://www.homepages.ucl.ac.uk/~ucakche/papers/henniglossfu.pdf. 2006.

[12] Oliver Demetz, David Hafner, and Joachim Weickert. "The Complete Rank Transform: A Tool for Accurate and Morphologically Invariant Matching of Structures". In: Jan. 2013. DOI: 10.5244/C.27.50.

[13] James Diebel. "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors". In: *Matrix* 58 (Jan. 2006).

[14] Ethan Eade. *Lie Groups for 2D and 3D Transformations*. http://ethaneade.com/lie.pdf. 2017.

[15] Johannes Ebert. *VAN EST'S EXPOSITION OF CARTAN'S PROOF OF LIE'STHIRD THEOREM, EXPLAINED BY JOHANNES EBERT*. https://ivv5hpp.uni-muenster.de/u/jeber_02/talks/lieIII.pdf. 2016.

[16] R. M. Eustice, O. Pizarro, and H. Singh. "Visually Augmented Navigation for Autonomous Underwater Vehicles". In: *IEEE Journal of Oceanic Engineering* 33.2 (Apr. 2008), pp. 103–122. ISSN: 2373-7786. DOI: 10.1109/JOE.2008.923547.

[17] Fahimeh Farhadifard. "Marine Snow Detection and Removal : Underwater Image Restoration using Background Modeling". In: 2017.

[18] Olivier Faugeras and F. Lustman. "Motion and Structure from Motion in a Piecewise Planar Environment". In: *International Journal of Pattern Recognition and Artificial Intelligence - IJPRAI* 02 (Sept. 1988). DOI: 10.1142/S0218001488000285.

[19] Maxime Ferrera et al. "AQUALOC: An Underwater Dataset for Visual-Inertial-Pressure Localization". In: *arXiv e-prints*, arXiv:1910.14532 (Oct. 2019), arXiv:1910.14532. arXiv: 1910.14532 [cs.CV].

[20] Maxime Ferrera et al. "Real-time Monocular Visual Odometry for Turbid and Dynamic Underwater Environments". In: *CoRR* abs/1806.05842 (2018). arXiv: 1806.05842. URL: http://arxiv.org/abs/1806.05842.

[21] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692.

[22] John Fox and Sanford Weisberg. *Robust Regression*. http://users.stat.umn.edu/~sandy/courses/8053/handouts/robust.pdf. 2013.

[23] Adrian Galdran et al. "Automatic Red-Channel underwater image restoration". In: *Journal of Visual Communication and Image Representation* 26 (Nov. 2014). DOI: 10.1016/j.jvcir.2014.11.006.

[24] Guillermo Gallego and Anthony Yezzi. *A compact formula for the derivative of a 3-D rotation inexponential coordinates*. https://arxiv.org/pdf/1312.0788. 2014.

[25] Chris Harris and Mike Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference*. 1988, pp. 147–151.

[26] Matthew Johnson-Roberson et al. "Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys". In: *Journal of Field Robotics* 27.1 (2010), pp. 21–51. DOI: 10.1002/rob.20324. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20324. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20324.

[27] A. Kim and R. Eustice. "Pose-graph visual SLAM with geometric model selection for autonomous underwater ship hull inspection". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 1559–1565. DOI: 10.1109/IROS.2009.5354132.

[28] Georg Klein and David W. Murray. "Parallel Tracking and Mapping for Small AR Workspaces". In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality* (2007), pp. 225–234.

[29] Michał Koziarski and Boguslaw Cyganek. "Marine Snow Removal Using a Fully Convolutional 3D Neural Network Combined with an Adaptive Median Filter: ICPR 2018 International Workshops, CVAUI, IWCF, and MIPPSNA, Beijing, China, August 20-24, 2018, Revised Selected Papers". In: Jan. 2019, pp. 16–25. ISBN: 978-3-030-05791-6. DOI: 10.1007/978-3-030-05792-3_2.

[30] Chang-Ryeol Lee and Kuk-Jin Yoon. "Monocular Visual Odometry with a Rolling Shutter Camera". In: *CoRR* abs/1704.07163 (2017). arXiv: 1704.07163. URL: http://arxiv.org/abs/1704.07163.

[31] Anthony M.Bloch and Arieh Iserlesy. *Commutators of Skew-symmetric Matrices*. http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2004_04.pdf. 2004.

[32] Priyanka Madhumatke. "Underwater Image Enhancement Techniques: A Review". In: *International Journal for Research in Applied Science and Engineering Technology* V (Aug. 2017), pp. 1574–1580. DOI: 10.22214/ijraset.2017.8223.

[33] I. Mahon et al. "Efficient View-Based SLAM Using Visual Loop Closures". In: *Trans. Rob.* 24.5 (Oct. 2008), pp. 1002–1014. ISSN: 1552-3098. DOI: 10.1109/TRO.2008.2004888. URL: https://doi.org/10.1109/TRO.2008.2004888.

[34] Angelos Mallios et al. “Underwater caves sonar data set”. In: *The International Journal of Robotics Research* 36 (Oct. 2017), p. 027836491773283. DOI: 10.1177/0278364917732838.

[35] Holly Mandel. *WHAT DOES A LIE ALGEBRA KNOW ABOUT A LIE GROUP?* http://math.uchicago.edu/~may/REU2016/REUPapers/Mandel.pdf. 2016.

[36] et al. Mohamad Motasem Nawaf. *Experimental Comparison of Open Source Visual-Inertial-Based StateEstimation Algorithms in the Underwater Domain.* https://afrl.cse.sc.edu/afrl/publications/public_html/papers/IROS19_0500_FI.pdf. 2019.

[37] et al. Mohamad Motasem Nawaf. *Underwater Photogrammetry and Visual Odometry.* https://res.mdpi.com/bookfiles/edition/786/article/807/Underwater_Photogrammetry_and_Visual_Odometry.pdf?v=0. 2018.

[38] Philippe Moutarlier and Raja Chatila. “An Experimental System for Incremental Environment Modelling by an Autonomous Mobile Robot”. In: *The First International Symposium on Experimental Robotics I*. Berlin, Heidelberg: Springer-Verlag, 1990, pp. 327–346. ISBN: 3-540-52182-8. URL: http://dl.acm.org/citation.cfm?id=645621.661279.

[39] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *CoRR* abs/1502.00956 (2015). arXiv: 1502.00956. URL: http://arxiv.org/abs/1502.00956.

[40] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. “DTAM: Dense tracking and mapping in real-time”. In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2320–2327. DOI: 10.1109/ICCV.2011.6126513.

[41] Timo Ojala, Matti Pietikäinen, and David Harwood. “A comparative study of texture measures with classification based on featured distributions”. In: *Pattern Recognition* 29.1 (1996), pp. 51–59. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(95)00067-4. URL: http://www.sciencedirect.com/science/article/pii/0031320395000674.

[42] Paul Ozog et al. “Long-term Mapping Techniques for Ship Hull Inspection and Surveillance Using an Autonomous Underwater Vehicle”. In: *J. Field Robot.* 33.3 (May 2016), pp. 265–289. ISSN: 1556-4959. DOI: 10.1002/rob.21582. URL: https://doi.org/10.1002/rob.21582.

[43] N. Palomeras et al. “AUV homing and docking for remote operations”. In: *Ocean Engineering* 154 (2018), pp. 106–120. ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2018.01.114. URL: http://www.sciencedirect.com/science/article/pii/S0029801818301367.

[44] Edward Rosten and Tom Drummond. “Fusing points and lines for high performance tracking.” In: *IEEE International Conference on Computer Vision*. Vol. 2. Oct. 2005, pp. 1508–1511. DOI: 10.1109/ICCV.2005.104. URL: http://www.edwardrosten.com/work/rosten_2005_tracking.pdf.

[45] Edward Rosten and Tom Drummond. “Machine learning for high-speed corner detection”. In: *European Conference on Computer Vision*. Vol. 1. May 2006, pp. 430–443. DOI: 10.1007/11744023_34. URL: http://www.edwardrosten.com/work/rosten_2006_machine.pdf.

[46] Ethan Rublee et al. “ORB: An Efficient Alternative to SIFT or SURF”. In: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. ISBN: 978-1-4577-1101-5. DOI: 10.1109/ICCV.2011.6126544. URL: http://dx.doi.org/10.1109/ICCV.2011.6126544.

[47] Luan Silveira et al. “An Open-source Bio-inspired Solution to Underwater SLAM”. In: vol. 48. Dec. 2015. DOI: 10.1016/j.ifacol.2015.06.035.

[48] Carlo Tomasi and Takeo Kanade. *Detection and Tracking of Point Features*. Tech. rep. International Journal of Computer Vision, 1991.

[49] George Vogiatzis and Carlos Hernández. “Video-based, real-time multi-view stereo”. In: *Image and Vision Computing* (2012). ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2012.08.001. URL: http://www.sciencedirect.com/science/article/pii/S0262885612001254.

[50] N. Weidner et al. “Underwater cave mapping using stereo vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 5709–5715. DOI: 10.1109/ICRA.2017.7989672.

[51] Y. Zheng et al. “Single-Image Vignetting Correction”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.12 (Dec. 2009), pp. 2243–2256. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2008.263.