Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Simen Gangstad

Bare-Metal Visual-Inertial Odometry

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Kostas Alexis
June 2023

NTNU
Norwegian University of
Science and Technology

Simen Gangstad

Bare-Metal Visual-Inertial Odometry

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Kostas Alexis
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

The utilisation of cameras as guiding sensors for state estimation has become increasingly prevalent during the last two decades with methods such as visual odometry (VO), visual-inertial odometry (VIO) and simultaneous localisation and mapping (SLAM). Existing implementations for these methods are typically computationally demanding and unsuitable for nanoscale drones ($\leq$ 10 cm in diameter and $\leq$ 50 grams in weight). Due to these platforms' reduced computational resources and power budget, a different mindset is required for the design and implementation of these methods. This master's thesis presents a bare-metal implementation for a visual-inertial odometry pipeline running on a microcontroller with a clock speed of 1 GHz. The implementation is constructed under tight memory and runtime requirements, requires 3 MB of RAM and is tested against the EuRoC dataset. It achieves comparable runtime performance and accuracy to state-of-the-art implementations on more performant hardware.

Sammendrag

Bruken av kameraer som veiledende sensorer for tilstandsestimering har blitt stadig mer utbredt de siste to tiårene med metoder som visuell odometri (VO), visuell-inertiell odometri (VIO) og simultan lokalisering og kartlegging (SLAM). Eksisterende implementasjoner for disse metodene er typisk beregningskrevende og uegnet for nanoskala-droner ($\leq$ 10 cm i diameter og $\leq$ 50 gram i vekt). På grunn av disse plattformenes reduserte beregningsressurser og strømbudsjett kreves det en annen tankegang for designet og implementasjonen av disse metodene. Denne masteroppgaven presenterer en "på-metallet"-implementasjon for visuell-inertiell odometri som kjører på en mikrokontroller med en klokkehastighet på 1 GHz. Implementasjonen er konstruert under strenge krav til minne og kjøretid, krever 3 MB hurtigminne og er testet mot EuRoC-datasettet. Implementasjonen oppnår en kjøretid og nøyaktighet som er sammenlignbar med de mest avanserte implementasjonene som kjører på mer ytelsessterk maskinvare.

Preface

This thesis concludes the author’s master’s degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The work presented in this thesis was conducted in the spring of 2023 under the supervision and guidance of Professor Kostas Alexis at the Department of Engineering Cybernetics, NTNU.

The work in this thesis builds upon the author’s unpublished specialisation project [1] during the autumn of 2022. Parts of the material and theory from the specialisation project are included in this thesis to increase the coherency of the work for the reader. They are also included due to the tight coupling between this work and the specialisation project work. These sections and chapters are:

- Section 1.3 with modifications
- Section 2.2
- Chapter 4 with modifications, except section 4.5.6
- Chapter 7 with modifications

The implementation of the work in this thesis is based on the LARVIO project [2, 3] and the OpenVINS project [4], and carried out on NXP Semiconductor’s i.MX RT1170 microcontroller with the accompanying software development kit. The implementation uses Eigen [5] for linear algebra and the Embedded Template Library [6] for containers. Moreover, the KITTI flow 2015 dataset [7, 8] and the EuRoC dataset [9] was used for testing the implementation.

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Kostas Alexis, for his expertise and guidance and for providing me with valuable insights into how to see the numerous problem settings I have encountered in a different light. This has helped me greatly in finding solutions to the various problems I have encountered. He has provided guidance for the things which are closely related to his expertise and helped me get in contact with people who have specialised in other fields. This thesis would not have been completed to the same degree without him.

I would also like to thank my mum and dad for their support throughout my degree and for their reassurance that things will work out in the end. Even though they probably have not fully understood every technical problem I have encountered, they have always given me affirmation and encouragement.

Lastly, I would like to thank the many friends I have made here at NTNU throughout the last five years. I thank them for the many laughs, the many interesting discussions, the activities in the various student organisations here at NTNU and, in general, for making the time spent at NTNU something I will look back at with fondness.

Simen Gangstad
Trondheim, June 2023

Contents

Acronyms

ALU arithmetic logic unit. 71

ANEES average normalised estimated error squared. 106, 123

ASIC application-specific integrated circuit. 2–5, 124

BRIEF binary robust independent elementary features. 7, 24, 33, 34, 89–91, 95, 107, 122, 125

CDF cumulative distribution function. 37

CPU central processing unit. 76, 81, 82

DMA direct memory access. 76

DTCM data tightly-coupled memory. 76, 80, 83, 90, 96–99, 101, 103, 123

EKF extended kalman filter. 51

EuRoC euroc dataset. iii, v, vii, 87, 88, 90, 95, 108, 114, 116, 122, 125

FAST features from accelerated segment test. 6, 7, 24–26, 83, 85, 87, 88, 103, 105, 107, 122

FPGA field programmable gate array. 2–5

FPU floating point unit. 73, 80

GNSS global navigation satellite system. 1, 15

IMU inertial measurement unit. 1, 5–7, 13–15, 51–54, 59, 60, 63, 67–69, 92–94, 96, 97, 105, 107, 115, 123

ITCM instruction tightly-coupled memory. 76, 80, 123

LIDAR light detection and ranging. 15

MCU microcontroller unit. 2–7, 71–78, 80, 87, 108, 123

MPU microprocessing unit. 2–5, 121

MSCKF multi-state constraint kalman filter. 6, 7, 51, 67, 83, 92, 95, 122, 124, 125

OCRAM on-chip RAM. 80, 82, 98, 99, 102, 103, 123

ORB oriented FAST and rotated BRIEF. 24, 25, 33

PCB printed circuit board. 108, 123, 124

PDF probability density function. 37–41

RAM random access memory. 3, 80, 87

RANSAC random sample consensus. 122

SDRAM synchronous dynamic RAM. 80–82, 90, 98

SIFT scale-invariant feature transform. 6, 24, 26, 83

SIMD single instruction multiple data. 7, 73, 74, 80, 83, 85, 87, 125

SLAM simultaneous localisation and mapping. iii, v, 1, 3–5, 14

SURF speeded up robust features. 24, 26, 33

TCM tightly-coupled memory. 74, 76, 80–82

VIO visual-inertial odometry. iii, v, 1, 3–7, 13, 14, 37, 51, 79, 80, 92, 103, 113, 114, 119, 121, 123–125

VO visual odometry. iii, v, 1, 3–6, 14

Notation & Conventions

	Scalars
x, X	Scalars are given by regular type face letters
	Vectors
$\mathbf{x}$	Vectors are given by lower-case bold letters
$\mathbf{x}_b^a$	Vector of element b in frame a
$\mathbf{x}^{aT}$	Vector in frame a transposed
$\mathbf{x}_{[to][from]}$	Translation vector from frame $[from]$ to frame $[to]$
$\tilde{\mathbf{x}}$	Homogeneous vector
$[\mathbf{x}]_\times$	Skew symmetric matrix constructed from $\mathbf{x}$
	Matrices
$\mathbf{A}$	Matrices are given by upper-case bold letters
$\lvert\mathbf{A}\rvert$	Determinant of the matrix $\mathbf{A}$
$\mathbf{A}^{-T}$	Transpose of the inverse of $\mathbf{A}$
$\mathbf{R}_{[to][from]}$	Rotation matrix rotating from frame $[from]$ to frame $[to]$
	Quaternions
$\mathbf{q}$	Quaternions are given by lower-case bold $\mathbf{q}$
$\mathbf{q}^{-1}$	Conjugate of the quaternion $\mathbf{q}$
$\mathbf{q}_{[to][from]}$	Quaternion rotation from frame $[from]$ to frame $[to]$
$\mathbf{R}(\mathbf{q})$	Rotation matrix constructed from the quaternion $\mathbf{q}$
	Probability Theory
x'	Realisation of the scalar stochastic variable x
$\mathbf{x}'$	Realisation of the multi-dimensional stochastic variable $\mathbf{x}$
$\mathbf{x}^-$	A priori estimate of the stochastic variable $\mathbf{x}$
$\hat{\mathbf{x}}$	Expected value of the stochastic variable $\mathbf{x}$

Chapter 1

Introduction

This introductory chapter will present the motivation behind this master's thesis and its problem statement. It will also review the relevant literature in the academic field before listing the contributions this thesis has made.

1.1 Motivation

Autonomous robots have become increasingly prevalent in various industries over the last few decades, from the early developments targeted towards military use and space exploration to automotive, agriculture, and consumer electronics such as drones. Due to the increasing capabilities made possible by more performant computational units, autonomous robots have been able to push the boundaries into more advanced and complex tasks. To perform these tasks, there has been an increasing need for an accurate representation of their state and the environment they operate in, popularly called *state estimation*.

The use of cameras as guiding sensors for state estimation has become more and more relevant in recent years as they have become cheaper, smaller, lighter, and better. The early work, however, dates back to the 1980s [10–12]. A range of highly accurate methods [13–21] has been developed to perform state estimation with visual data for use in environments where other sensors — such as GNSS (global navigation satellite system) receivers — are unsuitable. State estimation with visual data is often referred to as *visual odometry* (VO), where the name relates to how wheel odometry is used for state estimation of rovers. Methods have also been developed to include auxiliary sensors such as inertial measurement units (IMUs) [13–15], referred to as *visual-inertial odometry* (VIO). Moreover, during the last decade, there has been an ongoing push into incorporating maps of the environment into these methods [17–21], where the system uses the map to recognise where it has previously been. This is popularly called SLAM (simultaneous localisation and mapping).

Processing visual input and performing state estimation is inherently computationally expensive, and the popular state-of-the-art implementations require perform-

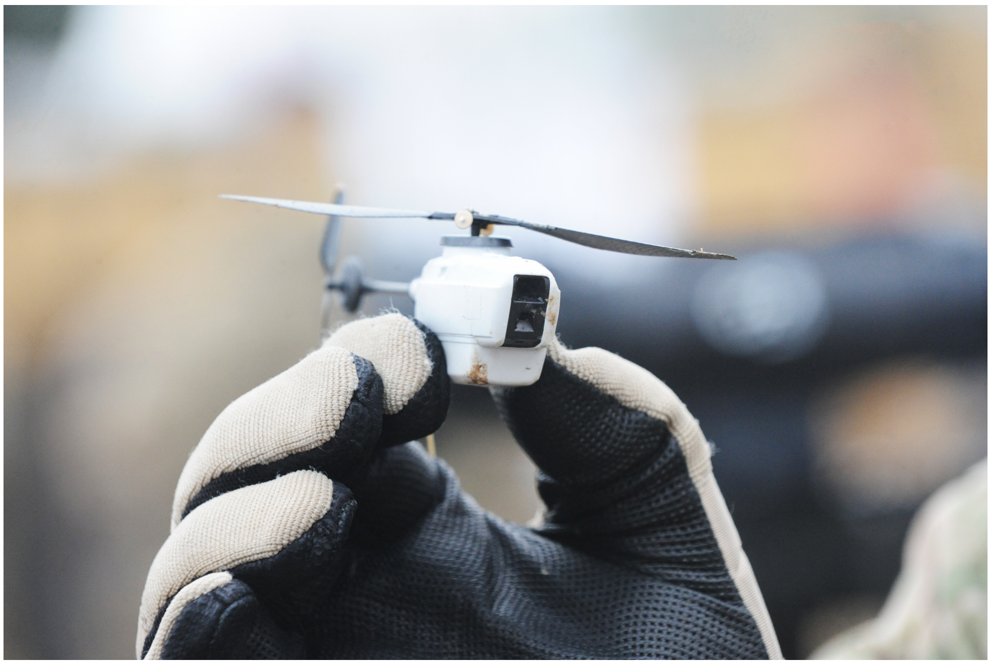


Figure 1.1: Teledyne FLIR's Black Hornet drone [22].

ant hardware. In the search for enabling more intelligent and more autonomous nano-scale robots and drones ($\leq$ 10 cm in diameter and $\leq$ 50 grams in weight), which can perform inspections, aid search and rescue and accompany military operations in tightly confined areas, these implementations are mainly unsuitable due to the smaller weight and power budgets which the required compute units for these implementations do not fit within. This master's thesis seeks to explore an implementation of state estimation with cameras differently, closely related to a statement made by Zhang *et al.* [23]:

> *We argue that scaling down VIO to miniaturised platforms (without sacrificing performance) requires a paradigm shift in the design of perception algorithms, and we advocate a co-design approach in which algorithmic and hardware design choices are tightly coupled.*

1.2 Problem Statement

Enabling state estimation with cameras on miniaturised platforms requires a great concern of exploiting compute units to their fullest extent due to the complexity and computational expensiveness of the algorithms. The compute units suitable on these miniaturised platforms are not necessarily conventional processors, but rather *micro-controller unit*s (MCUs), *field programmable gate array*s (FPGAs) or *application-specific integrated circuit*s (ASICs). Typically, *microprocessing unit*s (MPUs) are utilised within the bigger — but still small-scale — robotic domain, with examples ranging from the NVIDIA Jetson family and Raspberry Pi to Hardkernel Co., Ltd.'s ODROID. These compute platforms, despite their name, are crossing into the domain of conventional processors: running conventional operating systems such as Ubuntu, having multiple cores with clock speeds well above 1 GHz and requiring a significant amount of power compared with what is available in the domain of miniaturised robotics. Furthermore, these devices are often used due to the implementations for the popular state-of-the-art methods for VO, VIO and SLAM — as well as other software utilised in the domain of robotics — have been designed to run on conventional operating systems.

FPGAs are programmable logic circuits, enabling the creation of application-specific hardware for a given task and are often used for prototyping new hardware solutions. They are also used for simulation or as accelerators due to their inherent parallel processing capabilities. Whereas FPGAs can be re-programmed, ASICs cannot. Application-specific integrated circuits — as given by their name — are designed with a specific use case in mind at the hardware level and are often used as accelerators for a greater system of compute units. ASICs can often result from a prototype initially developed with an FPGA. The size, amount of power and computational power of FPGAs and ASICs are thus dependent on the application. Still, being inherently programmed or designed for a specific application, they can achieve great performance at smaller power consumption levels, with the loss of not being general purpose and often having to be accompanied by other compute units.

Microcontroller units share many similarities with MPUs, being general-purpose compute units programmed in software rather than hardware. However, they reside in a different domain than MPUs regarding hardware complexity, software complexity, power consumption, size and price. MCUs are often used in applications requiring a general-purpose, low-power, real-time and cheap control unit, with applications ranging from the automotive industry and the space industry to home appliances. Unlike MPUs, MCUs do not run conventional operating systems. They are programmed directly, popularly called *bare-metal*. This firstly makes it easier for MCUs to abide by real-time constraints — which is crucial in control applications — due to not having the concern of a scheduler kicking in irregularly. Secondly, this makes it easier to take greater advantage of the hardware than what is often possible on MPUs, allowing for greater control over how memory is used and where data should be placed in memory. MCUs are generally less powerful, smaller and require less power. Their clock speed typically ranges from tens of MHz to 1 GHz for the most performant MCUs, with RAM ranging between a few kilobytes to a few megabytes.

This master's thesis explores what can be made possible in the domain of state estimation with cameras when the implementation can take greater advantage of the hardware it runs on. It explores the use of MCUs for *monocular* VIO based on the following arguments:

- MCUs make it easier to take full advantage of the hardware compared with MPUs and use less power.
- MCUs are more general-purpose than FPGAs and ASICs, allowing for easily including other aspects such as navigation and control, as well as being able to draw inspiration from the vast amount of existing software within the field of robotics.
- VIO is generally more accurate and robust than VO and less computationally expensive than SLAM.

Thus, the problem statement this master's thesis seeks to explore is how to enable VIO on miniaturised platforms, using MCUs through tightly-coupled software-hardware considerations in the implementation, where the use case is smaller and more autonomous drones.

1.3 Related Work

This section will outline the related work to the problem statement of this thesis, focusing mainly on the MCU, ASIC and FPGA domains. The ideas from the ASIC and FPGA domains are highly applicable despite the vastly different hardware platforms. This section will also compare state-of-the-art VIO methods.

1.3.1 MCU Solutions

He *et al.* [24] proposed a VO method based on an RGB-D (colour and depth) camera, where an STM32F767ZI with a 216 MHz Cortex-M7 and 512 KB of RAM is used. The proposed visual odometry solution runs at 33 FPS with an image resolution of 320 × 240. It utilises a modified edge-based detection algorithm, lookup tables, sparse-to-dense tracking scheme instead of image pyramids and a thoroughly minimised data representation within the non-linear solver to meet the MCU's hard compute and memory requirements. Even with these optimisations and approximations, the algorithm has comparable performance with other VO algorithms, such as ORB-SLAM [19] (without loop closure enabled), at the cost of just 355 KB of RAM usage.

Moreover, in Decroon *et al.*'s [25] work, they proposed a flapping wing robot which utilises an onboard stereo vision system based on the STM32F405 microcontroller. The microcontroller has a 168 MHz ARM Cortex-M4 core and 192 KB of RAM. The vision system is used for obstacle avoidance on the robot to achieve collision-free flight. The proposed method utilises 128 × 96 images and can process them at $\sim$ 11 Hz. That said, the vision system is not used for state estimation, it is used as a strategy in blind flight: The robot does not know where it is but can avoid obstacles.

Somewhat related is also the *Fünfiiber nano-drone* project [26], where Müller *et al.* proposed an open-source nano-drone platform based on the PULP platform. However, the work is a proposition for a general-purpose computing platform for nano-drones, not necessarily a specific solution for VIO, VO or SLAM. The prevalent use of MCUs in other academic works is mainly as auxiliary compute units for control, requiring an extra conventional processor for state estimation [27, 28].

Contrary to the work by He *et al.*, the work in this thesis is based on visual-inertial odometry, using a colour-based camera and an IMU. This thesis also separates itself from the work by Decroon *et al.* as a system targeted towards state estimation rather than obstacle avoidance.

1.3.2 ASIC and FPGA Solutions

Within the ASIC and FPGA domain, there is a wide range of academic work [29–33]. The work by Suleiman *et al.* [29] on the *Navion* chip stands out as one of the most promising ASIC solutions for VIO. They heavily utilise the sparsity of the estimation problem, lossy compression of the image data, pre-integration of the IMU measurements between keyframes and tightly packing of feature track data to reduce computational cost and memory usage, achieving a VIO pipeline consuming 854 KB of RAM with a trajectory error of 0.28%. The chip consumes, on average, 2 mW whilst running the VIO pipeline on 752×480 stereo images from the EuRoC dataset [9] at 20 FPS.

1.3.3 Industrial Solutions

In the industry, there are products such as Teledyne FLIR's Black Hornet, which has capabilities for vision-based navigation in GNSS-denied environments, e.g. indoors [34]. The hardware utilised on the Black Hornet is not public information, but it serves as an example of an equivalent platform for which this master's thesis is situated around.

1.3.4 Comparison of State-Of-The-Art Methods

Delmerico and Scaramuzza [35] outlined a benchmark comparison study on the popular state-of-the-art methods for monocular VIO. The methods were compared on the following platforms:

- A conventional consumer laptop with a quad-core 2.8 GHz Intel Core i7 processor.
- The Intel NUC, a small form factor desktop computer with a dual-core 3.1 GHz Intel Core i7 processor.
- The Up Board, a single-board computer with a quad-core Intel Atom processor operating at 1.44 GHz.
- The ODROID XU4, an embedded system-on-chip with a quad-core 1.3 GHz ARM Cortex-A7 MPU and a quad-core 1.9 GHz ARM Cortex-A15 MPU.

Delmerico and Scaramuzza found that the state-of-the-art methods that stood out in low computational cost were SVO+MSF [16, 36], SVO+GTSAM [16, 37, 38], ROVIO [14], OKVIS [15] and MSCKF [13].

SVO (fast Semi-direct monocular Visual Odometry) [16] is one of the fastest monocular VO methods in the field. However, due to the inherent unobservability of the scale of a scene in monocular VO [39], it requires a way to make the scale observable. This is similar to how humans and animals with front-facing eyes perceive depth better than animals with side-facing eyes. To alleviate this in the comparison study, the method was bundled with the use of an IMU (which can render the scale observable given sufficient excitation of the system) with two frameworks for sensor fusion: the Multi-Sensor Fusion framework [36] and GTSAM [38]. These two variants demonstrate the two popular ways of doing sensor fusion: filter-based or factor graph-based. SVO+GTSAM was shown to be the most accurate method among those mentioned. However, SVO+MSF and SVO+GTSAM require a lot of memory, one of the scarcest resources on MCUs, rendering them less feasible for such a platform.

ROVIO (Robust Visual-Inertial Odometry) [14] is a robo-centric method for VIO with a filter-based sensor fusion framework. Compared with the other methods mentioned, it is accurate for higher-end platforms but requires high clock speeds.

OKVIS (Open Keyframe-based Visual-Inertial SLAM) [15] is a non-linear optimisation-based method that can operate on both stereo and monocular cameras. It is generally more accurate than ROVIO but has a much higher required processing time per camera frame.

MSCKF (Multi-State Constraint Kalman Filter) [13] is a filter-based method for VIO, where the geometric constraints occurring from observing a point in a scene from multiple different viewpoints are exploited to estimate the position and orientation of the system. MSCKF's computational complexity is linear in the number of features observed and has a low memory footprint whilst requiring a small amount of processing time per frame, making it one of the more suitable methods to explore on an MCU platform. That said, it is not one of the most accurate methods.

1.4 Contributions

This master's thesis will not outline a new VIO method. It will draw inspiration from the LARVIO project [2, 3] and the OpenVINS project [4]. Both these projects are MSCKF implementations and utilise a computationally less expensive *frontend* (the component processing the visual input). The frontends are based on FAST [40, 41] and Lucas-Kanade [42] for feature extraction and tracking instead of SIFT [43], which is the method utilised in the original paper by Mourikis and Roumeliotis [13].

That said, even though this master's thesis draws inspiration from these implementations, their ideas are heavily adapted by the constraints imposed by a hardware platform with less computational power and resources, with various modifications and optimisations influenced by these constraints. This master's thesis's primary concern is tightly coupled software-hardware *implementation* details, which the reader should keep in mind throughout the rest of this thesis.

The work in this thesis builds upon the work of the author's specialisation project [1], where a frontend for a VIO pipeline was constructed based on the FAST feature extractor and the pyramidal Lucas-Kanade feature tracker. The implementation was tested on the NXP i.MX RT1160 MCU using the EuRoC dataset [9]. Thus, this thesis will mainly focus on the development of the *backend* (which fuses the IMU data with the camera data). The main contributions of the preceding specialisation project were:

- A SIMD accelerated FAST feature extractor implementation (section 8.3.1).
- A memory-conservative pyramidal Lucas-Kanade feature tracker implementation which does not require two full image pyramids (section 8.3.2).

The main contributions of this thesis are:

- A highly memory-conservative BRIEF descriptor implementation where a separate copy of the image is not required (section 8.3.3).
- A multi-region allocator with prioritisation for fast memory access (section 8.4.2).
- A tightly coupled software-hardware optimised MSCKF-based backend (section 8.4).

The code associated with the specialisation project and this master's thesis can be found on GitHub [44].

1.5 Outline

The rest of this thesis is outlined as the following. Chapters 2-7 summarises the relevant background theory. Chapter 8 outlines the choice of the hardware platform, a review of the frontend components made in the preceding specialisation project among adaptions and additions and the implementation details of the backend. It also outlines the runtime and memory optimisations made in the backend.

Chapter 9-11 presents how the pipeline was tested against the KITTI flow 2015 dataset [7, 8] and the EuRoC [9] dataset, accompanied by the results and a discussion. Chapter 12 presents the conclusion and further work.

Chapter 2

Rigid Transformations

This chapter will briefly outline the theory behind 3D rotations and 3D rigid transformations, using the special orthogonal group $SO(3)$, the 3-sphere group S^3 and the special Euclidean group $SE(3)$.

2.1 Rotation Matrices

3×3 rotation matrices represent orientation around the origin of the three-dimensional Euclidean space $\mathbb{R}^3$ and reside under the special orthogonal group of dimension 3: $SO(3)$. Rotation matrices are unitary, yielding that the vectors forming the basis of the matrix are orthogonal. This gives the rotation matrices the following properties:

$$|\mathbf{R}| = \pm 1 \qquad\qquad \mathbf{R}^{-1} = \mathbf{R}^T \tag{2.1}$$

With rotation matrices, a vector $\mathbf{x}^b$ in frame b can be rotated to frame a by means of matrix multiplication:

$$\mathbf{x}^a = \mathbf{R}_{ab}\mathbf{x}^b \tag{2.2}$$

where the subscript for the rotation matrix is read right-to-left: from frame b to frame a. Composition of rotations with rotation matrices is given by multiplying the matrices together:

$$\mathbf{R}_{ac} = \mathbf{R}_{ab}\mathbf{R}_{bc} \tag{2.3}$$

2.2 Quaternions

Quaternions are four-dimensional components residing in the S^3 group. They are denoted by $\mathbf{q} = \begin{bmatrix}\eta & \boldsymbol{\epsilon}^T\end{bmatrix}^T = \begin{bmatrix}w & x & y & z\end{bmatrix}^T$, where $\eta \in \mathbb{R}$ is the scalar part and $\boldsymbol{\epsilon} \in \mathbb{R}^3$ is the imaginary part. The full theory behind them is out of scope for this work (a more comprehensive guide can be found in the work by Solà [45]), but the following will underline the needed components for this thesis. Note that this thesis uses the *Hamilton* convention, where the quaternions have right-handedness and represent local to global rotation. Moreover, it should be noted that when quaternions are used for spatial rotation, they are assumed to be unit quaternions with a norm of one.

2.2.1 Multiplication

Multiplication of two quaternions is represented with the $\otimes$ operator, and given by:

$$\mathbf{q}_1 \otimes \mathbf{q}_2 = \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\epsilon}_1^T\boldsymbol{\epsilon}_2 \\ \eta_2\boldsymbol{\epsilon}_1 + \eta_1\boldsymbol{\epsilon}_2 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 \end{bmatrix} \tag{2.4}$$

As with rotation matrices, the product $\mathbf{q}_{ac} = \mathbf{q}_{ab} \otimes \mathbf{q}_{bc}$ represents the composition of rotating from frame c to frame b with the rotation from the frame b to the frame a. Quaternion multiplication is not commutative: $\mathbf{q}_1 \otimes \mathbf{q}_2 \neq \mathbf{q}_2 \otimes \mathbf{q}_1$. Quaternion multiplication can also be expressed as two equivalent matrix products:

$$\mathbf{q}_1 \otimes \mathbf{q}_2 = [\mathbf{q}_1]_L\mathbf{q}_2 \qquad \mathbf{q}_1 \otimes \mathbf{q}_2 = [\mathbf{q}_2]_R\mathbf{q}_1 \tag{2.5}$$

where:

$$[\mathbf{q}]_L = \begin{bmatrix} w & -x & -y & -z \\ x & w & -z & y \\ y & z & w & -x \\ z & -y & x & w \end{bmatrix} \qquad [\mathbf{q}]_R = \begin{bmatrix} w & -x & -y & -z \\ x & w & z & -y \\ y & -z & w & x \\ z & y & -x & w \end{bmatrix} \tag{2.6}$$

If the real part of the quaternion is zero, this result becomes:

$$[\mathbf{q}]_L = \begin{bmatrix} 0 & -\boldsymbol{\epsilon}^T \\ \boldsymbol{\epsilon} & [\boldsymbol{\epsilon}]_\times \end{bmatrix} \qquad [\mathbf{q}]_R = \begin{bmatrix} 0 & -\boldsymbol{\epsilon}^T \\ \boldsymbol{\epsilon} & -[\boldsymbol{\epsilon}]_\times \end{bmatrix} \tag{2.7}$$

2.2.2 Rotation Using Quaternions

Given a vector $\mathbf{x}^b$ in frame b, the rotation bringing the vector to frame a is given by the following quaternion product:

$$\mathbf{x}^a = \mathbf{q}_{ab} \otimes \mathbf{x}^b \otimes \mathbf{q}_{ab}^{-1} = \mathbf{R}_{ab}\mathbf{x}^b \tag{2.8}$$

where a slight abuse of notation is used: implicitly assuming that when $\mathbf{x}^b$ is used in the quaternion product, it represents the four-dimensional vector given by $\begin{bmatrix} 0 & \mathbf{x}^{bT} \end{bmatrix}^T$.

2.2.3 The Derivative of a Quaternion

For a given axis-angle representation with the rotation angle $\frac{\alpha}{2}$ and axis $\boldsymbol{\lambda}$, the quaternion is defined as:

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ \boldsymbol{\lambda}\sin\frac{\alpha}{2} \end{bmatrix} \tag{2.9}$$

which for small angles can be approximated as (where $\delta\boldsymbol{\theta} := \boldsymbol{\lambda}\alpha$):

$$\mathbf{q} = \begin{bmatrix} 1 \\ \boldsymbol{\lambda}\frac{\alpha}{2} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix} \tag{2.10}$$

For an infinitesimal duration, one can treat rotation as additive:

$$\boldsymbol{\omega} = \lim_{\delta t \to 0} \frac{\delta\boldsymbol{\theta}}{\delta t} \tag{2.11}$$

This is utilised to express the derivative of the quaternion:

$$\begin{aligned}
\dot{\mathbf{q}} &= \lim_{\delta t \to 0} \frac{\mathbf{q}(t+\delta t) - \mathbf{q}(t)}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{\mathbf{q} \otimes \delta\mathbf{q} - \mathbf{q}}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{\mathbf{q} \otimes \left(\begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\theta}}{2} \end{bmatrix} - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \right)}{\delta t} \\
&= \lim_{\delta t \to 0} \frac{\mathbf{q} \otimes \begin{bmatrix} 0 \\ \frac{\delta\boldsymbol{\theta}}{2} \end{bmatrix}}{\delta t} \\
&= \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}
\end{aligned} \tag{2.12}$$

where the third step follows from the fact that $-\mathbf{q} = \mathbf{q} \otimes \begin{bmatrix} -1 \\ \mathbf{0} \end{bmatrix}$. The derivative can be represented as a matrix product by utilising the right matrix product from equation (2.7):

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) := \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}_R = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}]_x \end{bmatrix} \tag{2.13}$$

which yields:

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \tag{2.14}$$

2.3 Homogeneous Transformations

3D homogeneous transformations are combined 3D rotations and 3D translations residing in the special Euclidean group of dimension 3: $SE(3)$. Homogeneous transformations give an elegant way to represent both the orientation and the position (the *pose*) of an object in 3D space or the rotational and translational transformation between frames in 3D space. This is done by expressing every point in the 3D Euclidean space by a four-dimensional homogeneous vector:

$$\tilde{\mathbf{x}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \in \mathbb{R}^4 \tag{2.15}$$

The transformation is then defined by the 4×4 homogeneous transformation matrix:

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \mathbf{t}_{ba}^{a} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.16}$$

where the notation for the translation should be read as the translation vector from frame a to frame b, expressed in frame a. This allows for transforming a vector in frame b to frame a by matrix multiplication:

$$\tilde{\mathbf{x}}^a = \mathbf{T}_{ab}\tilde{\mathbf{x}}^b = \mathbf{R}_{ab}\mathbf{x}^b + \mathbf{t}_{ba}^{a} \tag{2.17}$$

Chapter 3

Visual-Inertial Odometry

Visual-inertial odometry concerns the problem of estimating the ego-motion of an agent based on acceleration and angular velocity measurement from an IMU fused with image data from a camera. In a VIO pipeline, a distinction is made between the *frontend* and the *backend*. The frontend processes the image, whereas the backend processes the information extracted by the frontend and fuses the information with the measurements from the IMU.

3.1 Frontend

A frontend can either work *indirectly* or *directly* on the image data. An indirect frontend uses the pixel intensities to build *descriptors* of certain areas in the image, here called *features*. These descriptors are used to match features across multiple images. A direct frontend also extracts features from certain areas of interest but does not build descriptors. The feature tracking/matching is done by means of optical flow, where the corresponding matched feature in another image is found by searching in a local area around where the feature was in the original image. The distinction can therefore be summarised as the following: an indirect frontend builds descriptors of the image data to match features, whereas a direct frontend directly matches features by using the pixel intensities.

3.2 Backend

The backend in a VIO pipeline can mainly take two forms: filter-based and factor graph-based. Arguably, the main difference between filter-based backends and factor graph ones is the horizon they operate on. Filter-based backends only relate the next estimate to the previous estimate. In contrast, a factor graph keeps a range of estimates in its graph, which all can contribute to refining the estimate of the current state and correcting previous states. This can result in filter-based methods being less robust than factor graph-based backends at the benefit of being less computationally demanding.

Visual-inertial odometry methods are bound to drift since there is no long-term data association for features. SLAM alleviates this by keeping a long-term map of the features seen in particular *keyframes*. Thus, when the pipeline recognises that it is currently at a location where it has been before, it can correct the drift accumulated in its trajectory, as seen in figure 3.1. SLAM is, however, relatively computationally demanding.

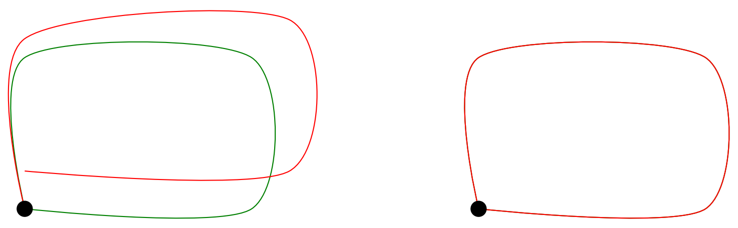


Figure 3.1: Demonstration of loop-closure in SLAM, where the agent finds itself in a location it has observed before and can correct the drift in its estimates. The red line is the estimate and the green line is the actual trajectory.

3.3 Scale Ambiguity

In a visual odometry method, where no IMU is utilised, the scale of the scene is not observable [39]. Thus, the trajectory can only be correct up to a scale factor. Certain methods require the scale to be initialised during initialisation. This can be problematic if the scale is not observable throughout the rest of the estimation pipeline, as drift in the scale can occur. Visual-inertial odometry alleviates this by utilising an IMU. With sufficient excitation of the system, the scale can be rendered observable due to having a scale-correct estimate of how far the agent has travelled between camera frames.

Chapter 4

Computer Vision Fundamentals

Computer vision is the field of utilising digital cameras to perceive and draw information out from the surrounding world. As image sensors have become cheaper, smaller and better throughout the last few decades, research in the use of computer vision has increased drastically. The use of digital cameras is heavily motivated by the vast amount of information they provide compared to other sensors such as GNSS receivers, IMUs and simpler LIDARs, giving them a tremendous amount of useful applications. This section will outline some of the fundamentals in computer vision, mainly the idealised lens model, how geometric constraints can be imposed between different camera frames, distortion, extracting features and tracking features over a range of images.

4.1 The Pinhole Camera Model

The pinhole camera model resembles the early works towards photography: the *camera obscura*. A pinhole (here denoted by an *aperture*) lets light in from a scene which is projected to an *image plane*, as seen in figure 4.1. The length from the aperture to the image plane is the *focal length*, f. This section will outline the basics of the pinhole camera model. For a more comprehensive examination, the work by Szeliski [46] is highly relevant.

For convenience, a *virtual image plane* is used to represent the image in front of the aperture. This construct is however purely a mathematical representation. As the image in the image plane will be flipped upside down and inverted, the virtual image plane alleviates this by representing the image in front of the aperture at the focal length distance. A given world 3D coordinate $\mathbf{x} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ thus maps to the virtual image plane camera coordinate $\mathbf{x}^* = \begin{bmatrix} x & y \end{bmatrix}^T$ by using similar triangles:

$$x = f\frac{X}{Z} \qquad y = f\frac{Y}{Z} \tag{4.1}$$

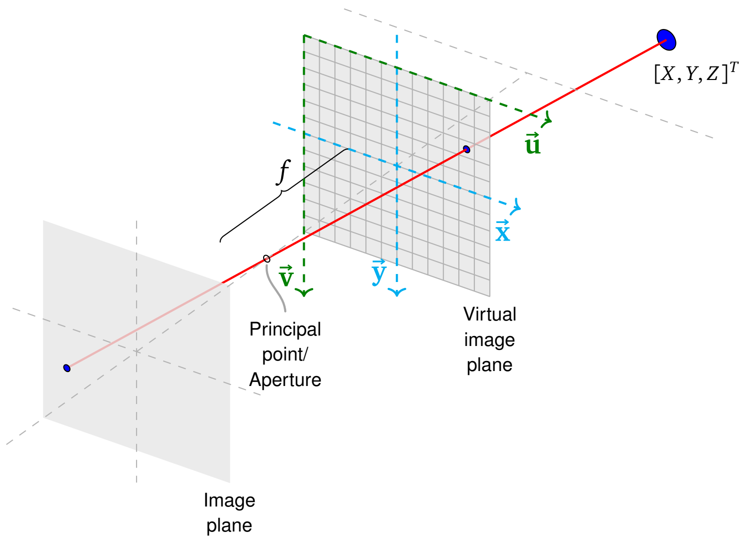


Figure 4.1: The pinhole camera model, where a virtual image plane is used to represent the scene in front of the aperture. Here, $\left[X,Y,Z\right]^T$ is a point in 3D space. $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ form the basis for the 2D space (in meters) where the projected point appears on the virtual image plane. $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ form the basis for the same plane represented in pixels.

$\mathbf{x}^*$ is given in meters, which can be transformed to pixel coordinates by means of scaling and translating:

$$u = s_x x + c_x \qquad\qquad v = s_y y + c_y \tag{4.2}$$

Here (s_x, s_y) is the pixel width and height density per meter and (c_x, c_y) is the principal point offset. The principal point offset is determined by the principal axis, the line which is perpendicular to the image plane and passes through the aperture. The principal point offset is thus given by the intersection point of the principal axis and the image plane. Representing the pixel coordinates as homogeneous coordinates, $\tilde{\mathbf{u}} = \begin{bmatrix}\tilde{u} & \tilde{v} & \tilde{w}\end{bmatrix}^T$, the projection is given by:

$$\tilde{\mathbf{u}} = \mathbf{K}\mathbf{x} = \begin{bmatrix} f s_x & 0 & c_x \\ 0 & f s_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4.3}$$

The inhomogeneous representation is thus given by $(u,v) = (\frac{\tilde{u}}{\tilde{w}}, \frac{\tilde{v}}{\tilde{w}})$. $\mathbf{K}$ is popularly called the camera's *intrinsic matrix*. Furthermore, for convenience with $SE(3)$ operations where homogeneous 3D coordinates are utilised, a projection matrix $\mathbf{P}$ is needed:

$$\tilde{\mathbf{u}} = \mathbf{KP}\tilde{\mathbf{x}} = \begin{bmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.4}$$

Further in the theory, the notation $(u, v) = \pi(\tilde{\mathbf{x}})$ will be utilised as a shorthand for the projection mapping homogeneous 3D coordinates to pixel coordinates, given by equation (4.4) and de-homogenisation. The notation $(u, v) = \pi(\mathbf{x})$ will likewise be utilised if $\mathbf{x}$ is not homogeneous, given by equation (4.3) and de-homogenisation.

4.2 Epipolar Geometry

Epipolar geometry describes the geometric relationship between two perspective cameras or between two images taken with a perspective camera at different viewpoints. As seen in figure 4.2, the epipolar geometry defines the epipolar plane: the plane spanned out from the origin of the left camera frame, right camera frame and an arbitrary observed 3D point which is seen from both cameras.

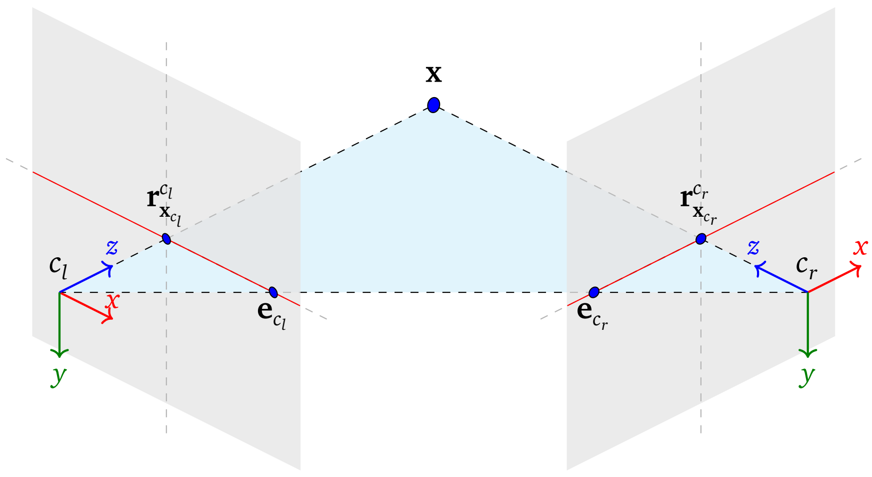


Figure 4.2: Two camera frames observing the same point in a 3D scene, where the plane formed by c_l, $\mathbf{x}$ and c_r is the epipolar plane.

The *baseline* is given by the line joining the two camera frames c_l and c_r. Furthermore, the epipolar lines are where the epipolar plane intersects the image planes, here denoted in red. The epipoles are where the baseline intersects the image planes, here denoted $\mathbf{e}_{c_l}$ and $\mathbf{e}_{c_r}$.

Observing the same 3D point from two different camera frames yields geometric constraints on the transformation from c_r to c_l. These constraints are summarised in a 3×3 matrix called the *essential matrix*. Let the transformation from frame c_r to frame c_l be given by the following:

$$\mathbf{T}_{c_l c_r} = \begin{bmatrix} \mathbf{R}_{c_l c_r} & \mathbf{t}^{c_l}_{c_r c_l} \\ \mathbf{0} & 1 \end{bmatrix} \in SE(3). \tag{4.5}$$

where $\mathbf{t}^{c_l}_{c_r c_l}$ is the vector describing the baseline going from c_l to c_r and $\mathbf{R}_{c_l c_r}$ is the rotation from frame c_r to frame c_l. The normal vector of the epipolar plane can then be described by the cross product of the homogeneous coordinate of the ray pointing towards $\mathbf{x}$ in frame c_l and the baseline vector: $\tilde{\mathbf{r}}^{c_l}_{\mathbf{x}_{c_l}} \times \mathbf{t}^{c_l}_{c_r c_l}$. Furthermore, as $\tilde{\mathbf{r}}^{c_r}_{\mathbf{x}_{c_r}}$ lies in the epipolar plane, it must be perpendicular to the normal vector when expressed in frame c_l:

$$(\tilde{\mathbf{r}}^{c_l}_{\mathbf{x}_{c_l}} \times \mathbf{t}^{c_l}_{c_r c_l}) \cdot (\mathbf{R}_{c_l c_r} \tilde{\mathbf{r}}^{c_r}_{\mathbf{x}_{c_r}}) = 0 \tag{4.6}$$

Utilising that $\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = -\mathbf{a}^T [\mathbf{b}]_\times$, this result becomes:

$$(\tilde{\mathbf{r}}^{c_l}_{\mathbf{x}_{c_l}})^T [\mathbf{t}^{c_l}_{c_r c_l}]_\times \mathbf{R}_{c_l c_r} \tilde{\mathbf{r}}^{c_r}_{\mathbf{x}_{c_r}} = 0 \tag{4.7}$$

where the essential matrix is defined as $\mathbf{E}_{c_l c_r} := [\mathbf{t}^{c_l}_{c_r c_l}]_\times \mathbf{R}_{c_l c_r}$, and thus encodes the transformation between the frames. The inverse representation is given by the transpose: $\mathbf{E}_{c_r c_l} = \mathbf{E}^T_{c_l c_r}$. When utilising pixel coordinates, the *fundamental* matrix can be used to describe the relationship between the frames:

$$(\tilde{\mathbf{u}}^{c_l}_{\mathbf{x}_{c_l}})^T \mathbf{F}_{c_l c_r} \tilde{\mathbf{u}}^{c_r}_{\mathbf{x}_{c_r}} \tag{4.8}$$

where $\mathbf{F}_{c_l c_r} := \mathbf{K}^{-T}_{c_l} \mathbf{E}_{c_l c_r} \mathbf{K}^{-1}_{c_r}$, $\mathbf{K}_{c_l}$ is the left camera's intrinsic matrix and $\mathbf{K}_{c_r}$ is the right camera's intrinsic matrix.

Given known intrinsic matrices and point correspondences between two frames, the essential matrix can be computed from e.g. the 8-point algorithm [47, 48]. If the problem is over-determined, the essential matrix can be found by least-squares optimisation and singular-value decomposition. Furthermore, singular-value decomposition can also be used to extract the transformation the essential matrix encodes. This will yield 4 solutions for the transformation. The correct solution can be found by transforming the points into the scene and finding the solution which results in the points being in front of both camera frames.

Once the essential matrix is found, it can be useful for finding new correspondences. Given a feature represented by a homogeneous pixel coordinate $\tilde{\mathbf{u}}^{c_r}_{\mathbf{x}_{c_r}}$, the homogeneous representation of the epipolar line in the left image is given by: $\tilde{\mathbf{l}}^{c_l} = \mathbf{F}_{c_l c_r} \tilde{\mathbf{u}}^{c_r}_{\mathbf{x}_{c_r}}$. The corresponding pixel in the left frame must then lie along this line, represented by:

$$\tilde{\mathbf{l}} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \Rightarrow \qquad ax + by + c = 0 \tag{4.9}$$

4.3 Triangulation

If a given feature is observed in multiple images with sufficient motion between where the images were captured, triangulation can be used to find an estimate for the 3D position of the feature. Let the given 3D position in the camera frame c_k be denoted by $\mathbf{p}^{c_k}$. The relation to the 3D position of the feature in an *anchor* frame c_a is then given by:

$$\mathbf{p}^{c_k} = \mathbf{R}(\mathbf{q}_{c_k c_a})\mathbf{p}^{c_a} + \mathbf{p}^{c_k}_{c_k c_a} \tag{4.10}$$

where $\mathbf{q}_{c_k c_a}$ and $\mathbf{p}^{c_k}_{c_k c_a}$ denotes the known or estimated orientation and translation from frame c_a to frame c_k, respectively.

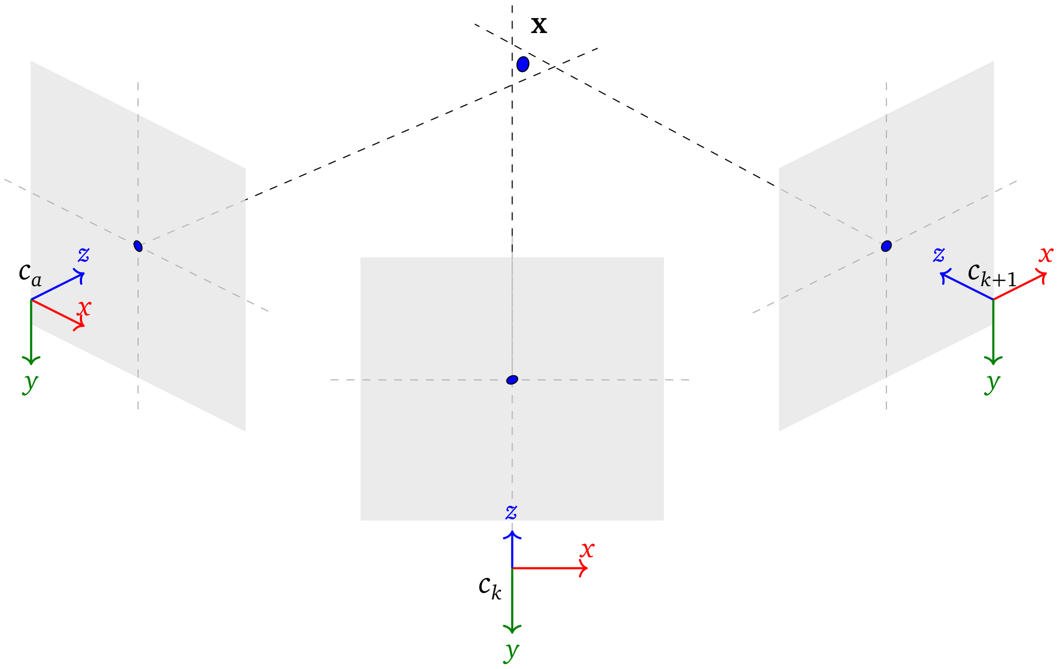


Figure 4.3: Three camera frames observing the same feature, where there are slight imperfections due to uncertainty in the camera model. The goal of the triangulation is to minimise the area spanned by the observation rays, to find a good estimate for the point $\mathbf{x}$.

Equation (4.10) can then be written as the following:

$$\begin{aligned}
\mathbf{p}^{c_k} &= Z^{c_a}\left(\mathbf{R}(\mathbf{q}_{c_k c_a})\begin{bmatrix}\frac{X^{c_a}}{Z^{c_a}}\\ \frac{Y^{c_a}}{Z^{c_a}}\\ 1\end{bmatrix} + \frac{1}{Z^{c_a}}\mathbf{p}^{c_k}_{c_k c_a}\right)\\
&= Z^{c_a}\left(\mathbf{R}(\mathbf{q}_{c_k c_a})\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix} + \rho\mathbf{p}^{c_k}_{c_k c_a}\right)\\
&= Z^{c_a}\begin{bmatrix}h_{k1}(\alpha,\beta,\rho)\\ h_{k2}(\alpha,\beta,\rho)\\ h_{k3}(\alpha,\beta,\rho)\end{bmatrix}
\end{aligned} \tag{4.11}$$

where $\alpha = \frac{X^{c_a}}{Z^{c_a}}$, $\beta = \frac{Y^{c_a}}{Z^{c_a}}$ and $\rho = \frac{1}{Z^{c_a}}$. This leads to the following measurement relation for the 3D position of feature $\mathbf{p}^{c_k}$ when de-homogenising:

$$\begin{aligned}
\mathbf{z}^*_k &= \begin{bmatrix}\frac{\cancel{Z^{c_a}}h_{k1}(\alpha,\beta,\rho)}{\cancel{Z^{c_a}}h_{k3}(\alpha,\beta,\rho)}\\ \frac{\cancel{Z^{c_a}}h_{k2}(\alpha,\beta,\rho)}{\cancel{Z^{c_a}}h_{k3}(\alpha,\beta,\rho)}\end{bmatrix}\\
&= \begin{bmatrix}\frac{h_{k1}(\alpha,\beta,\rho)}{h_{k3}(\alpha,\beta,\rho)}\\ \frac{h_{k2}(\alpha,\beta,\rho)}{h_{k3}(\alpha,\beta,\rho)}\end{bmatrix}
\end{aligned} \tag{4.12}$$

By then minimising the squared norm of the error between the observed coordinate and the triangulated coordinate, $\mathbf{p}^{c_a}$ can be found:

$$\mathbf{p}^{c_a} = \underset{\alpha,\beta,\rho}{\operatorname{argmin}}(||\mathbf{z}_k - \mathbf{z}^*_k||^2_2) \tag{4.13}$$

To minimise the squared norm, Levenberg-Marquardt [49] can be utilised. Other methods also exist, such as Gauss-Newton. That said, Levenberg-Marquardt can be more robust and offer better convergence as it works as a blending function between standard gradient descent and the Gauss-Newton method [50]. For better convergence, an initial guess for the triangulated point has to be made, which can be found as the solution to a least-squares problem with the observation in the anchor frame and the observation in the camera frame c_k. During the minimisation, the jacobian of the residual $\mathbf{r}_k = \mathbf{z}_k - \mathbf{z}^*_k$ with respect to α, β and ρ is used to find the steepest descent. It is given by:

$$\frac{\partial\mathbf{r}_k}{\partial[\rho,\beta,\rho]} = \frac{\partial\mathbf{r}_k}{\partial\mathbf{h}_k(\rho,\beta,\rho)}\frac{\partial\mathbf{h}_k(\rho,\beta,\rho)}{\partial[\rho,\beta,\rho]} \tag{4.14}$$

where:

$$\frac{\partial \mathbf{r}_k}{\partial \mathbf{h}_k(\rho,\beta,\rho)} = -\begin{bmatrix} \frac{1}{h_{k3}(\alpha,\beta,\rho)} & 0 & -\frac{h_{k1}(\alpha,\beta,\rho)}{h_{k3}(\alpha,\beta,\rho)^2} \\ 0 & \frac{1}{h_{k3}(\alpha,\beta,\rho)} & -\frac{h_{k2}(\alpha,\beta,\rho)}{h_{k3}(\alpha,\beta,\rho)^2} \end{bmatrix} \tag{4.15}$$

$$\frac{\partial \mathbf{h}_k(\rho,\beta,\rho)}{\partial[\rho,\beta,\rho]} = \mathbf{R}(\mathbf{q}_{c_k c_a}) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{p}^{c_k}_{c_k c_a} \end{bmatrix} \tag{4.16}$$

The minimisation can then be done as a combined problem where N observations are related to the anchor frame, imposing a range of constraints on the features in the anchor frame, as shown in equation (4.17) and illustrated in figure 4.3.

$$\mathbf{r} = \begin{bmatrix} \mathbf{r}_k \\ \vdots \\ \mathbf{r}_{k+N-1} \end{bmatrix} \tag{4.17}$$

Finally, when a triangulation for the feature is found in the anchor frame, it can be transformed into the world frame using the known or estimated transformation from the anchor frame to the world frame.

4.4 Distortion

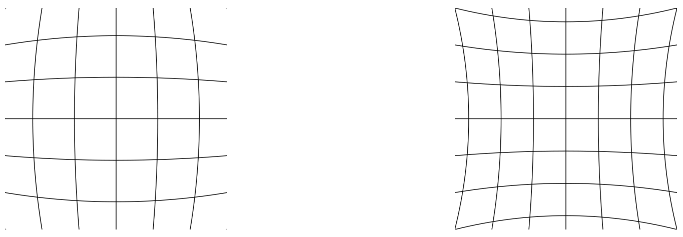
Image distortion occurs when there is a deviation from the straight line assumption of the perspective camera model: that a straight line in the scene is represented as a straight line in the image. This can occur due to deviations from the assumed perspective model and the physical aspect of the lens and sensor of a camera. If not considered, the direct correspondence between a pixel and an observed point in the scene cannot be made.

Distortion can be irregular or occur in patterns. Commonly, radial and tangential distortion is assumed to be the main contributing factors of the distortion [46, 51, 52]. The image can be undistorted as a whole or the distortion model can be included as an extra step in the camera model. Calibration of the camera has to be performed to find the coefficients which determine the distortion.

4.4.1 Radial Distortion

Radial distortion is radially symmetric around the principal point of the sensor. Straight lines in the scene are warped to appear curved in the image, with increasing effect further from the principal point. Radial distortion is often classified into two main types: *barrel distortion* and *pincushion distortion*, which can be observed in figure 4.4.

In barrel distortion, pixels are warped outwards increasingly from the principal point. For pincushion distortion, the opposite is true. Radial distortion, due to radial symmetry, is modelled after a polynomial. Let (x_u, y_u) be undistorted de-homogenised normalised image coordinates. The distorted normalised image coordinates (x_d, y_d) are then given by the model in equation (4.18) [55].



(a) Barrel distortion, from [53]. **(b)** Pincushion distortion, from [54].

Figure 4.4: Examples of radial distortion.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + \cdots + k_n r^{2n}) \begin{bmatrix} x_u \\ y_u \end{bmatrix} \tag{4.18}$$

Here $r = \sqrt{x_u^2 + y_u^2}$ and k_i are the radial distortion coefficients. In barrel distortion, the coefficients k_i will typically be positive, whereas they will be negative for pincushion distortion. However, this depends purely on how the distortion model is defined. If the distortion model is defined as the product of the distortion and the distorted pixel, k_i would typically be negative for barrel distortion and positive for pincushion distortion. The number of distortion coefficients is up to the implementation, but care has to be taken into consideration to not over-fit the model to the data used during calibration.

4.4.2 Tangential Distortion

Tangential distortion occurs when the camera sensor is not parallel to the camera lens, which can be observed in figure 4.5. The tangential distortion can be modelled after equation (4.19) [55].

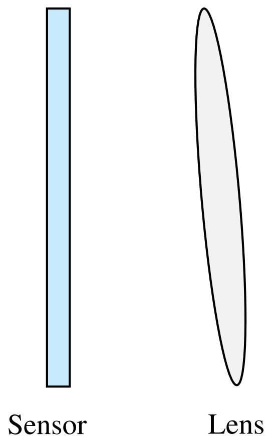


Figure 4.5: Camera and lens not being parallel results in tangential distortion.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} x_u \\ y_u \end{bmatrix} + \begin{bmatrix} 2p_1 x_u y_u + p_2(r^2 + 2x_u^2) \\ 2p_2 x_u y_u + p_1(r^2 + 2y_u^2) \end{bmatrix} \tag{4.19}$$

Here p_1 and p_2 are the tangential distortion coefficients.

4.4.3 Combined Radial-Tangential Distortion

Combined radial-tangential distortion can be modelled after the Brown-Conrady model [55, 56], as shown in equation (4.20).

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + \cdots + k_n r^{2n}) \begin{bmatrix} x_u \\ y_u \end{bmatrix} + \begin{bmatrix} 2p_1 x_u y_u + p_2(r^2 + 2x_u^2) \\ 2p_2 x_u y_u + p_1(r^2 + 2y_u^2) \end{bmatrix} \tag{4.20}$$

The inverse of equation (4.20) has no analytic form, however numeric approximations can be found with iterative methods [57].

4.5 Feature Extraction & Tracking

Feature extraction is the process of finding distinct features in an image (typically corners or areas of an image with strong gradients). Often, a blur is applied before the feature extraction to remove noise from the image. Feature tracking/matching is the process of finding a given feature extracted in image $\mathbf{I}^k$ in image $\mathbf{I}^{k+1}$. This can be observed in figure 4.6. The number of features to extract and track/match is up to the implementation. Extracting more features can increase robustness for the rest of the estimation pipeline, but has the downside of increased computational cost. Moreover, if the feature extraction is done with a low threshold for the distinctiveness of the features — yielding features which can be hard to track/match — more excessive outlier rejection might be necessary for the tracker/matcher.

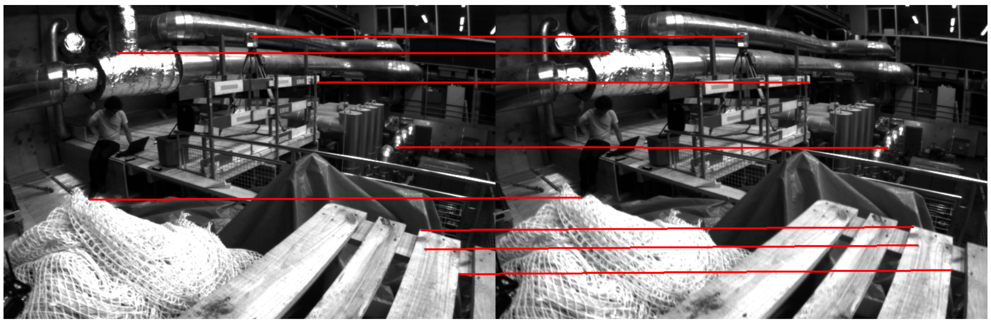


Figure 4.6: Feature extraction and tracking/matching between two images with slightly different viewpoints (images are from the EuRoC dataset [9]).

Feature extraction and tracking pipelines are generally split into two main categories: direct or indirect. Moreover, each category can operate on the visual input in a *sparse* or *dense* manner. The distinction between sparse and dense is the amount of data considered. A sparse method will only concern itself with certain areas of the image, whereas a dense method typically operates on a much greater part of the image or possibly the whole image. In this theory section, only sparse methods will be discussed.

4.5.1 Indirect Sparse Methods

An indirect method extracts features from two images and builds descriptors — identifiers — for these features. It matches features seen in the two images by finding features with the same descriptor.

Let the matched feature be given by the pixel coordinate pair $\mathbf{u}^{c_{k-1}}$ and $\mathbf{u}^{c_k}$ in camera frame c_{k-1} and c_k, respectively. Furthermore, the estimated pixel coordinate in frame c_k, given by the corresponding feature pair pixel coordinate in frame c_{k-1} and the unknown transformation $\mathbf{T}_{c_k c_{k-1}}$, is given by: $\hat{\mathbf{u}}^{c_k} = \pi(\mathbf{T}_{c_k c_{k-1}} \pi^{-1}(\mathbf{u}_i^{c_{k-1}}))$. The *re-projection* error — the error between the estimated and the observed feature pixel coordinate — can then be used to find the transformation from N matched features, as shown in equation (4.21).

$$\mathbf{T}_{c_k c_{k-1}} = \underset{\mathbf{T}_{c_k c_{k-1}}}{\operatorname{argmin}} \sum_{i=0}^{N} \left|\left|\hat{\mathbf{u}}_i^{c_k} - \mathbf{u}_i^{c_k}\right|\right|^2 \tag{4.21}$$

Examples of indirect sparse feature extractors and matchers are SIFT (scale-invariant feature transform) [43], SURF (speeded up robust features) [58] and ORB (oriented FAST and rotated BRIEF) [59]. Both SIFT and SURF are scale and rotation invariant methods, which makes them quite robust. SURF was invented with inspiration from SIFT, trying to alleviate that SIFT is rather computationally demanding. However, SURF also suffers from being one of the most computationally demanding feature extractors in speed and memory usage. Both are based on approximating the Laplacian of Gaussian filter (which works as a blob detector). SIFT uses image gradients around a feature to build descriptors, whereas SURF uses wavelet responses of patches around the feature. ORB utilises FAST [40, 41] for feature extraction and rotated BRIEF [60] to build descriptors. FAST works directly on the pixel intensities without filtering, making it one of the fastest feature extractors. BRIEF descriptors are built by representing a patch of pixel intensities around the feature as a binary string. The Hamming distance — the number of positions at which the bits are different in two binary strings — can then be used to compare the similarity of two arbitrary features. ORB can be an order of magnitude faster than SIFT and SURF, to a minimal cost of less robustness [61].

4.5.2 Direct Sparse Methods

A direct method will also extract features, but descriptors are not constructed. Rather, the transformation between camera frame c_{k-1} and c_k is found by utilising the pixel intensities directly, hence the name. The *photometric* error is minimised rather than the re-projection error, as shown in equation (4.22).

$$\mathbf{T}_{c_k c_{k-1}} = \underset{\mathbf{T}_{c_k c_{k-1}}}{\operatorname{argmin}} \sum_{i=0}^{N} \left|\left|\mathbf{I}^{c_k}\big(\pi(\mathbf{T}_{c_k c_{k-1}} \pi^{-1}(\mathbf{u}_i^{c_{k-1}})\big) - \mathbf{I}^{c_{k-1}}\big(\mathbf{u}_i^{c_{k-1}}\big)\right|\right|^2 \tag{4.22}$$

Here $\mathbf{I}(\mathbf{u})$ is the pixel intensity value in the image for the pixel coordinate $\mathbf{u}$. The transformation of the pixel coordinates in the two images is represented by a rigid transformation in $SE(3)$. However, an affine warp can also be applied.

4.5.3 The Distinction Between Indirect & Direct Sparse Methods

The mathematical difference between an indirect and a direct method lies in the fact that a direct method will not necessarily perform explicit matching. Meaning, it does not need features from frame c_k since the only ones used are those from frame c_{k-1}. The direct methods are typically more efficient and have the benefit that all information in the image can be exploited. That said, they can be less robust than indirect methods when there is a great baseline or when scale and rotation must be considered. Direct methods can alleviate this by imposing small movements between consecutive camera frames. Moreover, a light-varying scene or exposure changes can also become problematic for direct methods, where histogram equalisation of the image can alleviate this to a certain degree.

4.5.4 Features From Accelerated Segment Test

Features from accelerated segment test or *FAST* is a feature extractor created by Rosten and Drummond [40, 41]. It can be utilised in a direct frontend pipeline or an indirect pipeline with an additional descriptor algorithm, as with ORB. It builds on the principle of checking intensities around a candidate pixel to determine if that candidate is a feature.

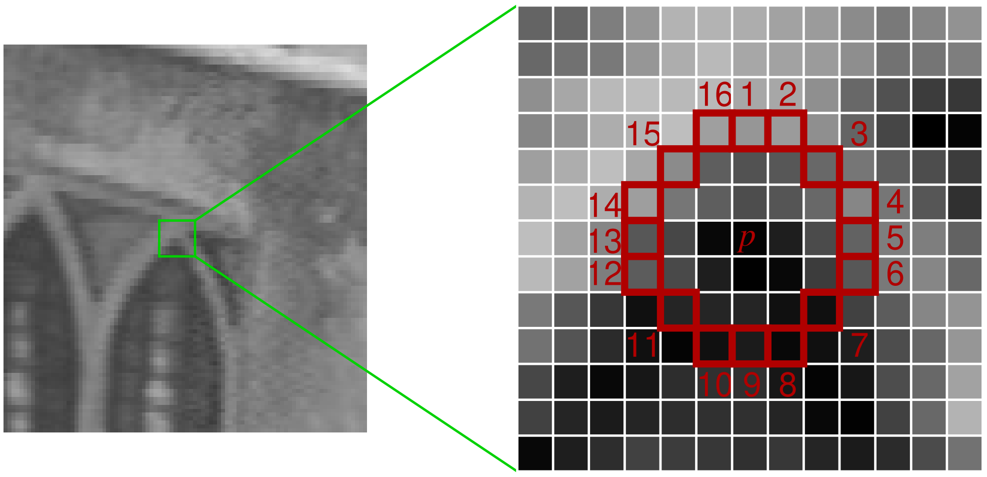


Figure 4.7: FAST pattern. Courtesy of Rosten [62].

Given the pixel intensity $\mathbf{I}(\mathbf{u})$ for a pixel coordinate $\mathbf{u}$, let $\mathbf{u}$ be determined a feature if there are 9 consecutive pixels within the FAST pattern around the pixel which all are either less than $\mathbf{I}(\mathbf{u})-t$ or greater than $\mathbf{I}(\mathbf{u})+t$. t is denoted the threshold and can be adjusted accordingly to allow for more features to be detected by keeping t low or having a stricter test by keeping t big. This can be seen in figure 4.7, where there are 9 consecutive pixels from index 14 to index 6 which all are greater than the pixel candidate tested.

FAST is an efficient method as it allows for quickly rejecting candidates. The requirement of 9 consecutive pixels allows assumptions to be made which can reject candidates with few operations. Mainly, a possible solution needs to include at least one of the pixels opposite each other (e.g. pixel 1 and pixel 9, pixel 5 and pixel 13 etc.), as seen in figure 4.7. If both pixels opposite to each other are within the threshold given by $[\mathbf{I}(\mathbf{u})-t, \mathbf{I}(\mathbf{u})+t]$, the candidate cannot possibly be a feature. This can be done incrementally for each opposite pixel pair, which yields 8 operations for quickly rejecting a pixel before a more thorough test has to be done to check for consecutiveness.

FAST is one of the most efficient feature extractors, but can suffer from poor robustness as it is not invariant to scale and rotation. It does not require much memory compared to SIFT and SURF, which use multiple instances of the image at various levels of scale and blur.

4.5.5 Lucas-Kanade

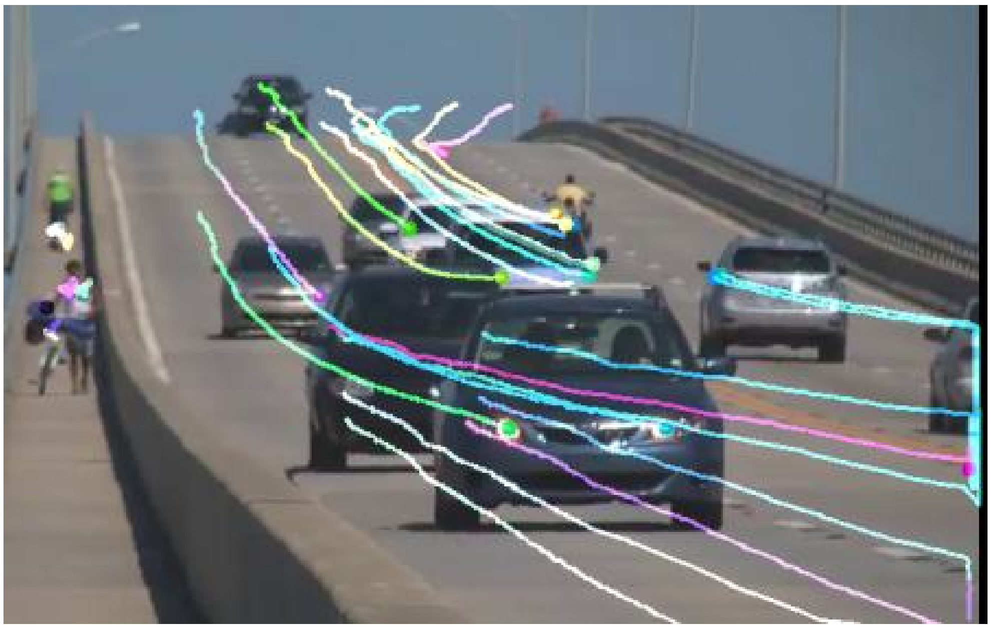


Figure 4.8: Demonstration of optical flow tracking, from [63].

The Lucas-Kanade [42] algorithm is a direct feature tracking method (often called optical flow), which tracks features over a sequence of images. This can be seen in figure 4.8, where pixels are tracked in a sequence of images from cars along a motorway. One can e.g. see that the feature registered in the headlight of the front-most Subaru has been tracked as it has driven along from the top of the bridge.

The fundamental idea behind the algorithm is based on the optical flow equation, where it is assumed that the intensity consistency assumption holds. The intensity consistency assumption assumes that a pixel's intensity will not change between two images, even though its spatial location change.

$$\mathbf{I}(x,y,t) = \mathbf{I}(x+\delta x, y+\delta y, t+\delta t) \tag{4.23}$$

A pixel's intensity in the image is represented by the spatial and temporal locations. The original pixel location and a delta thus yield the same pixel in the second image. With the assumption that the movement of the pixel is small, the pixel intensity in the second image can be represented by a first-order Taylor series expansion:

$$\begin{aligned}\mathbf{I}(x+\delta x, y+\delta y, t+\delta t) &= \mathbf{I}(x,y,t) + \frac{\partial \mathbf{I}}{\partial x}\delta x + \frac{\partial \mathbf{I}}{\partial y}\delta y + \frac{\partial \mathbf{I}}{\partial t}\delta t \\ 0 &= \frac{\partial \mathbf{I}}{\partial x}\delta x + \frac{\partial \mathbf{I}}{\partial y}\delta y + \frac{\partial \mathbf{I}}{\partial t}\delta t\end{aligned} \tag{4.24}$$

Dividing by δt yields:

$$\begin{aligned}0 &= \frac{\partial \mathbf{I}}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial \mathbf{I}}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial \mathbf{I}}{\partial t}\frac{\delta t}{\delta t} \\ 0 &= \frac{\partial \mathbf{I}}{\partial x}v_x + \frac{\partial \mathbf{I}}{\partial y}v_y + \frac{\partial \mathbf{I}}{\partial t}\end{aligned} \tag{4.25}$$

where (v_x, v_y) represents the flow vector: the pixel displacement from the first image to the second image. The partial derivative of the image with respect to x and y are given by horizontal and vertical gradients computed from convolving the image $\mathbf{I}^k$ with *derivative kernels* (e.g. Sobel or Laplacian) [46]. The partial derivative with respect to time is simply given by the difference of intensity values divided by δt.

Convolution of an image by a kernel $\mathbf{G}$ is represented mathematically by $\mathbf{I} * \mathbf{G}$. The operation is performed by sliding a kernel over the image where each entry the kernel overlaps is multiplied with the corresponding entry in the kernel and summed together. An example can be seen in the following illustration, where the image is convoluted with a 3×3 horizontal Sobel kernel to detect the edge (where the intensities go from 0 to 1) in the image. To have the same input size and output size, a border has to be applied to the input, typically through extrapolation.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 5 & \boxed{5} & 0 \\ 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 5 & 5 & 0 \end{pmatrix}$$

Convolution with Sobel kernels thus represents spatial differentiation as it extracts changes in intensity. In this example, only change in the x direction is extracted, but the same can be done vertically — in the y direction — by the vertical Sobel kernel. The Sobel kernels are given by:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{4.26}$$

The intuition behind the time derivative of an image can be built from the following formal definition of a derivative:

$$\frac{\partial \mathbf{I}(x,y)}{\partial t} = \lim_{\delta t \to 0} \frac{\mathbf{I}(x,y,t+\delta t) - \mathbf{I}(x,y,t)}{\delta t} \tag{4.27}$$

As images are captured at discrete time points, both t and δt can be deemed to represent integer values. Thus, the time derivative between two successive frames then becomes:

$$\begin{aligned} \frac{\partial \mathbf{I}(x,y)}{\partial t} &= \frac{\mathbf{I}(x,y,t+1) - \mathbf{I}(x,y,t)}{1} \\ &= \mathbf{I}(x,y,t+1) - \mathbf{I}(x,y,t) \end{aligned} \tag{4.28}$$

Lucas-Kanade assumes that the displacement — or flow — is uniform within a small neighbourhood of a pixel, often denoted as a *patch*. The number of pixels to include in the patch is up to the implementation. More pixels can increase robustness, but will also introduce extra computational cost. If the patch is too big, the assumption that the flow is uniform might become less valid. Commonly, patch sizes of 7×7, 9×9, 11×11, 13×13 or 15×15 are used, but this heavily depends on the setting at hand.

For brevity, let:

$$\mathbf{I}_x(\mathbf{u}) := \frac{\partial \mathbf{I}(\mathbf{u})}{\partial x} = \mathbf{I}(\mathbf{u}) * \mathbf{G}_x \tag{4.29}$$

$$\mathbf{I}_y(\mathbf{u}) := \frac{\partial \mathbf{I}(\mathbf{u})}{\partial y} = \mathbf{I}(\mathbf{u}) * \mathbf{G}_y \tag{4.30}$$

$$\mathbf{I}_t(\mathbf{u}) := \frac{\partial \mathbf{I}(\mathbf{u})}{\partial t} = \mathbf{I}(\mathbf{u}, t+1) - I(\mathbf{u}, t) \tag{4.31}$$

Here $\mathbf{u} = (x, y)$. The constraint of equation (4.25) in the neighbourhood around the pixel can then be written as:

$$\begin{aligned} \mathbf{I}_x(\mathbf{u}_1)v_x + \mathbf{I}_y(\mathbf{u}_1)v_y &= -\mathbf{I}_t(\mathbf{u}_1) \\ \mathbf{I}_x(\mathbf{u}_2)v_x + \mathbf{I}_y(\mathbf{u}_2)v_y &= -\mathbf{I}_t(\mathbf{u}_2) \\ &\dots \\ \mathbf{I}_x(\mathbf{u}_n)v_x + \mathbf{I}_y(\mathbf{u}_n)v_y &= -\mathbf{I}_t(\mathbf{u}_n) \end{aligned} \tag{4.32}$$

In matrix form, the constrains are given by $\mathbf{Av} = \mathbf{b}$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_x(\mathbf{u}_1) & \mathbf{I}_y(\mathbf{u}_1) \\ \mathbf{I}_x(\mathbf{u}_2) & \mathbf{I}_y(\mathbf{u}_2) \\ \dots & \dots \\ \mathbf{I}_x(\mathbf{u}_n) & \mathbf{I}_y(\mathbf{u}_n) \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -\mathbf{I}_t(\mathbf{u}_1) \\ -\mathbf{I}_t(\mathbf{u}_2) \\ \dots \\ -\mathbf{I}_t(\mathbf{u}_n) \end{bmatrix} \tag{4.33}$$

The system is over-determined due to having more equations than unknowns. Therefore, Lucas-Kanade introduces a least-squares representation by pre-multiplying with $\mathbf{A}^T$:

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\mathbf{v} &= \mathbf{A}^T\mathbf{b} \\ \mathbf{v} &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \end{aligned} \tag{4.34}$$

Equation (4.34) can then be written as:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum_i \mathbf{I}_x(\mathbf{u}_i)^2 & \sum_i \mathbf{I}_x(\mathbf{u}_i)\mathbf{I}_y(\mathbf{u}_i) \\ \sum_i \mathbf{I}_x(\mathbf{u}_i)\mathbf{I}_y(\mathbf{u}_i) & \sum_i \mathbf{I}_y(\mathbf{u}_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i \mathbf{I}_x(\mathbf{u}_i)\mathbf{I}_t(\mathbf{u}_i) \\ -\sum_i \mathbf{I}_y(\mathbf{u}_i)\mathbf{I}_t(\mathbf{u}_i) \end{bmatrix} \tag{4.35}$$

$\mathbf{S} := \mathbf{A}^T\mathbf{A}$ is the structure tensor of the image at the pixel candidate. As Harris, Stephens *et al.* formulated for the Harris-Stephens corner detector [64] and Shi and Tomasi formulated for the Shi-Tomasi corner detector [65], the eigenvalues of the structure tensor contains information about the properties of the feature candidate. If either eigenvalue is significantly larger than the other, the pixel candidate lies on an edge, whereas if both eigenvalues are small, then the region is uniform. If both eigenvalues are large and approximately the same magnitude, the pixel candidate is defined as a corner where displacement in both x and y yields a change in intensity. This can be seen in figure 4.9.

It is firstly necessary that the structure tensor is invertible and that both eigenvalues of the structure tensor abide by the constraint of $\lambda_1 \geq \lambda_2 > 0$. Secondly, the requirement of $\lambda_1 \gg 0$ and $\lambda_2 \gg 0$ as well as $\frac{\lambda_1}{\lambda_2} < D$, where D is some not too large threshold, will yield better feature candidates.

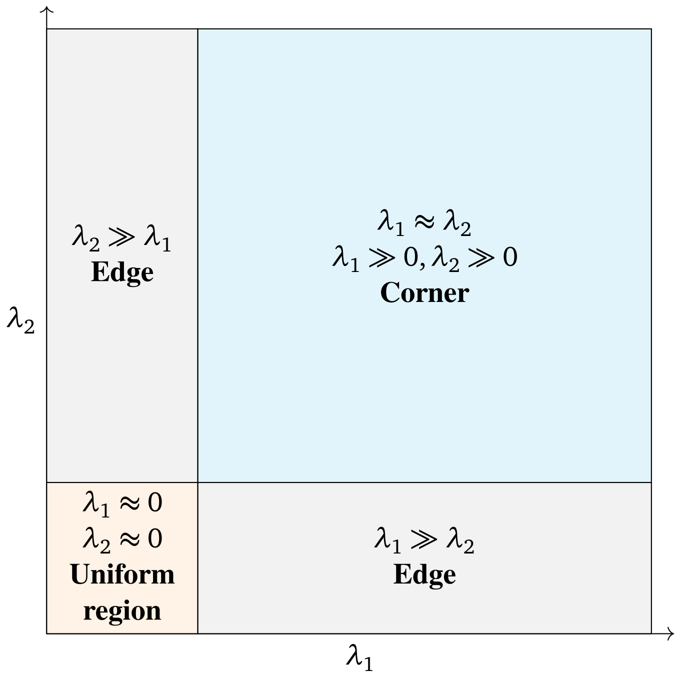

$\lambda_2 \gg \lambda_1$
Edge
$\lambda_1 \approx \lambda_2$
$\lambda_1 \gg 0, \lambda_2 \gg 0$
Corner
λ_2
$\lambda_1 \approx 0$
$\lambda_2 \approx 0$
Uniform region
$\lambda_1 \gg \lambda_2$
Edge
λ_1

Figure 4.9: Eigenvalues of the structure tensor. If the eigenvalues are both $\gg 0$ and somewhat equal in magnitude, then there are strong gradients in the x and y direction. This yields that the patch is situated around a corner as there will be a non-insignificant change in intensity when traversing along the x and y direction. If only one of the gradients is $\gg 0$, then there is a strong gradient in either the x or y direction, yielding that the patch is situated around an edge. If the eigenvalues are small, the patch is uniform with no strong gradients.

With optical flow, it is absolutely essential that sub-pixel intensity values are calculated to provide a robust estimate of the neighbourhood of the feature candidate when the flow vector is applied. This can be done by means of bilinear filtering. Let a pixel coordinate be given by the following:

$$x = x_0 + \alpha_x \qquad\qquad y = y_0 + \alpha_y \tag{4.36}$$

where (x_0, y_0) represent the integer part of the coordinate and (α_x, α_y) represents the fractional part. The intensity value in an image is then given by:

$$\begin{aligned}\mathbf{I}(x,y) = (1-\alpha_x)(1-\alpha_y)\mathbf{I}(x_0,y_0) + \alpha_x(1-\alpha_y)\mathbf{I}(x_0+1,y_0) + \\ (1-\alpha_x)\alpha_y\mathbf{I}(x_0,y_0+1) + \alpha_x\alpha_y\mathbf{I}(x_0+1,y_0+1)\end{aligned} \tag{4.37}$$

The Lucas-Kanade algorithm can be applied iteratively to make a more robust estimate of the flow. The flow vector is then updated during each iteration. The algorithm will terminate when the change in the flow vector is below a given threshold. With this, the time derivative has to be re-calculated for the new estimate of the pixel coordinate in the second image, as the intensities of the patch around the feature will be given by $\mathbf{I}(\mathbf{u}_i + \mathbf{v})$. This will thus require bilinear filtering.

Lucas-Kanade assumes the pixel displacement is small between two images. This can become error-prone for greater camera movements. To alleviate this, image pyramids can be constructed where the images are downscaled at each level, as seen in figure 4.10. At the upper levels of the image pyramid, greater pixel displacements will be downscaled, so the assumptions of Lucas-Kanade are to a greater extent applicable. The displacement at each level is then propagated downwards in the pyramid, upscaled and used as a constant bias, so the relative displacement at each level is small.

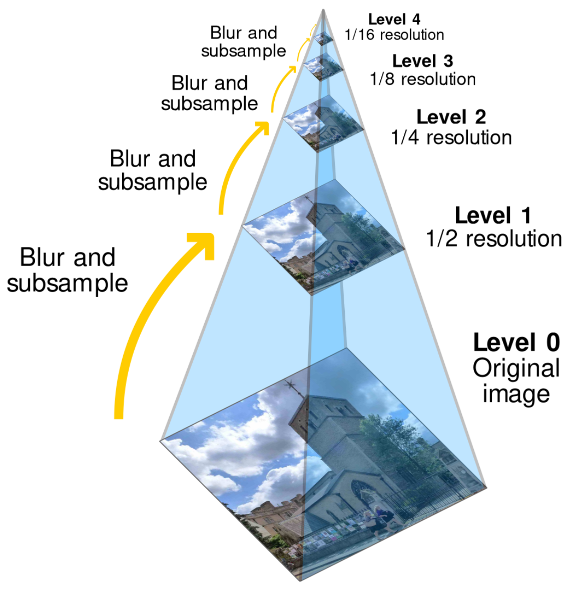


Figure 4.10: Demonstration of an image pyramid, from [66].

The iterative Lucas-Kanade algorithm with image pyramids can be examined in algorithm 1 (where an image at time instant k is represented by $\mathbf{I}^k$). In the algorithm, the vector $\mathbf{g}_p$ stores the displacement from the previous level $p+1$ upscaled by 2, representing the constant bias carried over from each level.

Algorithm 1 Iterative Lucas-Kanade algorithm with image pyramids

```
Input: $\mathbf{I}^k_{pyr}$, image pyramid from image $\mathbf{I}^k$
Input: $\mathbf{I}^{k+1}_{pyr}$, image pyramid from image $\mathbf{I}^{k+1}$
Input: $\begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & ... & \mathbf{u}_n\end{bmatrix}$, feature's neighbourhood pixel coordinates (including itself)
Input: $P$, number of pyramid levels
Input: $\epsilon$, threshold for change in flow vector
Output: $\mathbf{v}$, flow/displacement vector from feature in $\mathbf{I}^k$ to feature in $\mathbf{I}^{k+1}$

 1: for $p \leftarrow P-1$ to 0 do
 2:     $\mathbf{g}_p \leftarrow [0,0]$
 3: end for

 4: for $p \leftarrow P-1$ to 0 do
 5:     $I^k_p \leftarrow I^k_{pyr}[p]$
 6:     $I^{k+1}_p \leftarrow I^{k+1}_{pyr}[p]$

 7:     for $i \leftarrow 0$ to $N$ do
 8:         $\mathbf{I}^k_x(\mathbf{u}_i) \leftarrow \mathbf{I}^k_p(\mathbf{u}_i) * G_x$
 9:         $\mathbf{I}^k_y(\mathbf{u}_i) \leftarrow \mathbf{I}^k_p(\mathbf{u}_i) * G_y$
10:     end for

11:     $\mathbf{v} \leftarrow [0,0]$
12:     $\delta\mathbf{v} \leftarrow [0,0]$
13:     $\mathbf{S} \leftarrow \begin{bmatrix} \sum_i \mathbf{I}^k_x(\mathbf{u}_i)^2 & \sum_i \mathbf{I}^k_x(\mathbf{u}_i)\mathbf{I}^k_y(\mathbf{u}_i) \\ \sum_i \mathbf{I}^k_x(\mathbf{u}_i)\mathbf{I}^k_y(\mathbf{u}_i) & \sum_i \mathbf{I}^k_y(\mathbf{u}_i)^2 \end{bmatrix}$
14:     while $||\delta\mathbf{v}||_2 > \epsilon$ do
15:         for $i \leftarrow 0$ to $N$ do
16:             $\mathbf{I}_t(\mathbf{u}_i) \leftarrow \mathbf{I}^{k+1}(\mathbf{u}_i + \mathbf{v} + \mathbf{g}_p) - \mathbf{I}^k(\mathbf{u}_i)$
17:         end for
18:         $\mathbf{b}^* \leftarrow \begin{bmatrix} -\sum_i \mathbf{I}^k_x(\mathbf{u}_i)\mathbf{I}_t(\mathbf{u}_i) \\ -\sum_i \mathbf{I}^k_y(\mathbf{u}_i)\mathbf{I}_t(\mathbf{u}_i) \end{bmatrix}$
19:         $\delta\mathbf{v} \leftarrow \mathbf{S}^{-1}\mathbf{b}^*$
20:         $\mathbf{v} \leftarrow \mathbf{v} + \delta\mathbf{v}$
21:     end while

22:     if $p > 0$ then
23:         $\mathbf{g}_{p-1} \leftarrow 2(\mathbf{v} + \mathbf{g}_p)$
24:     else
25:         $\mathbf{v} \leftarrow \mathbf{g}_0 + \mathbf{v}$
26:     end if
27: end for
```

4.5.6 Binary Robust Independent Elementary Features

BRIEF (binary robust independent elementary features) [60] is a feature descriptor based on building a binary string of the image data around a given feature. As shown by Calonder *et al.*, BRIEF vastly outperforms SURF [58] when it comes to computational time, with an equivalent recognition rate. It should be mentioned that BRIEF is neither scale nor rotation invariant. However, there exist variants of BRIEF which are rotation invariant, e.g. ORB.

Hamming distance is used to compare the descriptors, such that a feature in one image can be matched with a feature in another image. This idea can also be used to reject track outliers. Given e.g. a direct pipeline where the track of a feature diverges, the descriptor of the feature in image frame $\mathbf{I}^k$ can be compared with the descriptor in frame $\mathbf{I}^{k+1}$ to detect the divergence.

The binary string is built from a pattern on a patch around the feature, where the pattern is given by a set of N pixel coordinate pairs $(\mathbf{u}, \mathbf{v})$. Let the patch around the feature be given by $\mathbf{P}$. The following intensity test then gives each bit in the descriptor:

$$\tau(\mathbf{P}; \mathbf{u}, \mathbf{v}) := \begin{cases} 1 & \text{if } \mathbf{P}(\mathbf{u}) < \mathbf{P}(\mathbf{v}) \\ 0 & \text{otherwise} \end{cases} \tag{4.38}$$

yielding that the binary string can be constructed from:

$$\eta(\mathbf{P}) := \sum_{i=1}^{N} 2^{i-1} \tau(\mathbf{P}; \mathbf{u}_i, \mathbf{v}_i) \tag{4.39}$$

Calonder *et al.* examined several different patterns and strategies for increasing the recognition rate of the descriptor, showing that blur greatly contributes to the robustness of hard-to-match features. Relying on a random pattern was also shown to increase the robustness, preventing over-fitting to a specific type of neighbourhood around a feature. As mentioned, the pattern is constructed from a set of pixel coordinate pairs $(\mathbf{u}, \mathbf{v})$, where the pattern can look something like what is shown in figure 4.11. Typically, the set of pixel pairs is constructed from sampling a uniform or Gaussian distribution.

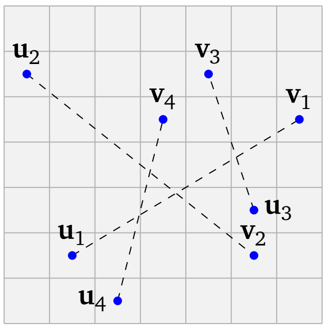


Figure 4.11: Example of a BRIEF pattern.

To compare descriptors, the Hamming distance is utilised. It is given by the number of different bits in the descriptor strings and can be found by a logical XOR followed by counting the number of bits set after the XOR. An example of a Hamming distance of 4 can be seen in figure 4.12.

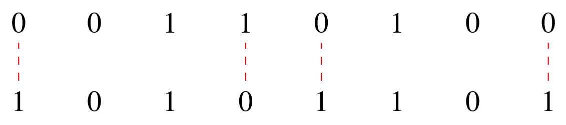


Figure 4.12: Demonstration of the Hamming distance for two 8-bit descriptors.

Rotated BRIEF

To make BRIEF rotation invariant, the moment of the patch can be utilised (note here that (u, v) is a single pixel coordinate in the patch $\mathbf{P}$, where the centre of the patch is given by $\mathbf{o} = (0, 0)$):

$$m_{pq} = \sum_{u,v} u^p v^q \mathbf{I}(u, v) \tag{4.40}$$

This can be used to find the "centre of intensity" in the patch:

$$\mathbf{c} = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{4.41}$$

A metric for the rotation of the patch can then be formed by means of the angle between the line segment between $\mathbf{c}$ and $\mathbf{o}$ and a horizontal line spanning out from $\mathbf{c}$, as shown in figure 4.13.

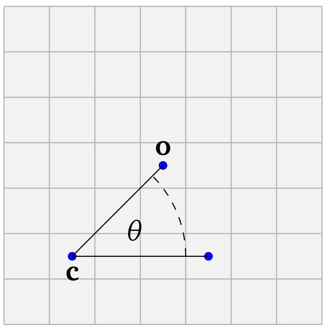


Figure 4.13: Angle between the centre of intensity and the origin of the patch.

The angle can then be constructed from the following:

$$\theta = \operatorname{atan2}\left(\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}} \right) = \operatorname{atan2}(m_{01}, m_{10}) \tag{4.42}$$

where the last equality comes from the fact that $\frac{1}{m_{00}}$ acts solely as a scale factor for the vector **c**. θ can then be used to rotate the patch to a canonical representation, making the descriptor rotation invariant.

Chapter 5

Probabilistic State Estimation

At the core of a visual-inertial odometry backend, a probabilistic state estimation framework will reside. The objective of the framework is to utilise the measurements in the pipeline to provide an estimate, $\hat{\mathbf{x}}$, which is as close to the true state $\mathbf{x}$ as possible, as well as providing an estimate of the uncertainty of the estimate. This section will outline the basics of probability theory, the normal distribution, the chi-squared distribution as well as derive the Kalman Filter and the Extended Kalman Filter, which is used to estimate the state for linear and non-linear systems, respectively.

5.1 Basic Probability Theory

A *stochastic* variable x is modelled in the probability theory as a variable abiding some *probability density function* (PDF). The PDF is a continuous function mapping the likelihood of which observations one can make of x, denoted by $p(x = x')$, where x' is the *realisation* of x. The *cumulative distribution function*, here denoted by $P(x \leq x') = \int_{-\infty}^{x'} p(x = x')dx'$, signifies the probability of x being less than the value x'. Thus, the probability of x residing within the interval a to b is given by $P(a \leq x \leq b) = P(x \leq b) - P(x \leq a)$. The PDF must abide by the constraint that $\int_{-\infty}^{\infty} p(x = x')dx' = 1$, meaning that there must be a 100% chance that when all outcomes are considered, the stochastic variable will fall somewhere in that given range of all its possible outcomes.

Stochastic variables are often parametrised by their *expected* value, and their variance, σ^2. The expected value follows the definition of a weighted average and can be understood as the most likely outcome of the stochastic variable x.

$$\mathbb{E}[x] = \int_{-\infty}^{\infty} x' p(x = x')dx' \tag{5.1}$$

Furthermore, the variance is given by the stochastic variable minus the expected value, squared, as shown in equation (5.2). It can be understood as a measure of the

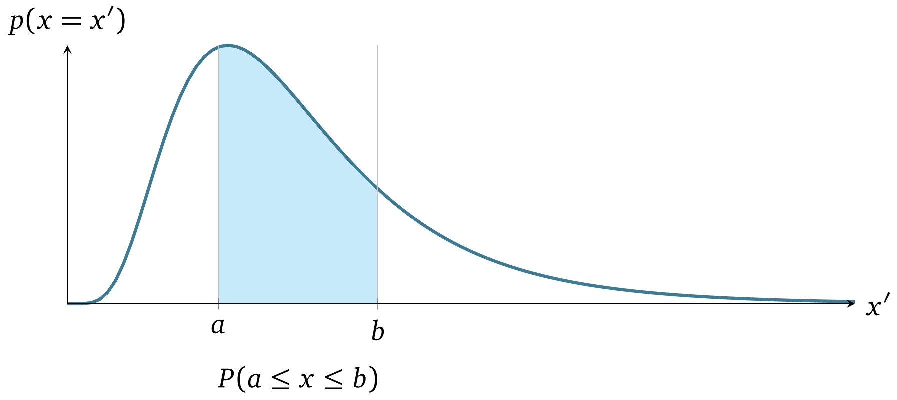


Figure 5.1: The PDF of a log-normal distribution. The integral of the interval a to b yields the probability of finding x within that region.

spread in the outcomes of the stochastic variable x. Moreover, the standard deviation of x, denoted σ, is given by the square root of the variance.

$$
\begin{aligned}
\sigma^2 := \mathbb{E}[(x - \mathbb{E}[x])^2] &= \int_{-\infty}^{\infty} (x' - \mathbb{E}[x])^2 p(x = x') dx' \\
&= \int_{-\infty}^{\infty} (x'^2 - 2x'\mathbb{E}[x] + \mathbb{E}[x]^2) p(x = x') dx \\
&= \int_{-\infty}^{\infty} x'^2 p(x = x') dx' - \mathbb{E}[x] \int_{-\infty}^{\infty} 2x' p(x = x') dx' \\
&\quad + \mathbb{E}[x]^2 \int_{-\infty}^{\infty} p(x = x') dx' \\
&= \int_{-\infty}^{\infty} x'^2 p(x = x') dx - 2\mathbb{E}[x]^2 + \mathbb{E}[x]^2 \\
&= \int_{-\infty}^{\infty} x'^2 p(x = x') dx' - \mathbb{E}[x]^2
\end{aligned}
\tag{5.2}
$$

5.2 The Normal Distribution

The normal distribution is a continuous probability distribution whose PDF follows a bell curve. The normal distribution is often also called the *Gaussian* distribution, and the names will be used interchangeably in this thesis. It is symmetric around its expected value(s) and is widely used within state estimation and various applications of probability theory due to its abundance in the world. The abundance comes from the central limit theorem, which states that given a sufficiently large sample of a given population with an expected value μ and a standard deviation σ, the *sample average* tends towards a normal distribution. For a given sensor such as an accelerometer, the central limit theorem thus states that the error between the actual acceleration and the measured acceleration tends towards a normal distribution, which is a widely used assumption for state estimation within robotics.

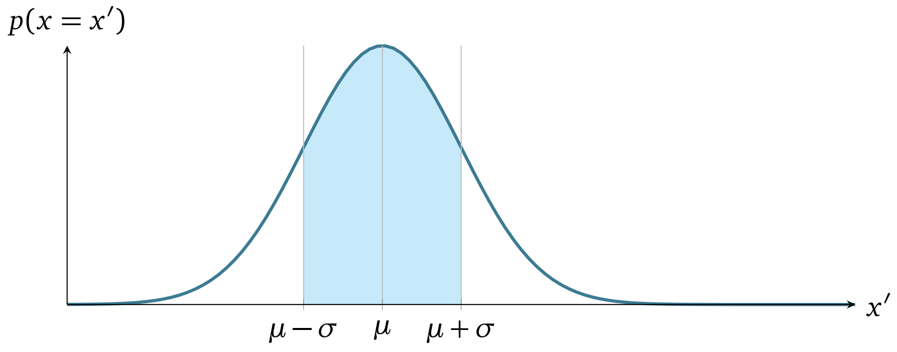


Figure 5.2: The PDF of a normal distribution centred around the mean μ with a variance $\sigma^2 = 1$

The PDF of the normal distribution is given by:

$$p(x = x') = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x'-\mu}{\sigma})^2} \tag{5.3}$$

A particular stochastic variable x being normally distributed is denoted by $x \sim \mathcal{N}(\mu, \sigma^2)$, parametrised by its expected value μ and variance σ^2. The *standard normal distribution* is given by a normal distribution with mean 0 and variance of 1: $\mathcal{N}(0, 1)$. Furthermore, a linear combination of normally distributed stochastic variables is also normally distributed, as well as the joint probability of multiple normally distributed stochastic variables.

The multidimensional stochastic normal variable $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by a range of scalar normally distributed stochastic variables, where $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \mu_2 & \dots & \mu_N \end{bmatrix}^T$ and the *covariance* $\boldsymbol{\Sigma}$ follows the same definition as for the variance, just with vectors instead of scalars. The covariance matrix captures the individual variance of each scalar stochastic variable and the cross-correlation between them. Thus, if the variables are independent, the covariance matrix reduces to a diagonal matrix with the respective independent stochastic variables' variance along the diagonal.

$$\boldsymbol{\Sigma} := \mathrm{Cov}[\mathbf{x}, \mathbf{x}] = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \tag{5.4}$$

The PDF for a multidimensional stochastic normal variable is given by:

$$p(\mathbf{x} = \mathbf{x}') = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}'-\boldsymbol{\mu})^T \boldsymbol{\Sigma}(\mathbf{x}'-\boldsymbol{\mu})} \tag{5.5}$$

For a given linear mapping $\mathbf{y} = \mathbf{A}\mathbf{x}$, the stochastic multidimensional variable $\mathbf{y}$ will be distributed according to $\mathcal{N}(\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$, which can be derived from the definition of the expected value and covariance, as shown in equation (5.6) and equation (5.7).

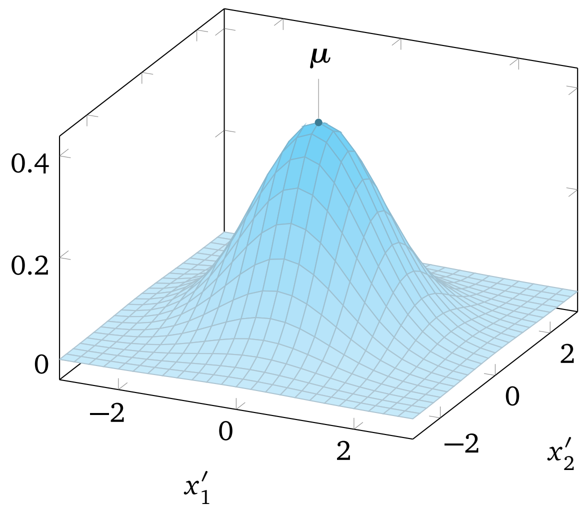


Figure 5.3: The joint PDF of a two-dimensional normally distributed stochastic variable.

$$\mathbb{E}[\mathbf{y}] = \mathbb{E}[\mathbf{Ax}] = \mathbf{A}\mathbb{E}[\mathbf{x}] = \mathbf{A}\boldsymbol{\mu} \tag{5.6}$$

$$\begin{aligned}\mathbb{E}[(\mathbf{Ax} - \mathbb{E}[\mathbf{Ax}])(\mathbf{Ax} - \mathbb{E}[\mathbf{Ax}])^T] &= \mathbb{E}[(\mathbf{Ax} - \mathbf{A}\mathbb{E}[\mathbf{x}])(\mathbf{Ax} - \mathbf{A}\mathbb{E}[\mathbf{x}])^T] \\ &= \mathbb{E}[\mathbf{A}(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T\mathbf{A}^T] \\ &= \mathbf{A}\mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T]\mathbf{A}^T \\ &= \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T\end{aligned} \tag{5.7}$$

5.3 The Chi-Squared Distribution

The Chi-Squared, or χ^2, distribution is defined as a stochastic variable consisting of the sum of k squared standard normal stochastic variables:

$$Q = \sum_{i=0}^{k} z_i^2 \qquad\qquad z_i \sim \mathcal{N}(0,1) \tag{5.8}$$

A χ^2 distribution is parameterised by the *degrees of freedom* k, which directly relates to how many normal stochastic variables there are in the sum.

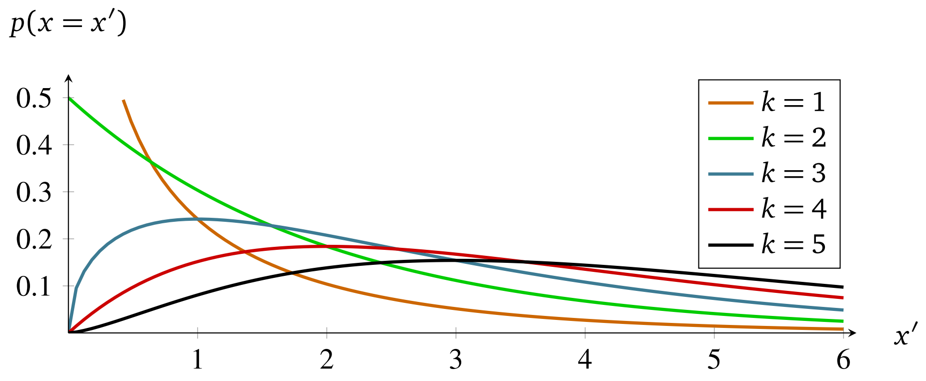


Figure 5.4: The PDF of a χ^2 distribution with different degrees of freedom.

5.3.1 The Mahalanobis Distance Test

Due to the abundance of normally distributed stochastic variables, the Mahalanobis distance test [67] is useful for determining how well a realisation of a set of stochastic normal variables matches the distribution. For a given N-dimensional stochastic variable $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}, \mathbf{P})$, it is defined as:

$$D^2 = (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{P}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) \sim \chi^2(N) \tag{5.9}$$

As D^2 is χ^2 distributed, its realisation can be tested against the confidence interval for a χ^2 variable with N degrees of freedom, yielding a statistic which can remove outliers that do not fit the hypothesis given by $\mathbf{x}$ and its associated mean and covariance. This can be observed in figure 5.5, where a 90 % confidence interval is used. The test would pass for any value between x'_l and x'_r.

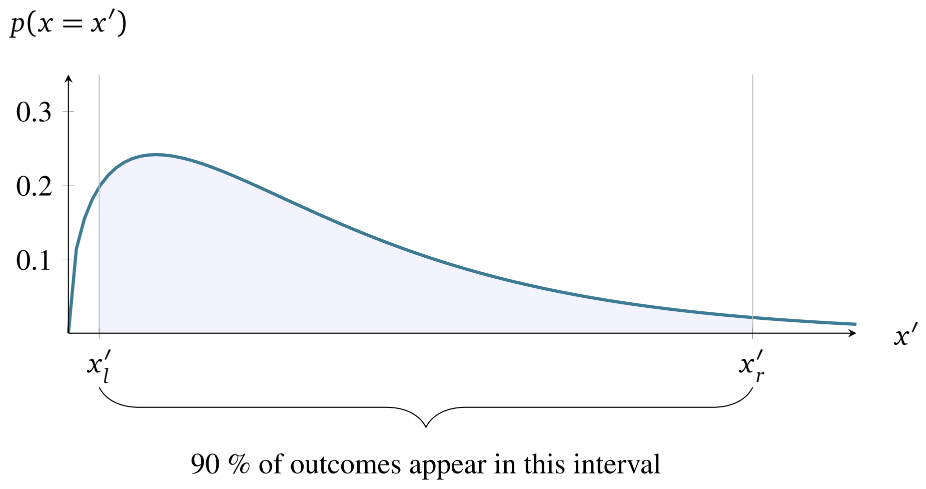


Figure 5.5: 90 % confidence interval for a χ^2 distribution with 3 degrees of freedom.

5.4 The Kalman Filter

The Kalman Filter [68] is a recursive algorithm estimating the state of a linear system based on a mathematical model of the system and measurements which can be related to the states in the system. The Kalman Filter assumes both measurements and the process model being affected by noise, where the optimality of the filter relies on the noise being Gaussian and zero-mean.

The filter is built as a Markov chain, where the likelihood of the next state only depends on the previous state. This can be observed in figure 5.6, where the state of the system is represented in a graph and z are the measurements taken at each time step for x.

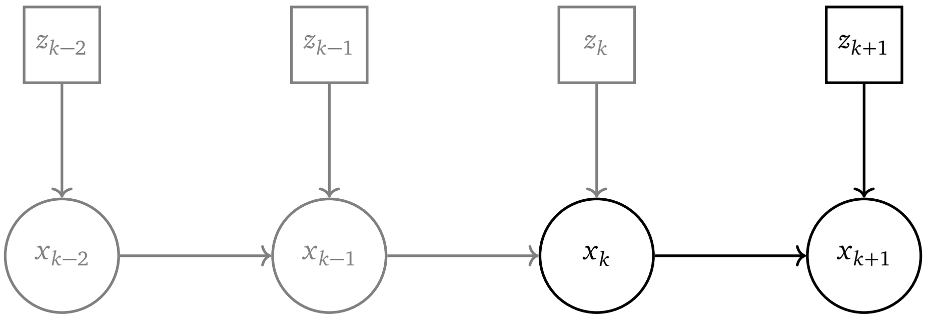


Figure 5.6: Graph demonstrating recursive probabilistic state estimation, such as the Kalman Filter.

5.4.1 State Representation & Dynamics

Let the *discrete* system dynamics be given by the following:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \quad \mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \\ \mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned} \tag{5.10}$$

Here, $\boldsymbol{A}$ propagates the state to the next time step along with the input matrix $\mathbf{B}$ and the input $\mathbf{u}_k$. The state is modelled after a multidimensional Gaussian: $\mathbf{x}_k \sim \mathcal{N}(\hat{\mathbf{x}}_k, \mathbf{P}_k)$, where $\mathbf{P}_k$ is the covariance matrix of the stochastic variable $\mathbf{x}_k$, denoting the uncertainty in the error between the actual value and the estimated value. Furthermore, it is assumed that some white additive Gaussian noise process affects the model (due to e.g. model uncertainties). It is also assumed that the process noise is uncorrelated with the state, meaning $\mathbb{E}[\mathbf{x}_k\mathbf{w}_k] = \mathbb{E}[\mathbf{x}_k]\mathbb{E}[\mathbf{w}_k] \Rightarrow \text{Cov}[\mathbf{x}_k, \mathbf{w}_k] = \mathbf{0}$. The measurement is related to the state by the measurement matrix $\mathbf{H}$. The measurement is also populated by a white additive Gaussian noise process $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R})$, where $\mathbf{x}_k$ and $\mathbf{v}_k$ are uncorrelated.

(5.11)

5.4.2 Prediction Step

The Kalman Filter will predict the state of the system based on the system dynamics outlined in equation (5.10). The Gaussian $\mathbf{x}_{k+1}$ is a linear sum of Gaussian stochastic variables, making itself a Gaussian. Here the notation $\mathbf{x}_{k+1}^-$ signifies that this is an *a priori* estimate.

$$\begin{aligned}\mathbb{E}[\mathbf{x}_{k+1}^-] &= \mathbb{E}[\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{n}_k] \\ &= \mathbf{A}\mathbb{E}[\mathbf{x}_k] + \mathbf{B}\mathbf{u}_k + \mathbb{E}[\mathbf{n}_k] \\ &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k + \mathbf{0} \\ &= \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k\end{aligned} \tag{5.12}$$

$$\begin{aligned}\mathbf{P}_{k+1}^- &= \mathbb{E}[(\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{n}_k - \mathbb{E}[\mathbf{x}_{k+1}^-])(\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{n}_k - \mathbb{E}[\mathbf{x}_{k+1}^-])^T] \\ &= \mathbb{E}[(\mathbf{A}\mathbf{x}_k + \cancel{\mathbf{B}\mathbf{u}_k} + \mathbf{n}_k - \mathbf{A}\hat{\mathbf{x}}_k - \cancel{\mathbf{B}\mathbf{u}_k})(\mathbf{A}\mathbf{x}_k + \cancel{\mathbf{B}\mathbf{u}_k} + \mathbf{n}_k - \mathbf{A}\hat{\mathbf{x}}_k - \cancel{\mathbf{B}\mathbf{u}_k})^T] \\ &= \mathbb{E}[(\mathbf{A}\mathbf{x}_k + \mathbf{n}_k - \mathbf{A}\hat{\mathbf{x}}_k)(\mathbf{A}\mathbf{x}_k + \mathbf{n}_k - \mathbf{A}\hat{\mathbf{x}}_k)^T] \\ &= \mathbb{E}[\mathbf{A}\mathbf{x}_k\mathbf{x}_k^T\mathbf{A}^T + \mathbf{A}\mathbf{x}_k\mathbf{n}_k^T - \mathbf{A}\mathbf{x}_k\hat{\mathbf{x}}_k^T\mathbf{A}^T \\ &\quad + \mathbf{n}_k\mathbf{x}_k^T\mathbf{A}^T + \mathbf{n}_k\mathbf{n}_k^T - \mathbf{n}_k\hat{\mathbf{x}}_k^T\mathbf{A}^T \\ &\quad - \mathbf{A}\hat{\mathbf{x}}_k\mathbf{x}_k^T\mathbf{A}^T - \mathbf{A}\hat{\mathbf{x}}_k\mathbf{n}_k^T + \mathbf{A}\hat{\mathbf{x}}_k\hat{\mathbf{x}}_k^T\mathbf{A}^T] \\ &= \mathbb{E}[(\mathbf{A}\mathbf{x}_k - \mathbf{A}\hat{\mathbf{x}}_k)(\mathbf{x}_k^T\mathbf{A}^T - \hat{\mathbf{x}}_k^T\mathbf{A}^T)] \\ &\quad + \cancel{\mathbb{E}[\mathbf{A}\mathbf{x}_k\mathbf{n}_k^T]} + \cancel{\mathbb{E}[\mathbf{n}_k\mathbf{x}_k^T\mathbf{A}^T]} + \mathbb{E}[\mathbf{n}_k\mathbf{n}_k^T] - \cancel{\mathbb{E}[\mathbf{n}_k]\hat{\mathbf{x}}_k^T\mathbf{A}^T} - \cancel{\mathbf{A}\hat{\mathbf{x}}_k\mathbb{E}[\mathbf{n}_k^T]} \\ &= \mathbb{E}[(\mathbf{A}\mathbf{x}_k - \mathbf{A}\hat{\mathbf{x}}_k)(\mathbf{A}\mathbf{x}_k - \mathbf{A}\hat{\mathbf{x}}_k)^T] + \mathbb{E}[\mathbf{n}_k\mathbf{n}_k^T] \\ &= \mathbf{A}\mathbb{E}[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T]\mathbf{A}^T + \mathbf{Q} \\ &= \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q}\end{aligned} \tag{5.13}$$

In the derivation for the covariance, the cancellations come from the assumption that $\mathbf{x}_k$ and $\mathbf{n}_k$ are uncorrelated, such that $\mathbb{E}[\mathbf{x}_k\mathbf{n}_k^T] = \mathbb{E}[\mathbf{x}_k]\mathbb{E}[\mathbf{n}_k^T] = \mathbb{E}[\mathbf{x}_k] \cdot \mathbf{0}$. Moreover, also given by the mean of $\mathbf{n}_k$ being $\mathbf{0}$, its covariance is given by $\mathbb{E}[\mathbf{n}_k\mathbf{n}_k^T]$, which can be confirmed from equation (5.4). This yields that:

$$\mathbf{x}_{k+1}^- \sim \mathcal{N}(\mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}, \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q}) \tag{5.14}$$

This concludes the prediction step of the Kalman Filter.

5.4.3 Update Step

By definition, the covariance of the state is given by:

$$\mathbf{P}_k = \mathbb{E}[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T] \tag{5.15}$$

Let the *a posteriori* predicted state be given by the a priori plus a correction term — denoted by $\mathbf{K}_{k+1}$ (which is the *Kalman gain*) — multiplied with the difference between the measurements and the predicted measurements.

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} &= \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbb{E}[\hat{\mathbf{z}}_{k+1}^{-}]) \\ &= \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}(\mathbf{H}\mathbf{x}_{k+1} + \mathbf{v}_{k+1} - \mathbf{H}\hat{\mathbf{x}}_{k+1}^{-}) \\ &= \hat{\mathbf{x}}_{k+1}^{-} + \mathbf{K}_{k+1}\mathbf{H}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) + \mathbf{K}_{k+1}\mathbf{v}_{k+1}\end{aligned} \tag{5.16}$$

This error in the state estimate is then given by:

$$\begin{aligned}\delta\mathbf{x}_{k+1} &= \mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1} \\ &= \mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1} \\ &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1}\end{aligned} \tag{5.17}$$

Substituting equation (5.17) into the definition of the covariance matrix of $\mathbf{x}_{k+1}$ yields the following. The cross correlations between the measurement noise $\mathbf{v}_k$ and $\mathbf{x}_{k+1}$ and the terms relating to $\mathbb{E}[\mathbf{v}_k] = \mathbf{0}$ have been cancelled implicitly in the derivation.

$$\begin{aligned}\mathbf{P}_{k+1} &= \mathbb{E}[((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1}) \\ &\quad ((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-}) - \mathbf{K}_{k+1}\mathbf{v}_{k+1})^T] \\ &= \mathbb{E}[((\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})(\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^{-})^T(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})^T] \\ &\quad + \mathbb{E}[\mathbf{K}_{k+1}\mathbf{v}_{k+1}\mathbf{v}_{k+1}^T\mathbf{K}_{k+1}^T] \\ &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})^T + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}_{k+1}^T\end{aligned} \tag{5.18}$$

The trace (the sum along the main diagonal) of the covariance matrix corresponds directly with the error in the state. The covariance matrix is by definition positive semi-definite. Thus, for the error to be minimised, the partial derivative of the trace of $\mathbf{P}_{k+1}$ is taken with respect to the Kalman gain and evaluated at where the derivative is $\mathbf{0}$.

$$\begin{aligned}\mathbf{P}_{k+1} &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})^T + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}_{k+1}^T \\ &= (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}_{k+1}^{-}(\mathbf{I} - \mathbf{H}^T\mathbf{K}_{k+1}^T) + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}_{k+1}^T \\ &= (\mathbf{P}_{k+1}^{-} - \mathbf{K}_{k+1}\mathbf{H}\mathbf{P}_{k+1}^{-})(\mathbf{I} - \mathbf{H}^T\mathbf{K}_{k+1}^T) + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}_{k+1}^T \\ &= \mathbf{P}_{k+1}^{-} - \mathbf{P}_{k+1}^{-}\mathbf{H}^T\mathbf{K}_{k+1}^T - \mathbf{K}_{k+1}\mathbf{H}\mathbf{P}_{k+1}^{-} \\ &\quad + \mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}_{k+1}^{-}\mathbf{H}^T + \mathbf{R})\mathbf{K}_{k+1}^T\end{aligned} \tag{5.19}$$

$$\begin{aligned}\mathrm{tr}(\mathbf{P}_{k+1}) &= \mathrm{tr}(\mathbf{P}_{k+1}^{-}) - \mathrm{tr}(\mathbf{P}_{k+1}^{-}\mathbf{H}^T\mathbf{K}_{k+1}^T) - \mathrm{tr}(\mathbf{K}_{k+1}\mathbf{H}\mathbf{P}_{k+1}^{-}) \\ &\quad + \mathrm{tr}(\mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}_{k+1}^{-}\mathbf{H}^T + \mathbf{R})\mathbf{K}_{k+1}^T) \\ &= \mathrm{tr}(\mathbf{P}_{k+1}^{-}) - 2\mathrm{tr}(\mathbf{K}_{k+1}\mathbf{H}\mathbf{P}_{k+1}^{-}) \\ &\quad + \mathrm{tr}(\mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}_{k+1}^{-}\mathbf{H}^T + \mathbf{R})\mathbf{K}_{k+1}^T)\end{aligned} \tag{5.20}$$

Here, the fact that the trace of a given matrix is equal to its transpose has been utilised to combine the trace of $\mathbf{P}_{k+1}^{-}\mathbf{H}^T\mathbf{K}_{k+1}^T$ and $\mathbf{K}_{k+1}\mathbf{H}\mathbf{P}_{k+1}^{-}$. The derivative of the trace can then be taken:

$$\begin{aligned}\frac{\partial \operatorname{tr}(\mathbf{P}_{k+1})}{\partial \mathbf{K}_{k+1}} &= \cancel{\frac{\partial \operatorname{tr}(\mathbf{P}^-_{k+1})}{\partial \mathbf{K}_{k+1}}} - \frac{\partial 2\operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}\mathbf{P}^-_{k+1})}{\partial \mathbf{K}_{k+1}} \\ &+ \frac{\partial \operatorname{tr}(\mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})\mathbf{K}^T_{k+1})}{\partial \mathbf{K}_{k+1}} \\ &= -\frac{\partial 2\operatorname{tr}(\mathbf{K}_{k+1}\mathbf{H}\mathbf{P}^-_{k+1})}{\partial \mathbf{K}_{k+1}} \\ &+ \frac{\partial \operatorname{tr}(\mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})\mathbf{K}^T_{k+1})}{\partial \mathbf{K}_{k+1}} \\ &= -2(\mathbf{H}\mathbf{P}^-_{k+1})^T + 2\mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})\end{aligned} \tag{5.21}$$

where $\frac{\partial \operatorname{tr}(\mathbf{AB})}{\partial \mathbf{A}} = \mathbf{B}^T$ and $\frac{\partial \operatorname{tr}(\mathbf{ABA}^T)}{\partial \mathbf{A}} = 2\mathbf{AB}$ has been utilised. This yields that the optimal Kalman gain, given by the derivative being $\mathbf{0}$, is:

$$\begin{aligned}\mathbf{K}_{k+1} &= (\mathbf{H}\mathbf{P}^-_{k+1})^T(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})^{-1} \\ &= \mathbf{P}^-_{k+1}\mathbf{H}^T(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})^{-1}\end{aligned} \tag{5.22}$$

where $\mathbf{P}^-_{k+1} = (\mathbf{P}^-_{k+1})^T$ due to the covariance matrix by definition being symmetric. The updated covariance matrix can then be rewritten according to:

$$\begin{aligned}\mathbf{P}_{k+1} &= \mathbf{P}^-_{k+1} - \mathbf{P}^-_{k+1}\mathbf{H}^T\mathbf{K}^T_{k+1} - \mathbf{K}_{k+1}\mathbf{H}\mathbf{P}^-_{k+1} \\ &+ \mathbf{K}_{k+1}(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})\mathbf{K}^T_{k+1} \\ &= \mathbf{P}^-_{k+1} - \mathbf{P}^-_{k+1}\mathbf{H}^T\mathbf{K}^T_{k+1} - \mathbf{K}_{k+1}\mathbf{H}\mathbf{P}^-_{k+1} \\ &+ \mathbf{P}^-_{k+1}\mathbf{H}^T\,\cancel{(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})^{-1}(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})}\mathbf{K}^T_{k+1} \\ &= \mathbf{P}^-_{k+1} - \cancel{\mathbf{P}^-_{k+1}\mathbf{H}^T\mathbf{K}^T_{k+1}} - \mathbf{K}_{k+1}\mathbf{H}\mathbf{P}^-_{k+1} \\ &+ \cancel{\mathbf{P}^-_{k+1}\mathbf{H}^T\mathbf{K}^T_{k+1}} \\ &= \mathbf{P}^-_{k+1} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}^-_{k+1} \\ &= (\mathbf{I} - \mathbf{H}^T\mathbf{K}^T_{k+1})\mathbf{P}^-_{k+1}\end{aligned} \tag{5.23}$$

This concludes the needed steps for updating the filter:

$$\mathbf{K}_{k+1} = \mathbf{P}^-_{k+1}\mathbf{H}^T(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})^{-1} \tag{5.24}$$

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}^-_{k+1} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbf{H}\hat{\mathbf{x}}^-_{k+1}]) \tag{5.25}$$

$$\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}^-_{k+1} \tag{5.26}$$

5.4.4 Intuitive Sense

The Kalman Filter effectively behaves as a blending function between the process model, which propagates the state, and the measurements which correct the state. During the prediction step the uncertainty in the estimates grows and in the update step, this uncertainty is reduced with the help of the measurement.

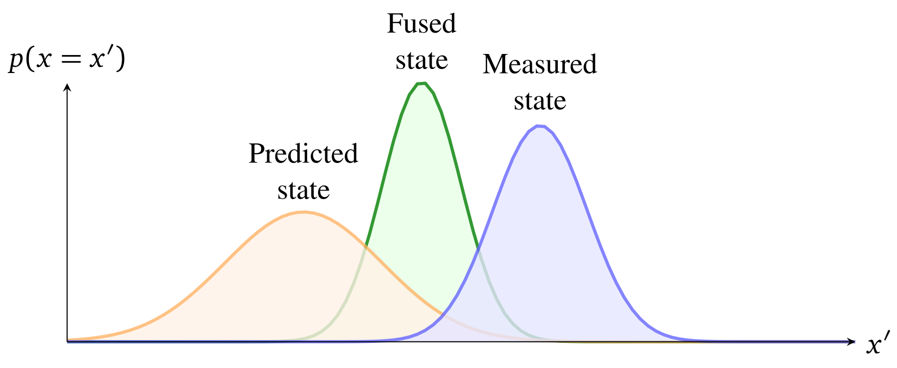


Figure 5.7: Fusion of the prediction and the measured state.

5.4.5 Summary

The Kalman Filter can be summarised by the equations shown in table 5.1.

Initialise with initial estimate	$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{P}_0)$
Prediction step	$\hat{\mathbf{x}}^-_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}$ $\hat{\mathbf{P}}^-_{k+1} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q}$
Update step	$\mathbf{K}_{k+1} = \mathbf{P}^-_{k+1}\mathbf{H}^T(\mathbf{H}\mathbf{P}^-_{k+1}\mathbf{H}^T + \mathbf{R})^{-1}$ $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}^-_{k+1} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbf{H}\hat{\mathbf{x}}^-_{k+1})$ $\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H})\mathbf{P}^-_{k+1}$

Table 5.1: Summary of the Kalman Filter.

5.5 The Extended Kalman Filter

The Extended Kalman Filter is an extension of the Kalman Filter for non-linear systems. The process model and measurement function are linearised to provide a similar prediction and update structure as with the Kalman Filter. For inertial navigation, a regular Kalman Filter cannot be used due to the non-linearities in the system. This is where the Extended Kalman Filter comes to aid.

Note that in the following derivation, the continuous-discrete version of the filter will be discussed, where the prediction step is derived continuously and the update step happens at fixed intervals and is thus discrete. The notation $\mathbf{x}(t)$ is used to denote a continuous variable, whereas $\mathbf{x}_k$ is the discrete version.

5.5.1 State Representation & Dynamics

The system dynamics are given by the non-linear function $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ and the measurement prediction function is given by $\mathbf{h}(\mathbf{x}_k)$:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{g}(\mathbf{x}(t), \mathbf{w}(t)) \quad & \mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}(t)) \\ \mathbf{z}_{k+1} &= \mathbf{h}(\mathbf{x}_{k+1}) + \mathbf{v}_{k+1} & \mathbf{v}_{k+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned} \tag{5.27}$$

where $\mathbf{w}(t)$ is a Weiner process with incremental covariance and independent increments between time intervals and $\mathbf{g}$ is a function mapping the noise to the state.

5.5.2 Prediction Step

By utilising the expected operator on the dynamics, a representation of the a priori dynamics of the system can be found. The a priori state at time step $k+1$ can then be found by the solution to this differential equation with the initial condition $\hat{\mathbf{x}}(t_k) = \hat{\mathbf{x}}_k$ (note that $\mathbb{E}[\mathbf{g}(\mathbf{x}(t), \mathbf{w}(t)] = \mathbf{0}$ as $\mathbf{x}(t)$ and $\mathbf{w}(t)$ are uncorrelated).

$$\begin{aligned} \dot{\hat{\mathbf{x}}}^-(t) &= \mathbb{E}[\mathbf{f}(\mathbf{x}^-(t), \mathbf{u}(t)) + \mathbf{g}(\mathbf{x}^-(t), \mathbf{w}(t))] \\ &= \mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t)) \end{aligned} \tag{5.28}$$

Moreover, let the first order Taylor expansion of $\mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ and $\mathbf{g}(\mathbf{x}(t), \mathbf{w}(t))$ around the a priori estimate for the state be given by:

$$\begin{aligned} \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) &\approx \mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t)) + \underbrace{\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}(t)}\Big|_{\mathbf{x}(t)=\hat{\mathbf{x}}^-(t)}}_{\mathbf{F}(t)} \delta\mathbf{x}(t) \\ &\approx \mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t)) + \mathbf{F}(t)\delta\mathbf{x}(t) \end{aligned} \tag{5.29}$$

$$\begin{aligned} \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) &\approx \mathbf{g}(\hat{\mathbf{x}}^-(t), \mathbf{w}(t)) + \underbrace{\frac{\partial \mathbf{g}(\mathbf{x}(t), \mathbf{w}(t))}{\partial \mathbf{w}(t)}\Big|_{\mathbf{x}(t)=\hat{\mathbf{x}}^-(t)}}_{\mathbf{G}(t)} \delta\mathbf{w}(t) \\ &\approx \mathbf{g}(\hat{\mathbf{x}}^-(t), \mathbf{w}(t)) + \mathbf{G}(t)(\mathbf{w}(t) - \mathbb{E}[\mathbf{w}(t)]) \\ &\approx \mathbf{g}(\hat{\mathbf{x}}^-(t), \mathbf{w}(t)) + \mathbf{G}(t)\mathbf{w}(t) \end{aligned} \tag{5.30}$$

This allows for the linearised dynamics of the error-state around the a priori estimate to be denoted as:

$$\begin{aligned} \dot{\delta}\mathbf{x}(t) &= \dot{\mathbf{x}}^-(t) - \dot{\hat{\mathbf{x}}}^-(t) \\ &\approx \cancel{\mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t))} + \mathbf{F}(t)\delta\mathbf{x}(t) + \cancel{\mathbf{g}(\hat{\mathbf{x}}^-(t), \mathbf{w}(t))} \\ &\quad + \mathbf{G}(t)\mathbf{w}(t) - \cancel{\mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t))} - \cancel{\mathbf{g}(\hat{\mathbf{x}}^-(t), \mathbf{w}(t))} \\ &\approx \mathbf{F}(t)\delta\mathbf{x}(t) + \mathbf{G}(t)\mathbf{w}(t) \end{aligned} \tag{5.31}$$

This linear time-varying stochastic differential system has the solution (see [69], chapter 4.3):

$$\delta\mathbf{x}(t) = \mathbf{\Phi}(t, t_k)\delta\mathbf{x}(t_k) + \int_{t_k}^{t} \mathbf{\Phi}(t, \tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau \tag{5.32}$$

where $\mathbf{\Phi}(t, t_k)$ is the *transition* matrix of the system, given by the following (see [70], chapter 12.2):

$$\mathbf{\Phi}(t, t_k) = \exp\Big(\int_{t_k}^{t} \mathbf{F}(\tau)d\tau\Big) \tag{5.33}$$

which fundamentally encodes the (closed-form) solution to the differential equations of the linearised system, allowing for propagating the covariance from one time point to another. The transition matrix can be found analytically by integrating the error-state dynamics. If there is no closed-form analytical solution, numerical integration can for example be used to approximate $\mathbf{\Phi}(t, t_k)$ by the solution of the derivative of equation (5.33):

$$\dot{\mathbf{\Phi}}(t, t_k) = \mathbf{F}(t)\mathbf{\Phi}(t, t_k) \tag{5.34}$$

$$\mathbf{\Phi}(t_k, t_k) = \mathbf{I} \tag{5.35}$$

The a priori covariance of $\mathbf{x}^{-}(t)$ by definition is then given by equation (5.36).

$$
\begin{aligned}
\mathbf{P}_{k+1}^{-}(t) &= \mathbb{E}[\delta\mathbf{x}(t)\delta\mathbf{x}(t)^T] \\
&= \mathbb{E}[(\mathbf{\Phi}(t,t_k)\delta\mathbf{x}(t_k) + \int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau) \\
&\quad (\mathbf{\Phi}(t,t_k)\delta\mathbf{x}(t_k) + \int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau)^T] \\
&= \mathbb{E}[\mathbf{\Phi}(t,t_k)\delta\mathbf{x}(t_k)\delta\mathbf{x}(t_k)^T\mathbf{\Phi}(t,t_k)^T] \\
&\quad + \mathbb{E}[(\int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau)(\int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau)^T] \\
&= \mathbf{\Phi}(t,t_k)\mathbb{E}[\delta\mathbf{x}(t_k)\delta\mathbf{x}(t_k)^T]\mathbf{\Phi}(t,t_k)^T \\
&\quad + \mathbb{E}[(\int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)d\tau)(\int_{t_k}^{t} \mathbf{w}(\tau)^T\mathbf{G}(\tau)^T\mathbf{\Phi}(t,\tau)^T d\tau)] \qquad (5.36) \\
&= \mathbf{\Phi}(t,t_k)\mathbf{P}_k(t_k)\mathbf{\Phi}(t,t_k)^T \\
&\quad + \mathbb{E}[\int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}(\tau)\mathbf{w}(\tau)^T\mathbf{G}(\tau)^T\mathbf{\Phi}(t,\tau)^T d\tau] \\
&= \mathbf{\Phi}(t,t_k)\mathbf{P}_k(t_k)\mathbf{\Phi}(t,t_k)^T \\
&\quad + \int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbb{E}[\mathbf{w}(\tau)\mathbf{w}(\tau)^T]\mathbf{G}(\tau)^T\mathbf{\Phi}(t,\tau)^T d\tau \\
&= \mathbf{\Phi}(t,t_k)\mathbf{P}_k(t_k)\mathbf{\Phi}(t,t_k)^T \\
&\quad + \int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{Q}(\tau)\mathbf{G}(\tau)^T\mathbf{\Phi}(t,\tau)^T d\tau
\end{aligned}
$$

where the cross-correlations between $\delta\mathbf{x}(\mathbf{t})$ and $\mathbf{w}(t)$ have been cancelled implicitely since they are independent. The fifth step of combining the integrals comes from Itô isometry (see [71], lemma 3.1.5). Furthermore, the final step comes from the definition of the covariance: $\mathbb{E}[\mathbf{w}(t)\mathbf{w}(t)^T] = \mathbb{E}[(\mathbf{w}(t)-\mathbb{E}[\mathbf{w}(t)])(\mathbf{w}(t)-\mathbb{E}[\mathbf{w}(t)])^T] = \mathrm{Cov}[\mathbf{w}(t),\mathbf{w}(t)] = \mathbf{Q}(t)$.

This concludes the necessary components needed to predict the filter. However, to express them in discrete time, a shorthand notation $\mathbf{x}_k^- := \mathbf{x}^-(t_k)$ and $\mathbf{P}_k^- := \mathbf{P}_k^-(t_k)$ is introduced. The prediction step can then be summarised as the following:

$$
\hat{\mathbf{x}}_{k+1}^{-} = \int_{t_k}^{t_{k+1}} \mathbf{f}(\hat{\mathbf{x}}^-(t),\mathbf{u}(t))dt + \hat{\mathbf{x}}_k \qquad (5.37)
$$

$$
\begin{aligned}
\mathbf{P}_{k+1}^{-} &= \mathbf{\Phi}(t_{k+1},t_k)\mathbf{P}_k\mathbf{\Phi}(t_{k+1},t_k)^T \\
&\quad + \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1},\tau)\mathbf{G}(\tau)\mathbf{Q}(\tau)\mathbf{G}(\tau)^T\mathbf{\Phi}(t_{k+1},\tau)^T d\tau
\end{aligned} \qquad (5.38)
$$

5.5.3 Update Step

The measurement prediction function has to be linearised around the current estimate to use the same framework as in the Kalman Filter to find the Kalman gain:

$$\mathbf{h}(\mathbf{x}^-_{k+1}) \approx \mathbf{h}(\hat{\mathbf{x}}^-_{k+1}) + \underbrace{\frac{\partial \mathbf{h}(\mathbf{x}^-_{k+1})}{\partial \mathbf{x}^-_{k+1}}\bigg|_{\mathbf{x}^-_{k+1}=\hat{\mathbf{x}}^-_{k+1}}}_{\mathbf{H}_{k+1}} \delta\mathbf{x}_{k+1} \tag{5.39}$$

yielding that the residual used in the update of the filter must be on the following form:

$$\begin{aligned} \mathbf{z}_{k+1} - \hat{\mathbf{z}}^-_{k+1} &= \mathbf{h}(\mathbf{x}_{k+1}) + \mathbf{v}_{k+1} - \mathbf{h}(\hat{\mathbf{x}}^-_{k+1}) \\ &\approx \cancel{\mathbf{h}(\hat{\mathbf{x}}^-_{k+1})} + \mathbf{H}_{k+1}\delta\mathbf{x}_{k+1} + \mathbf{v}_{k+1} - \cancel{\mathbf{h}(\hat{\mathbf{x}}^-_{k+1})} \\ &\approx \mathbf{H}_{k+1}\delta\mathbf{x}_{k+1} + \mathbf{v}_{k+1} \end{aligned} \tag{5.40}$$

The rest of the steps in the filter update are identical to the Kalman Filter.

5.5.4 Summary

The Extended Kalman Filter can be summarised as the following:

Initialise with initial estimate	$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{P}_0)$
Prediction step	$\hat{\mathbf{x}}^-_{k+1} = \int_{t_k}^{t_{k+1}} \mathbf{f}(\hat{\mathbf{x}}^-(t), \mathbf{u}(t))dt + \hat{\mathbf{x}}_k$ Find the error-state transition matrix $\mathbf{\Phi}(t_{k+1}, t_k)$ $\mathbf{P}^-_{k+1} = \mathbf{\Phi}(t_{k+1}, t_k)\mathbf{P}_k\mathbf{\Phi}(t_{k+1}, t_k)^T$ $+ \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1}, \tau)\mathbf{G}(\tau)\mathbf{Q}(\tau)\mathbf{G}(\tau)^T\mathbf{\Phi}(t_{k+1}, \tau)^T d\tau$
Update step	$\mathbf{H}_{k+1} = \frac{\partial \mathbf{h}(\mathbf{x}^-_{k+1})}{\partial \mathbf{x}^-_{k+1}}\Big\|_{\mathbf{x}^-_{k+1}=\hat{\mathbf{x}}^-_{k+1}}$ $\mathbf{K}_{k+1} = \mathbf{P}^-_{k+1}\mathbf{H}^T_{k+1}(\mathbf{H}_{k+1}\mathbf{P}^-_{k+1}\mathbf{H}^T_{k+1} + \mathbf{R})^{-1}$ $\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}^-_{k+1} + \mathbf{K}_{k+1}(\mathbf{z}_{k+1} - \mathbf{H}_{k+1}\hat{\mathbf{x}}^-_{k+1})$ $\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}^-_{k+1}$

Table 5.2: Summary of the Extended Kalman Filter.

Chapter 6

Multi-State Constraint Kalman Filter

The MSCKF [13] is a VIO algorithm based on a EKF-based backend. It keeps a sliding window of N augmented IMU poses in its state vector, relating a set of feature tracks to these augmented IMU poses which are appended to the state vector at every camera frame. The feature tracks are used to triangulate the features' position in 3D, which is used as constraints on the sliding window of IMU poses.

In the following sections the IMU frame is given by b, the world frame is given by w and the camera frame is given by c. The derivations in this chapter deviate from the traditional MSCKF implementation by utilising the *Hamilton* convention for quaternions, rather than the *Shuster/JPL* convention. The reader is advised to examine [45] for a comparison between the two conventions. Moreover, the IMU state (and the corresponding error-state) deviates from the original representation given in Mourikis and Roumeliotis [13] by including the camera-IMU extrinsics — which was proposed by Li and Mourikis for MSCKF 2.0 [72] — and the time offset between the camera and the IMU measurements.

6.1 State Representation

6.1.1 IMU State Vector

The continuous IMU state vector is given by:

$$\mathbf{x}_{\text{imu}} = \begin{bmatrix} \mathbf{q}_{wb}^T & \mathbf{v}_b^{wT} & \mathbf{p}_b^{wT} & \mathbf{b}_g^{bT} & \mathbf{b}_a^{bT} & \mathbf{q}_{cb}^T & \mathbf{p}_{cb}^{bT} & t_d \end{bmatrix}^T \tag{6.1}$$

where $\mathbf{q}_{wb}$ is the quaternion describing the orientation from the IMU frame to the world frame, $\mathbf{v}_b^w$ and $\mathbf{p}_b^w$ is the velocity and position of the IMU frame expressed in the world frame, $\mathbf{b}_g^b$ and $\mathbf{b}_a^b$ are the biases affecting the gyroscope and accelerometer in the IMU frame, $\mathbf{q}_{cb}$ and $\mathbf{p}_{cb}^b$ are the camera-IMU extrinsics and t_d is the time offset between camera and IMU measurements. The time offset is assumed to be constant. The biases are modelled as Weiner processes, where the increments are driven by

Gaussian noise: $\mathbf{w}_{wa}^{b} \sim \mathcal{N}(\mathbf{0}_{3\times1}, \mathbf{Q}_{wa})$ and $\mathbf{w}_{wg}^{b} \sim \mathcal{N}(\mathbf{0}_{3\times1}, \mathbf{Q}_{wg})$. The corresponding continuous IMU error-state is given by:

$$\delta\mathbf{x}_{\text{imu}} = \begin{bmatrix} \delta\boldsymbol{\theta}_{wb}^{T} & \delta\mathbf{v}_{b}^{wT} & \delta\mathbf{p}_{b}^{wT} & \delta\mathbf{b}_{g}^{bT} & \delta\mathbf{b}_{a}^{bT} & \delta\boldsymbol{\theta}_{cb}^{T} & \delta\mathbf{p}_{cb}^{bT} & \delta t_{d} \end{bmatrix}^{T} \tag{6.2}$$

For the linear translations, linear velocity and biases, standard error definition is utilised: $\delta\mathbf{x} = \mathbf{x} - \hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ is the estimated value. For the quaternions, the quaternion product is utilised: $\mathbf{q} = \delta\mathbf{q} \otimes \hat{\mathbf{q}} \Rightarrow \delta\mathbf{q} = \mathbf{q} \otimes \hat{\mathbf{q}}^{-1}$, where $\delta\mathbf{q}$ represents the **global** angular error. The error-quaternion is given by the following, where $\boldsymbol{\lambda}$ is the axis of rotation, α is the rotation along the axis and where the small angle assumption is used:

$$\delta\mathbf{q} = \begin{bmatrix} \cos(\frac{\alpha}{2}) \\ \boldsymbol{\lambda}\sin(\frac{\alpha}{2}) \end{bmatrix} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix} \tag{6.3}$$

Here $\delta\boldsymbol{\theta} := \boldsymbol{\lambda}\alpha$. Thus, $\delta\boldsymbol{\theta}$ is the minimal representation of the error-quaternion.

6.1.2 Full State Vector

N IMU poses are included in the full state vector at a given time-step k. These poses represent where the features currently tracked in the pipeline were observed spatially, and thus form a sliding window, as previously discussed. Note that in the following equations, the notation $\mathbf{x}_k$ denotes a specific discrete time point, given by $\mathbf{x}(t_k)$.

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{\text{imu}_k}^{T} & \mathbf{q}_{wb_1}^{T} & \mathbf{p}_{b_1}^{wT} & \dots & \mathbf{q}_{wb_N}^{T} & \mathbf{p}_{b_N}^{wT} \end{bmatrix} \tag{6.4}$$

The full error-state vector is thus given by:

$$\delta\mathbf{x}_k = \begin{bmatrix} \delta\mathbf{x}_{\text{imu}_k}^{T} & \delta\boldsymbol{\theta}_{wb_1}^{T} & \delta\mathbf{p}_{b_1}^{wT} & \dots & \delta\boldsymbol{\theta}_{wb_N}^{T} & \delta\mathbf{p}_{b_N}^{wT} \end{bmatrix} \tag{6.5}$$

6.1.3 Covariance Definition

The covariance for the state $\mathbf{x}_k$ at a particular time step k is given by:

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{P}_{II_k} & \mathbf{P}_{IA_k} \\ \mathbf{P}_{IA_k}^{T} & \mathbf{P}_{AA_k} \end{bmatrix} \tag{6.6}$$

Where $\mathbf{P}_{II_k}$ is the 22×22 covariance matrix for the IMU state, $\mathbf{P}_{AA_k}$ is the $6N \times 6N$ covariance matrix for the N IMU augmented poses' state and $\mathbf{P}_{IA_k}$ is the cross-correlation between the IMU state and the augmented IMU poses' state.

6.2 Sensor Models

6.2.1 Accelerometer Sensor Model

The sensor model for the accelerometer is assumed to be populated with Gaussian noise. Furthermore, the model accounts for the bias and the gravity vector, where the bias is assumed to be a Weiner process. The accelerometer measures specific force in the body frame, which includes the opposite normal force caused by gravity.

$$\mathbf{a}_m^b = \mathbf{a}^b - \mathbf{R}(\mathbf{q}_{wb})^T \mathbf{g}^w + \mathbf{b}_a^b + \mathbf{w}_a^b \qquad \mathbf{w}_a^b \sim \mathcal{N}(\mathbf{0}_{3\times 1}, \mathbf{Q}_a) \tag{6.7}$$

Here $\mathbf{a}_m^b$ is the measured acceleration, $\mathbf{a}^b$ is the true acceleration given in IMU frame, $g^w = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}$ is the gravity vector expressed in the world frame and $\mathbf{w}_a^b$ is the noise affecting the measurement. This can be rearranged to:

$$\mathbf{a}^b = \mathbf{a}_m^b + \mathbf{R}(\mathbf{q}_{wb})^T \mathbf{g}^w - \mathbf{b}_a^b - \mathbf{w}_a^b \tag{6.8}$$

which in the world frame becomes:

$$\mathbf{a}^w = \mathbf{R}(\mathbf{q}_{wb})(\mathbf{a}_m^b - \mathbf{b}_a^b - \mathbf{w}_a^b) + \mathbf{g}^w \tag{6.9}$$

6.2.2 Gyroscope Sensor Model

The sensor model for the gyroscope is also assumed to be populated with Gaussian noise and a bias following a Weiner process:

$$\boldsymbol{\omega}_m^b = \boldsymbol{\omega}_{wb}^b + \mathbf{b}_g^b + \mathbf{w}_g^b \qquad \mathbf{w}_g^b \sim \mathcal{N}(\mathbf{0}_{3\times 1}, \mathbf{Q}_g) \tag{6.10}$$

Here $\boldsymbol{\omega}_m^b$ is the measured angular velocity, $\boldsymbol{\omega}_{wb}^b$ is the true angular velocity and $\mathbf{w}_g^b$ is the noise affecting the measurement. This can be rearranged to:

$$\boldsymbol{\omega}_{wb}^b = \boldsymbol{\omega}_m^b - \mathbf{b}_g^b - \mathbf{w}_g^b \tag{6.11}$$

6.3 IMU State Dynamics

The measurement models yield the following continuous dynamics for the IMU state:

$$\dot{\mathbf{q}}_{wb} = \frac{1}{2}\mathbf{q}_{wb} \otimes (\boldsymbol{\omega}_m^b - \mathbf{b}_g^b - \mathbf{w}_g^b) \tag{6.12}$$

$$\dot{\mathbf{v}}_b^w = \mathbf{R}(\mathbf{q}_{wb})(\mathbf{a}_m^b - \mathbf{b}_a^b - \mathbf{w}_a^b) + \mathbf{g}^w \tag{6.13}$$

$$\dot{\mathbf{p}}_b^w = \mathbf{v}_b^w \tag{6.14}$$

$$\dot{\mathbf{b}}_g^b = \mathbf{w}_{w_g}^b \tag{6.15}$$

$$\dot{\mathbf{b}}_a^b = \mathbf{w}_{w_a}^b \tag{6.16}$$

$$\dot{\mathbf{q}}_{cb} = \mathbf{0}_{4\times 1} \tag{6.17}$$

$$\dot{\mathbf{p}}^b_{cb} = \mathbf{0}_{3\times 1} \tag{6.18}$$

$$\dot{t}_d = 0 \tag{6.19}$$

The camera-IMU extrinsics and time offset t_d are kept static in the dynamic model, as stated in [72].

6.3.1 Nominal Dynamics

Applying the expected operator on the IMU state yields the nominal state:

$$\dot{\hat{\mathbf{q}}}_{wb} = \frac{1}{2}\hat{\mathbf{q}}_{wb} \otimes (\boldsymbol{\omega}^b_m - \hat{\mathbf{b}}^b_g) \tag{6.20}$$

$$\dot{\hat{\mathbf{v}}}^w_b = \mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}^b_m - \hat{\mathbf{b}}^b_a) + \mathbf{g}^w \tag{6.21}$$

$$\dot{\hat{\mathbf{p}}}^w_b = \hat{\mathbf{v}}^w_b \tag{6.22}$$

$$\dot{\hat{\mathbf{b}}}^b_g = \mathbf{0}_{3\times 1} \tag{6.23}$$

$$\dot{\hat{\mathbf{b}}}^b_a = \mathbf{0}_{3\times 1} \tag{6.24}$$

$$\dot{\hat{\mathbf{q}}}_{cb} = \mathbf{0}_{4\times 1} \tag{6.25}$$

$$\dot{\hat{\mathbf{p}}}^b_{cb} = \mathbf{0}_{3\times 1} \tag{6.26}$$

$$\dot{\hat{t}}_d = 0 \tag{6.27}$$

6.4 IMU Error-State Dynamics

6.4.1 Angular Error-State Dynamics

The following derivations can also be found in [45], chapter 7.1.2. Let $\hat{\boldsymbol{\omega}}^b_{wb} = \boldsymbol{\omega}^b_m - \hat{\mathbf{b}}^b_g$ and $\delta\boldsymbol{\omega}^b_{wb} = -\delta\mathbf{b}^b_g - \mathbf{w}^b_g$, where $\delta\mathbf{b}^b_g$ is the error between the true bias and the estimated bias. With this, the true angular rate can be written as $\boldsymbol{\omega}^b_{wb} = \hat{\boldsymbol{\omega}}^b_{wb} + \delta\boldsymbol{\omega}^b_{wb}$. Furthermore, $\dot{\mathbf{q}}_{wb}$ can be expressed in two ways:

$$\begin{aligned}
\frac{d}{dt}(\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}) &= \frac{1}{2}\mathbf{q}_{wb}\otimes\boldsymbol{\omega}^b_{wb} \\
\Rightarrow \delta\dot{\mathbf{q}}_{wb}\otimes\hat{\mathbf{q}}_{wb} + \delta\mathbf{q}_{wb}\otimes\dot{\hat{\mathbf{q}}}_{wb} &= \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\boldsymbol{\omega}^b_{wb} \\
\Rightarrow \delta\dot{\mathbf{q}}_{wb}\otimes\hat{\mathbf{q}}_{wb} + \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\hat{\boldsymbol{\omega}}^b_{wb} &= \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\boldsymbol{\omega}^b_{wb}
\end{aligned} \tag{6.28}$$

By utilising that $\boldsymbol{\omega}^b_{wb} = \hat{\boldsymbol{\omega}}^b_{wb} + \delta\boldsymbol{\omega}^b_{wb}$, such that $\frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\boldsymbol{\omega}^b_{wb} - \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\hat{\boldsymbol{\omega}}^b_{wb} = \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\delta\boldsymbol{\omega}^b_{wb}$, this results in:

$$\delta\dot{\mathbf{q}}_{wb}\otimes\hat{\mathbf{q}}_{wb} = \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\delta\boldsymbol{\omega}^b_{wb} \tag{6.29}$$

Right-multiplying by $\hat{\mathbf{q}}^{-1}_{wb}$ yields:

$$\begin{aligned}
\delta\dot{\mathbf{q}}_{wb} &= \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\hat{\mathbf{q}}_{wb}\otimes\delta\boldsymbol{\omega}^b_{wb}\otimes\hat{\mathbf{q}}^{-1}_{wb} \\
&= \frac{1}{2}\delta\mathbf{q}_{wb}\otimes\delta\boldsymbol{\omega}^w_{wb}
\end{aligned} \tag{6.30}$$

Expanding yields (utilising equation (2.13)):

$$\begin{aligned}
\begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\theta}}_{wb} \end{bmatrix} &= 2\delta\dot{\mathbf{q}}_{wb} \\
&= \delta\mathbf{q}_{wb}\otimes\delta\boldsymbol{\omega}^w_{wb} \\
&= \boldsymbol{\Omega}(\delta\boldsymbol{\omega}^w_{wb})\delta\mathbf{q}_{wb} \\
&= \begin{bmatrix} 0 & -\delta\boldsymbol{\omega}^{wT}_{wb} \\ \delta\boldsymbol{\omega}^w_{wb} & -[\delta\boldsymbol{\omega}^w_{wb}]_\times \end{bmatrix}\begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\theta}_{wb}}{2} \end{bmatrix} + O(||\delta\boldsymbol{\theta}_{wb}||^2)
\end{aligned} \tag{6.31}$$

This results in a scalar and vector representation:

$$0 = -\delta\boldsymbol{\omega}^{wT}_{wb}\frac{\delta\boldsymbol{\theta}_{wb}}{2} + O(||\delta\boldsymbol{\theta}_{wb}||^2) \tag{6.32}$$

$$\delta\dot{\boldsymbol{\theta}}_{wb} = \delta\boldsymbol{\omega}^w_{wb} - \frac{1}{2}[\delta\boldsymbol{\omega}^w_{wb}]_\times\delta\boldsymbol{\theta}_{wb} + O(||\delta\boldsymbol{\theta}_{wb}||^2) \tag{6.33}$$

Neglecting the higher order terms and assuming $[\delta\boldsymbol{\omega}^w_{wb}]_\times\delta\boldsymbol{\theta}_{wb} \approx \mathbf{0}$ results in the following, where $\delta\boldsymbol{\omega}^b_{wb} = -\delta\mathbf{b}^b_g - \mathbf{w}^b_g$ has been utilised:

$$\begin{aligned}
\delta\dot{\boldsymbol{\theta}}_{wb} &\approx \delta\boldsymbol{\omega}^w_{wb} \\
&\approx \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\boldsymbol{\omega}^b_{wb} \\
&\approx -\mathbf{R}(\hat{\mathbf{q}}_{wb})(\delta\mathbf{b}^b_g - \mathbf{w}^b_g) \\
&\approx -\mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{b}^b_g - \mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{w}^b_g
\end{aligned} \tag{6.34}$$

6.4.2 Linear Velocity Error-State Dynamics

The following derivations can also be found in [45], chapter 7.1.1. As stated in equation (6.9), the true acceleration is given by:

$$\mathbf{a}^w = \mathbf{R}(\mathbf{q}_{wb})(\mathbf{a}_B^b + \delta\mathbf{a}_B^b) + \mathbf{g}^w \tag{6.35}$$

Where $\mathbf{a}_B^b := \mathbf{a}_m^b - \hat{\mathbf{b}}_a^b$ and $\delta\mathbf{a}_B^b := -\delta\mathbf{b}_a^b - \mathbf{w}_a^b$ such that $\mathbf{a}_B^b + \delta\mathbf{a}_B^b = \mathbf{a}_m^b - \mathbf{b}_a^b - \mathbf{w}_a^b$. Furthermore, for convenience in the derivations, let the true rotation matrix be denoted by: $\mathbf{R}(\mathbf{q}_{wb}) = (\mathbf{I} + [\delta\boldsymbol{\theta}_{wb}]_\times)\mathbf{R}(\hat{\mathbf{q}}_{wb}) + O(||\delta\boldsymbol{\theta}_{wb}||^2)$, where the higher order terms are neglected in the following derivations. This yields two representations for $\dot{\mathbf{v}}$:

$$\begin{aligned}
\dot{\hat{\mathbf{v}}}_b^w + \delta\dot{\mathbf{v}}_b^w &\approx (\mathbf{I} + [\delta\boldsymbol{\theta}_{wb}]_\times)\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_B^b + \delta\mathbf{a}_B^b) + \mathbf{g}^w \\
\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b + \cancel{\mathbf{g}^w} + \delta\dot{\mathbf{v}}_b^w &\approx (\mathbf{I} + [\delta\boldsymbol{\theta}_{wb}]_\times)\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_B^b + \delta\mathbf{a}_B^b) + \cancel{\mathbf{g}^w} \\
\cancel{\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b} + \delta\dot{\mathbf{v}}_b^w &\approx \cancel{\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b} + \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{a}_B^b + [\delta\boldsymbol{\theta}_{wb}]_\times\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b + [\delta\boldsymbol{\theta}_{wb}]_\times\mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{a}_B^b \\
\delta\dot{\mathbf{v}}_b^w &\approx \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{a}_B^b + [\delta\boldsymbol{\theta}_{wb}]_\times\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_B^b + \delta\mathbf{a}_B^b)
\end{aligned}$$

Assuming $[\delta\boldsymbol{\theta}_{wb}]_\times\mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{a}_B^b \approx \mathbf{0}$ and applying $[\delta\boldsymbol{\theta}_{wb}]_\times\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b = -[\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b]_\times\delta\boldsymbol{\theta}_{wb}$ results in:

$$\delta\dot{\mathbf{v}}_b^w \approx \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{a}_B^b - [\mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{a}_B^b]_\times\delta\boldsymbol{\theta}_{wb} \tag{6.36}$$

which fully expanded becomes:

$$\delta\dot{\mathbf{v}}_b^w \approx -[\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_m^b - \hat{\mathbf{b}}_a^b)]_\times\delta\boldsymbol{\theta}_{wb} - \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{b}_a^b - \mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{w}_a^b \tag{6.37}$$

6.4.3 Bias Error-State Dynamics

For gyroscope and accelerometer bias, the error-state derivative is given by:

$$\delta\dot{\mathbf{b}}_g^b = \dot{\mathbf{b}}_g^b - \dot{\hat{\mathbf{b}}}_g^b = \mathbf{w}_{w_g}^b - \mathbf{0}_{3x1} = \mathbf{w}_{w_g}^b \tag{6.38}$$

$$\delta\dot{\mathbf{b}}_a^b = \dot{\mathbf{b}}_a^b - \dot{\hat{\mathbf{b}}}_a^b = \mathbf{w}_{w_a}^b - \mathbf{0}_{3x1} = \mathbf{w}_{w_a}^b \tag{6.39}$$

6.4.4 IMU Error-State Dynamics

From the previous derivations, the full error-state dynamics for the IMU state is given by:

$$\delta\dot{\boldsymbol{\theta}}_{wb} \approx -\mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{b}_g^b - \mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{w}_g^b \tag{6.40}$$

$$\delta\dot{\mathbf{v}}_b^w \approx -[\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_m^b - \hat{\mathbf{b}}_a^b)]_\times\delta\boldsymbol{\theta}_{wb} - \mathbf{R}(\hat{\mathbf{q}}_{wb})\delta\mathbf{b}_a^b - \mathbf{R}(\hat{\mathbf{q}}_{wb})\mathbf{w}_a^b \tag{6.41}$$

$$\delta\dot{\mathbf{p}}_b^w = \delta\mathbf{v}_b^w \tag{6.42}$$

$$\delta\dot{\mathbf{b}}_g^b = \mathbf{w}_{w_g}^b \tag{6.43}$$

$$\delta\dot{\mathbf{b}}_a^b = \mathbf{w}_{w_a}^b \tag{6.44}$$

$$\delta\dot{\boldsymbol{\theta}}_{cb} = \mathbf{0}_{3x1} \tag{6.45}$$

$$\delta\dot{\mathbf{p}}_{cb}^b = \mathbf{0}_{3x1} \tag{6.46}$$

$$\delta\dot{t}_d = 0 \tag{6.47}$$

This yields that the continuous error-state dynamics can be written on the following form (note that previously, the dependence on time has been omitted to save space in the notation):

$$\delta\dot{\mathbf{x}}_{\text{imu}} = \mathbf{F}(t)\delta\mathbf{x}_{\text{imu}} + \mathbf{G}(t)\mathbf{w}_{\text{imu}}(t) \tag{6.48}$$

where $\mathbf{w}_{\text{imu}}(t) = \begin{bmatrix}\mathbf{w}_g^{bT}(t) & \mathbf{w}_a^{bT}(t) & \mathbf{w}_{w_g}^{bT}(t) & \mathbf{w}_{w_a}^{bT}(t)\end{bmatrix}^T$. $\mathbf{F}(t)$ and $\mathbf{G}(t)$ are given by:

$$\mathbf{F}(t) = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{R}(\hat{\mathbf{q}}_{wb}) & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ -[\mathbf{R}(\hat{\mathbf{q}}_{wb})(\mathbf{a}_m^b - \hat{\mathbf{b}}_a^b)]_\times & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{R}(\hat{\mathbf{q}}_{wb}) & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times7} \end{bmatrix} \tag{6.49}$$

$$\mathbf{G}(t) = \begin{bmatrix} -\mathbf{R}(\hat{\mathbf{q}}_{wb}) & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -\mathbf{R}(\hat{\mathbf{q}}_{wb}) & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} \end{bmatrix} \tag{6.50}$$

6.5 Prediction Step

6.5.1 Predicting the Nominal State

To find an a priori estimate $\mathbf{x}^{-}_{\text{imu}_{k+1}}$, numerical integration is used (in particular Runga-Kutta, see Appendix B). The augmented IMU states are assumed to be static and not propagated.

6.5.2 Predicting the Covariance

Following the derivations made in section 6.4.4, the error-state dynamics are now on a form which can be used for finding the transition matrix $\mathbf{\Phi}(t_{k+1}, t_k)$ and predicting the covariance. The reader should be aware that this is in fact a time-varying system, due to the noise affecting the measurements. Thus, one has no basis for stating that a time-shift of the dynamical system will represent the same system given an identical state, as the noise cannot be assumed to hold an identical value. In other words $\mathbf{F}(t_k) \neq \mathbf{F}(t_{k+1})$ given identical estimates, since noise is affecting $\mathbf{a}^b_m$. A representation on the following form has to be found (equation (5.32)):

$$\delta\mathbf{x}_{\text{imu}_{k+1}} = \mathbf{\Phi}(t_{k+1}, t_k)\delta\mathbf{x}_{\text{imu}_k} + \int_{t_k}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{w}_{\text{imu}}(\tau)d\tau \tag{6.51}$$

Note that in the system, this equation is not utilised directly. This is simply due to the fact that there is not a need to evaluate it explicitly; it is only required to find the transition matrix which makes this realisation true so that the a priori covariance matrix can be predicted. The transition matrix for this particular system is found to be the following, as shown in [2]:

$$\mathbf{\Phi}(t_{k+1}, t_k) = \begin{bmatrix}
\mathbf{I}_{3\times3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{\Phi}_{\theta b_g} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{\Phi}_{v\theta} & \mathbf{I}_{3x3} & \mathbf{0}_{3x3} & \mathbf{\Phi}_{vb_g} & \mathbf{\Phi}_{vb_a} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{\Phi}_{p\theta} & \Delta t\mathbf{I}_{3x3} & \mathbf{I}_{3\times3} & \mathbf{\Phi}_{pb_g} & \mathbf{\Phi}_{pb_a} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} & \mathbf{0}_{3\times3} & 0 \\
\mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{1}_{3\times3} & 0 \\
\mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & \mathbf{0}_{1\times3} & 1
\end{bmatrix} \tag{6.52}$$

where (Δt denotes the time between t_k and t_{k+1}):

$$\boldsymbol{\Phi}_{\theta b_g} = -\mathbf{R}(\hat{\mathbf{q}}_{wb})\Big(\Delta t\mathbf{I}_{3\times 3} + \frac{1}{2}\Delta t[\boldsymbol{\theta}]_\times\Big) \tag{6.53}$$

$$\boldsymbol{\Phi}_{v\theta} = -\Big[\hat{\mathbf{v}}_b^{w^-} - \hat{\mathbf{v}}_b^{w} - \mathbf{g}^w\Delta t\Big]_\times \tag{6.54}$$

$$\boldsymbol{\Phi}_{vb_g} = \Big[-\hat{\mathbf{p}}_b^{w^-} + \hat{\mathbf{p}}_b^{w} + \hat{\mathbf{v}}_b^{w^-}\Delta t - \frac{1}{2}\mathbf{g}^w\Delta t^2\Big]_\times \mathbf{R}(\hat{\mathbf{q}}_{wb}) \tag{6.55}$$

$$+\Big[-\frac{1}{2}\hat{\mathbf{p}}_b^{w^-} + \frac{1}{2}\hat{\mathbf{p}}_b^{w} + \frac{1}{2}\hat{\mathbf{v}}_b^{w^-}\Delta t - \frac{1}{6}\mathbf{g}^w\Delta t^2\Big]_\times \mathbf{R}(\hat{\mathbf{q}}_{wb})[\boldsymbol{\theta}]_\times \tag{6.56}$$

$$\boldsymbol{\Phi}_{vb_a} = -\mathbf{R}(\hat{\mathbf{q}}_{wb})\Big(\Delta t\mathbf{I}_{3\times 3} + \frac{1}{2}\Delta t[\boldsymbol{\theta}]_\times\Big) \tag{6.57}$$

$$\boldsymbol{\Phi}_{p\theta} = -\Big[\hat{\mathbf{p}}_b^{w^-} - \hat{\mathbf{p}}_b^{w} - \hat{\mathbf{v}}_b^{w}\Delta t - \frac{1}{2}\mathbf{g}^w\Delta t^2\Big] \tag{6.58}$$

$$\boldsymbol{\Phi}_{pb_g} = \Big[-\frac{1}{6}\mathbf{g}^w\Delta t^3\Big]_\times \mathbf{R}(\hat{\mathbf{q}}_{wb}) \tag{6.59}$$

$$+\Big[\frac{1}{4}\hat{\mathbf{p}}_b^{w^-}\Delta t - \frac{1}{4}\hat{\mathbf{p}}_b^{w}\Delta t - \frac{1}{24}\mathbf{g}^w\Delta t^3\Big]_\times \mathbf{R}(\hat{\mathbf{q}}_{wb}[\boldsymbol{\theta}]_\times \tag{6.60}$$

$$\boldsymbol{\Phi}_{pb_a} = -\frac{1}{6}\mathbf{R}(\hat{\mathbf{q}}_{wb})\Delta t^2\Big(3\mathbf{I}_{3\times 3} + [\boldsymbol{\theta}]_\times\Big) \tag{6.61}$$

and where $\boldsymbol{\theta}$ is given by the following:

$$\boldsymbol{\theta} = \frac{1}{2}\Big(\hat{\boldsymbol{\omega}}_{wb}^b(t_{t_k}) + \hat{\boldsymbol{\omega}}_{wb}^b(t_{k+1})\Big)\Delta t + \frac{1}{12}\Big(\hat{\boldsymbol{\omega}}_{wb}^b(t_k) \times \hat{\boldsymbol{\omega}}_{wb}^b(t_{k+1})\Big)\Delta t^2 \tag{6.62}$$

The covariance for the IMU state can then be propagated according to equation (5.38), where an approximation for the integral is made:

$$\begin{aligned}
\mathbf{P}_{II_{k+1}}^- &= \boldsymbol{\Phi}(t_{k+1}, t_k)\mathbf{P}_{II_k}\boldsymbol{\Phi}(t_{k+1}, t_k)^T \\
&\quad + \int_{t_k}^{t_{k+1}} \boldsymbol{\Phi}(t_{k+1}, \tau)\mathbf{G}(\tau)\mathbf{Q}(\tau)\mathbf{G}(\tau)^T\boldsymbol{\Phi}(t_{k+1}, \tau)^T d\tau \\
&\approx \boldsymbol{\Phi}(t_{k+1}, t_k)\mathbf{P}_{II_k}\boldsymbol{\Phi}(t_{k+1}, t_k)^T \\
&\quad + \boldsymbol{\Phi}(t_{k+1}, t_k)\mathbf{G}(t_k)\mathbf{Q}(t_k)\mathbf{G}(t_k)^T\boldsymbol{\Phi}(t_{k+1}, t_k)^T\Delta t
\end{aligned} \tag{6.63}$$

The whole a priori covariance matrix is thus propagated according to the following [13]:

$$\mathbf{P}_{k+1}^- = \begin{bmatrix} \mathbf{P}_{II_{k+1}}^- & \boldsymbol{\Phi}(t_{k+1}, t_k)\mathbf{P}_{IA_k} \\ \mathbf{P}_{AI_k}\boldsymbol{\Phi}(t_{k+1}, t_k)^T & \mathbf{P}_{AA_k} \end{bmatrix} \tag{6.64}$$

where one can notice that the cross-correlation is affected by the transition matrix, but the variance for the augmented IMU states is kept static.

6.6 State Augmentation

The full state vector is augmented with the current pose when a new image is captured. The augmented pose is constructed from the current IMU state according to:

$$\hat{\mathbf{q}}_{wb_k} = \hat{\mathbf{q}}_{wb} \qquad \hat{\mathbf{p}}^w_{b_k} = \hat{\mathbf{p}}^w_b \tag{6.65}$$

Furthermore, the covariance matrix is augmented according to:

$$\mathbf{P}_k \leftarrow \begin{bmatrix} \mathbf{I}_{(22+6N)\times(22+6N)} \\ \mathbf{J} \end{bmatrix} \mathbf{P}_k \begin{bmatrix} \mathbf{I}_{(22+6N)\times(22+6N)} \\ \mathbf{J} \end{bmatrix}^T \tag{6.66}$$

The jacobian $\mathbf{J}$ is given by the partial derivative of equation (6.65) with respect to the estimated nominal state vector $\hat{\mathbf{x}}$:

$$\begin{aligned} \mathbf{J} :&= \frac{\partial \begin{bmatrix} \hat{\mathbf{q}}^T_{wb_k} & \hat{\mathbf{p}}^{w^T}_{b_k} \end{bmatrix}^T}{\partial \hat{\mathbf{x}}} \\ &= \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times13} & \mathbf{0}_{3\times6N} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times13} & \mathbf{0}_{3\times6N} \end{bmatrix} \end{aligned} \tag{6.67}$$

6.7 Measurement Model

For the Extended Kalman Filter update, the measurement model should be on the following form (as derived in equation (5.40)):

$$\mathbf{r} = \mathbf{H}\delta\mathbf{x} + \mathbf{v} \tag{6.68}$$

Where $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise and $\mathbf{H}$ is the measurement jacobian matrix. To derive this expression, let f_j represent a particular feature seen in N_j augmented IMU frames, and let the total amount of features be M. Furthermore, let the camera frame c_k be related to the given augmented IMU frame b_k at time step k by the camera-IMU extrinsics, as shown in equation (6.69). Note that the camera-IMU extrinsics are tied to a given time step k, as they are estimated in the filter.

$$\mathbf{q}_{wc_k} = \mathbf{q}_{wb_k} \otimes \mathbf{q}^{-1}_{c_kb_k} \qquad \mathbf{p}^w_{c_k} = \mathbf{p}^w_{b_k} + \mathbf{R}(\mathbf{q}_{wb_k})\mathbf{p}^{b_k}_{c_kb_k} \tag{6.69}$$

Let the measurement be given by the de-homogenised 3D coordinates of the feature, $\mathbf{p}^w_{f_j}$, in the camera frame c_k:

$$\mathbf{z}_{j,k} = \frac{1}{Z^{c_k}_j} \begin{bmatrix} X^{c_k}_j \\ Y^{c_k}_j \end{bmatrix} + \mathbf{v}_{j,k} \tag{6.70}$$

$$\begin{aligned} \begin{bmatrix} X^{c_k}_j \\ Y^{c_k}_j \\ Z^{c_k}_j \end{bmatrix} &= \mathbf{R}(\mathbf{q}_{c_kb_k})\mathbf{R}(\mathbf{q}_{wb_k})^T(\mathbf{p}^w_{f_j} - \mathbf{p}^w_{c_k}) \\ &= \mathbf{R}(\mathbf{q}_{c_kb_k})\mathbf{R}(\mathbf{q}_{wb_k})^T(\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k} - \mathbf{R}(\mathbf{q}_{wb_k})\mathbf{p}^{b_k}_{c_kb_k}) \\ &= \mathbf{R}(\mathbf{q}_{c_kb_k})\mathbf{R}(\mathbf{q}_{wb_k})^T(\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k}) - \mathbf{R}(\mathbf{q}_{c_kb_k})\mathbf{p}^{b_k}_{c_kb_k} \end{aligned} \tag{6.71}$$

The index j represents the particular feature and the index k represents the camera frame c_k where it was seen. It is assumed that the noise follows a Gaussian distribution: $\mathbf{v}_{j,k} \sim \mathcal{N}([0,0]^T, \sigma^2_{\text{im}}\mathbf{I}_{2\times2})$. This leads to the following residual:

$$\mathbf{r}_{j,k}(\mathbf{z}_{j,k}) = \mathbf{z}_{j,k} - \hat{\mathbf{z}}_{j,k} \tag{6.72}$$

6.7.1 Residual Linearisation

Let $\mathbf{z}_{j,k} = \hat{\mathbf{z}}_{j,k} + \delta\mathbf{z}_{j,k}$. Furthermore $\mathbf{r}_{j,k}(\hat{\mathbf{z}}_{j,k}) = \mathbf{0}$ by definition. Linearising the residual with respect to the estimated measurement evaluated at the state estimate and the estimate for the feature position yields:

$$\begin{aligned}
\mathbf{r}_{j,k}(\hat{\mathbf{z}}_{j,k} + \delta\mathbf{z}_{j,k}) &\approx \cancel{\mathbf{r}_{j,k}(\hat{\mathbf{z}}_{j,k})} + \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{x}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \delta\mathbf{x} + \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{p}^w_{f_j}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \delta\mathbf{p}^w_{f_j} \\
&\approx \underbrace{\frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{x}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}}}_{\mathbf{H}_{\hat{\mathbf{x}}_{j,k}}} \delta\mathbf{x} + \underbrace{\frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{i,j})}{\partial \mathbf{p}^w_{f_j}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}}}_{\mathbf{H}_{\hat{\mathbf{p}}_{f_{j,k}}}} \delta\mathbf{p}^w_{f_j}
\end{aligned} \tag{6.73}$$

For convenience in the derivations, the jacobian $\mathbf{H}_{\hat{\mathbf{x}}_{j,k}}$ is partitioned into $\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{wb_k}}$, $\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^w_{b_k}}$, $\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{c_k b_k}}$ and $\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^{c_k}_{c_k b_k}}$ which are with respect to the augmented IMU state orientation, augmented IMU state position and orientation extrinsics and translation extrinsics, respectively:

$$\begin{aligned}
\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{wb_k}} &= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{q}_{wb_k}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{z}_{j,k}} \frac{\partial \mathbf{z}_{j,k}}{\partial \begin{bmatrix} X_j^{c_k} & Y_j^{c_k} & Z_j^{c_k}\end{bmatrix}^T} \frac{\begin{bmatrix} X_j^{c_k} & Y_j^{c_k} & Z_j^{c_k}\end{bmatrix}^T}{\partial \mathbf{q}_{wb_k}}\Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \mathbf{I}_{2x2} \begin{bmatrix} \frac{1}{Z_j^{c_k}} & 0 & -\frac{X_j^{c_k}}{(Z_j^{c_k})^2} \\ 0 & \frac{1}{Z_j^{c_k}} & -\frac{Y_j^{c_k}}{(Z_j^{c_k})^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{wb_k})^T [\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k}]_\times \Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{Z_j^{c_k}} & 0 & -\frac{X_j^{c_k}}{(Z_j^{c_k})^2} \\ 0 & \frac{1}{Z_j^{c_k}} & -\frac{Y_j^{c_k}}{(Z_j^{c_k})^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{wb_k})^T [\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k}]_\times \Bigg|_{\substack{\mathbf{p}_j^{c_k} = \hat{\mathbf{p}}_j^{c_k} \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{\hat{Z}_j^{c_k}} & 0 & -\frac{\hat{X}_j^{c_k}}{(\hat{Z}_j^{c_k})^2} \\ 0 & \frac{1}{\hat{Z}_j^{c_k}} & -\frac{\hat{Y}_j^{c_k}}{(\hat{Z}_j^{c_k})^2} \end{bmatrix} \mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})\mathbf{R}(\hat{\mathbf{q}}_{wb_k})^T [\hat{\mathbf{p}}^w_{f_j} - \hat{\mathbf{p}}^w_{b_k}]_\times
\end{aligned} \tag{6.74}$$

The derivation for the jacobian of a given product $\mathbf{R}(\mathbf{q})^T\mathbf{v}$ with respect to $\mathbf{q}$ can be found in equation (A.2), Appendix A.

$$\begin{aligned}
\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^w_{b_k}} &= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{p}^w_{b_k}}\bigg|_{\substack{\mathbf{z}_{j,k}=\hat{\mathbf{z}}_{j,k}\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{z}_{j,k}} \frac{\partial \mathbf{z}_{j,k}}{\partial \begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T} \frac{\begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T}{\partial \mathbf{p}^w_{b_k}}\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\mathbf{I}_{2x2}\begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\begin{bmatrix} \frac{1}{\hat{Z}^{c_k}_j} & 0 & -\frac{\hat{X}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \\ 0 & \frac{1}{\hat{Z}^{c_k}_j} & -\frac{\hat{Y}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})\mathbf{R}(\hat{\mathbf{q}}_{w b_k})^T
\end{aligned} \tag{6.75}$$

$$\begin{aligned}
\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{c_k b_k}} &= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{q}_{c_k b_k}}\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{z}_{j,k}} \frac{\partial \mathbf{z}_{j,k}}{\partial \begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T} \frac{\begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T}{\partial \mathbf{q}_{c_k b_k}}\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \mathbf{I}_{2x2}\begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} (-[\mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T(\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k})]_\times + \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{p}^{b_k}_{c_k b_k})\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} (-[\mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T(\mathbf{p}^w_{f_j} - \mathbf{p}^w_{b_k})]_\times + [\mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{p}^{b_k}_{c_k b_k}]_\times)\bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{\hat{Z}^{c_k}_j} & 0 & -\frac{\hat{X}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \\ 0 & \frac{1}{\hat{Z}^{c_k}_j} & -\frac{\hat{Y}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \end{bmatrix} (-[\mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})\mathbf{R}(\hat{\mathbf{q}}_{w b_k})^T(\hat{\mathbf{p}}^w_{f_j} - \hat{\mathbf{p}}^w_{b_k})]_\times + [\mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})\hat{\mathbf{p}}^{b_k}_{c_k b_k}]_\times)
\end{aligned} \tag{6.76}$$

The derivation for the jacobian of a given product $\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a$ with respect to $\mathbf{q}_{cb}$ and the jacobian of $\mathbf{R}(\mathbf{q})\mathbf{v}$ with respect to $\mathbf{q}$ can be found in equations (A.1) and (A.3), Appendix A.

$$
\begin{aligned}
\mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^{b_k}_{c_k b_k}} &= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{p}^{b_k}_{c_k b_k}}\Bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{z}_{j,k}} \frac{\partial \mathbf{z}_{j,k}}{\partial \begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T} \frac{\begin{bmatrix} X^{c_k}_j & Y^{c_k}_j & Z^{c_k}_j \end{bmatrix}^T}{\partial \mathbf{p}^{b_k}_{c_k b_k}}\Bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\mathbf{I}_{2x2} \begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\Bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\begin{bmatrix} \frac{1}{Z^{c_k}_j} & 0 & -\frac{X^{c_k}_j}{(Z^{c_k}_j)^2} \\ 0 & \frac{1}{Z^{c_k}_j} & -\frac{Y^{c_k}_j}{(Z^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\Bigg|_{\substack{\mathbf{p}^{c_k}_j=\hat{\mathbf{p}}^{c_k}_j \\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= -\begin{bmatrix} \frac{1}{\hat{Z}^{c_k}_j} & 0 & -\frac{\hat{X}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \\ 0 & \frac{1}{\hat{Z}^{c_k}_j} & -\frac{\hat{Y}^{c_k}_j}{(\hat{Z}^{c_k}_j)^2} \end{bmatrix} \mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})
\end{aligned}
\tag{6.77}
$$

In summary, the jacobian $\mathbf{H}_{\hat{\mathbf{x}}_{j,k}}$ is then given by:

$$
\mathbf{H}_{\hat{\mathbf{x}}_{j,k}} = \begin{bmatrix} \mathbf{0}_{2\times 15} & \mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{c_k b_k}} & \mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^{b_k}_{c_k b_k}} & \mathbf{0}_{2\times 1} & \cdots & \mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{q}}_{w b_k}} & \mathbf{H}_{\hat{\mathbf{x}}_{j,k},\hat{\mathbf{p}}^{w}_{b_k}} & \cdots \end{bmatrix}
$$

where the elements besides the jacobian with respect to the augmented IMU state are filled with zeros and are dependent on how many augmented states there currently are in the state vector. The jacobian with respect to the feature position is given by equation (6.78).

$$
\begin{aligned}
\mathbf{H}_{\hat{\mathbf{p}}^{w}_{f_{j,k}}} &= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{p}_{f_j}}\Bigg|_{\substack{\mathbf{p}_j^{c_k}=\hat{\mathbf{p}}_j^{c_k}\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \frac{\partial \mathbf{r}_{j,k}(\mathbf{z}_{j,k})}{\partial \mathbf{z}_{j,k}} \frac{\partial \mathbf{z}_{j,k}}{\partial \begin{bmatrix} X_j^{c_k} & Y_j^{c_k} & Z_j^{c_k}\end{bmatrix}^T} \frac{\begin{bmatrix} X_j^{c_k} & Y_j^{c_k} & Z_j^{c_k}\end{bmatrix}^T}{\partial \mathbf{p}^{w}_{f_j}}\Bigg|_{\substack{\mathbf{p}_j^{c_k}=\hat{\mathbf{p}}_j^{c_k}\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \mathbf{I}_{2x2}\begin{bmatrix} \frac{1}{Z_j^{c_k}} & 0 & -\frac{X_j^{c_k}}{(Z_j^{c_k})^2} \\ 0 & \frac{1}{Z_j^{c_k}} & -\frac{Y_j^{c_k}}{(Z_j^{c_k})^2}\end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T\Bigg|_{\substack{\mathbf{p}_j^{c_k}=\hat{\mathbf{p}}_j^{c_k}\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{Z_j^{c_k}} & 0 & -\frac{X_j^{c_k}}{(Z_j^{c_k})^2} \\ 0 & \frac{1}{Z_j^{c_k}} & -\frac{Y_j^{c_k}}{(Z_j^{c_k})^2}\end{bmatrix} \mathbf{R}(\mathbf{q}_{c_k b_k})\mathbf{R}(\mathbf{q}_{w b_k})^T\Bigg|_{\substack{\mathbf{p}_j^{c_k}=\hat{\mathbf{p}}_j^{c_k}\\ \mathbf{x}=\hat{\mathbf{x}}_k}} \\
&= \begin{bmatrix} \frac{1}{\hat{Z}_j^{c_k}} & 0 & -\frac{\hat{X}_j^{c_k}}{(\hat{Z}_j^{c_k})^2} \\ 0 & \frac{1}{\hat{Z}_j^{c_k}} & -\frac{\hat{Y}_j^{c_k}}{(\hat{Z}_j^{c_k})^2}\end{bmatrix} \mathbf{R}(\hat{\mathbf{q}}_{c_k b_k})\mathbf{R}(\hat{\mathbf{q}}_{w b_k})^T
\end{aligned}
\tag{6.78}
$$

When the jacobians are stacked for all observations and features, the residual becomes:

$$
\mathbf{r} \approx \mathbf{H}_{\hat{\mathbf{x}}}\delta\mathbf{x} + \mathbf{H}_{\hat{\mathbf{p}}_f}\begin{bmatrix}\delta\mathbf{p}^{w}_{f_0} \\ \vdots \\ \delta\mathbf{p}^{w}_{f_M}\end{bmatrix} + \mathbf{v} \tag{6.79}
$$

6.7.2 Finding Estimates for the Feature Positions

The jacobians require estimates for the feature positions $\mathbf{p}_j^{c_k}$ in the respective camera frames (and also $\mathbf{p}_j^{w}$, which can be found by transforming the estimate of $\mathbf{p}_j^{c_k}$ by the estimated transformation between the camera frames and the world frame). These estimates are found by means of triangulation, following the method and derivations outlined in section 4.3.

6.7.3 Null-Space Projection

From the estimate of the feature positions $\hat{\mathbf{p}}^{w}_{f_0}, ..., \hat{\mathbf{p}}^{w}_{f_M}$, the jacobians in the residuals can be numerically evaluated with the use of the current state estimate:

$$
\mathbf{r} \approx \mathbf{H}_{\hat{\mathbf{x}}}\delta\mathbf{x} + \mathbf{H}_{\hat{\mathbf{p}}_f}\begin{bmatrix}\delta\mathbf{p}^{w}_{f_0} \\ \vdots \\ \delta\mathbf{p}^{w}_{f_M}\end{bmatrix} + \mathbf{v} \tag{6.80}
$$

As $\delta\mathbf{p}^w_{f_j}$ and $\delta\mathbf{x}$ are correlated (equation (4.10)), another residual is defined based on projecting to the null-space of $\mathbf{H}_{\hat{\mathbf{p}}_f}$. The projection removes the correlation by effectively cancelling the terms related to $\delta\mathbf{p}^w_{f_j}$, which is motivated by the desire of having zero error in the feature position. Restricting to the null space of the matrix is an effective way to limit the solution to the parameters which yield this configuration.

Let $\mathbf{A}^T$ be a unitary matrix of the left null-space of $\mathbf{H}_{\hat{\mathbf{p}}_f}$, composed from the singular value decomposition of the matrix and the rightmost columns of $\mathbf{U}$ from the decomposition (see [73], chapter 7.4, example 6). This yields:

$$\begin{aligned}
\mathbf{r}_n &= \mathbf{A}^T\mathbf{H}_{\hat{\mathbf{x}}}\delta\mathbf{x} + \cancel{\mathbf{A}^T\mathbf{H}_{\hat{\mathbf{p}}_f}\begin{bmatrix}\delta\mathbf{p}^w_{f_0}\\ \vdots \\ \delta\mathbf{p}^w_{f_M}\end{bmatrix}} + \mathbf{A}^T\mathbf{v} \\
&= \mathbf{A}^T\mathbf{H}_{\hat{\mathbf{x}}}\delta\mathbf{x} + \mathbf{A}^T\mathbf{v} \\
&= \mathbf{H}_{\hat{\mathbf{x}},n}\delta\mathbf{x} + \mathbf{v}_n
\end{aligned} \tag{6.81}$$

Here $\mathbf{H}_{\hat{\mathbf{x}},n} := \mathbf{A}^T\mathbf{H}_{\hat{\mathbf{x}}}$ and $\mathbf{v}_n := \mathbf{A}^T\mathbf{v}$. The matrix $\mathbf{H}_{\hat{\mathbf{p}}_f}$ is of dimension $2M \times 3$ and has full column rank, yielding the left null-space being $(2M-3)\times 2M$ following that for the left null-space nullity$(\mathbf{H}_{\hat{\mathbf{p}}_f})$ = dim$(\mathbf{H}^T_{\hat{\mathbf{p}}_f})$ − rank$(\mathbf{H}_{\hat{\mathbf{p}}_f})$. Thus, the amount of rows in $\mathbf{H}_{\hat{\mathbf{x}},n}$ and $\mathbf{v}_n$ will be $2M-3$. Furthermore, after projecting onto the null-space and since $\mathbf{A}$ is unitary and thus semi-orthogonal, the noise will be distributed after the following:

$$\begin{aligned}
\mathbf{v}_n &\sim \mathcal{N}(\mathbf{0}_{(2M-3)\times 1}, \mathbf{A}^T\sigma^2_{im}\mathbf{A}) \\
&\sim \mathcal{N}(\mathbf{0}_{(2M-3)\times 1}, \sigma^2_{im}\mathbf{I}_{(2M-3)\times(2M-3)})
\end{aligned} \tag{6.82}$$

This concludes the measurement prediction linearisation, such that the residual is on a form equivalent with equation (5.40).

6.8 Update Step

The filter goes through an update step if one of the following conditions are true when a new image is captured:

- There are features which have been tracked past the maximum track length.
- There are features which have gone out of the frame (out of the image).
- The sliding window is at maximum capacity and an augmented IMU state and the corresponding features which were tracked at that given time step need to be marginalised,

6.8.1 Outlier Rejection

Before the entries in the residual $\mathbf{r}_n$ are utilised in an update, they undergo a Mahalanobis distance test (section 5.3.1):

$$D^2 = (\mathbf{z}-\hat{\mathbf{z}})^T\mathbf{P}^{-1}(\mathbf{z}-\hat{\mathbf{z}}) \sim \chi^2(2M) \tag{6.83}$$

However, the residual is on the following form for the EKF update (note that as $\delta\mathbf{x}$ and $\mathbf{v}_n$ are uncorrelated, the covariance of the residual is simply the sum of the covariance for $\delta\mathbf{x}$ and $\mathbf{v}_n$):

$$\mathbf{H}_{\hat{\mathbf{x}},n}\delta\mathbf{x} + \mathbf{v}_n \sim \mathcal{N}(\mathbf{0}_{(2M-3)\times 1}, \mathbf{H}_{\hat{\mathbf{x}},n}\mathbf{P}\mathbf{H}_{\hat{\mathbf{x}},n}^T + \sigma_{im}^2\mathbf{I}_{(2M-3)\times(2M-3)}) \tag{6.84}$$

yielding that the squared Mahalanobis distance becomes:

$$D^2 = (\delta\mathbf{x})^T(\mathbf{H}_{\hat{\mathbf{x}},n}\mathbf{P}\mathbf{H}_{\hat{\mathbf{x}},n}^T + \sigma_{im}^2\mathbf{I}_{(2M-3)\times(2M-3)})^{-1}(\delta\mathbf{x}) \sim \chi^2(2M-3) \tag{6.85}$$

The realisation of D^2 is then compared against a confidence interval for a χ^2 distributed stochastic variable with $2M-3$ degrees of freedom.

6.8.2 QR Decomposition

During the update, a QR decomposition is performed on the rectangular matrix $\mathbf{H}_{\hat{\mathbf{x}},n}$ in order to reduce the computational cost:

$$\mathbf{H}_{\hat{\mathbf{x}},n} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}\begin{bmatrix}\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \\ \mathbf{0}\end{bmatrix} \tag{6.86}$$

$\mathbf{Q}_1$ and $\mathbf{Q}_2$ are unitary, yielding that the transpose is equal to the their respective inverses. The residual can thus be written as:

$$\mathbf{r}_n = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}\begin{bmatrix}\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \\ \mathbf{0}\end{bmatrix}\delta\mathbf{x} + \mathbf{w}_n \tag{6.87}$$

$$\begin{bmatrix}\mathbf{Q}_1^T \\ \mathbf{Q}_2^T\end{bmatrix}\mathbf{r}_n = \begin{bmatrix}\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \\ \mathbf{0}\end{bmatrix}\delta\mathbf{x} + \begin{bmatrix}\mathbf{Q}_1^T \\ \mathbf{Q}_2^T\end{bmatrix}\mathbf{w}_n \tag{6.88}$$

$$\mathbf{Q}_1^T\mathbf{r}_n = \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}\delta\mathbf{x} + \mathbf{Q}_1^T\mathbf{w}_n \tag{6.89}$$

Here, the residual $\mathbf{Q}_2^T\mathbf{r}_n$ is only noise and can be discarded. Let $\mathbf{r}_Q := \mathbf{Q}_1^T\mathbf{r}_n$ and $\mathbf{w}_Q := \mathbf{Q}_1^T\mathbf{w}_n$, where the noise becomes distributed according to:

$$\begin{aligned}\mathbf{w}_Q &\sim \mathcal{N}(\mathbf{0}_{\text{rows}(\mathbf{Q}_1^T)}, \mathbf{Q}_1^T\sigma_{im}^2\mathbf{I}_{(2M-3)\times(2M-3)}\mathbf{Q}_1) \\ &\sim \mathcal{N}(\mathbf{0}_{\text{rows}(\mathbf{Q}_1^T)}, \sigma_{im}^2\mathbf{I}_{\text{rows}(\mathbf{Q}_1^T)\times\text{rows}(\mathbf{Q}_1^T)})\end{aligned} \tag{6.90}$$

The finalised residual after null-space projection and QR decomposition thus becomes:

$$\mathbf{r}_Q = \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}\delta\mathbf{x} + \mathbf{w}_Q \tag{6.91}$$

which is on the form needed for the EKF update (equation (5.40)).

6.8.3 Updating the Filter

The Kalman gain can then be computed according to:

$$\mathbf{K}_{k+1} = \mathbf{P}_k^- \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}^T (\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \mathbf{P}_k^- \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}^T + \sigma_{im}^2 \mathbf{I}_{\mathrm{rows}(\mathbf{Q}_1^T)\times \mathrm{rows}(\mathbf{Q}_1^T)})^{-1} \tag{6.92}$$

This renders the correction to the nominal state accessible via:

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k^- \oplus \mathbf{K}_{k+1}\mathbf{r}_Q \tag{6.93}$$

where $\oplus$ here is defined as regular addition for all parts of the state except the orientations, where the quaternion product is utilised. The a posteriori covariance is updated according to (equation (5.26)):

$$\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1})\mathbf{P}_{k+1}^- \tag{6.94}$$

This concludes the update step of the filter.

6.9 Observability

In the original paper for MSCKF by Mourikis and Roumeliotis [13], the linearised system model utilised in the EKF can proven to be inconsistent, as Li and Mourikis [74] stated:

> By analyzing the observability properties of the linearized system model employed by the EKF, we prove that the MSCKF is inconsistent, i.e., that the covariance matrix of the estimation errors is larger than that computed by the filter [...]. In turn, this inconsistency leads to inaccurate state updates and ultimately a loss of accuracy.

Qiu *et al.* [3] utilises first-estimate jacobians to alleviate for the observability inconsistency in LARVIO, which are given by means of the a priori estimate, as shown in equation (6.52) and the following entries within the transition matrix.

6.10 Stationary Detection & Update

In order to alleviate drift when the system is stationary, a stationary detection can be used to swap the update step of the filter for a stationary situation. The stationary detection is based on whether the tracked features have sufficient movement between multiple images or not. If they do not, the system is assumed stationary and the stationary update is utilised. Since the full state vector consists of augmented IMU poses, these can be used to enforce constraints between the two last augmented IMU frames.

Note that in the following, the main constraint is given by the velocity of the IMU state being zero and the auxiliary constraints are given by the last two augmented IMU poses having identical position and orientation. The reader should also note that the noise has been excluded in order to ease the derivation, and will appear in the final linearised residual. The constraints are given by:

$$\begin{aligned} \mathbf{z}_{\mathbf{v}_b^w} &= \mathbf{v}_b^w = \mathbf{0}_{3\times1} \\ \mathbf{z}_{\mathbf{q}_{wb_k}} &= \mathbf{q}_{wb_k} \otimes \mathbf{q}_{wb_{k-1}}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T \\ \mathbf{z}_{\mathbf{p}_{b_k}^w} &= \mathbf{p}_{b_k}^w - \mathbf{p}_{b_{k-1}}^w = \mathbf{0}_{3\times1} \end{aligned} \tag{6.95}$$

which has the following nominal behaviour when applying the expectation:

$$\begin{aligned} \hat{\mathbf{z}}_{\hat{\mathbf{v}}_b^w} &= \hat{\mathbf{v}}_b^w \\ \hat{\mathbf{z}}_{\hat{\mathbf{q}}_{wb_k}} &= \hat{\mathbf{q}}_{wb_k} \otimes \hat{\mathbf{q}}_{wb_{k-1}}^{-1} \\ \hat{\mathbf{z}}_{\hat{\mathbf{p}}_{b_k}^w} &= \hat{\mathbf{p}}_{b_k}^w - \hat{\mathbf{p}}_{b_{k-1}}^w \end{aligned} \tag{6.96}$$

This allows for forming residuals between the true constraints and the expected ones. The velocity residual for the IMU state is given by the following:

$$\begin{aligned} \mathbf{r}_{\mathbf{v}_b^w} &= \mathbf{z}_{\mathbf{v}_b^w} - \hat{\mathbf{z}}_{\mathbf{v}_b^w} \\ &= \mathbf{v}_b^w - \hat{\mathbf{v}}_b^w \\ &= \delta\mathbf{v}_b^w \end{aligned} \tag{6.97}$$

whereas the orientation residual between the last two augmented IMU states is given by:

$$\begin{aligned} \mathbf{r}_{\mathbf{q}_{wb_k}} &= \mathbf{z}_{\mathbf{q}_{wb_k}}^{-1} \otimes \hat{\mathbf{z}}_{\mathbf{q}_{wb_k}} \\ &= (\underbrace{\mathbf{q}_{wb_k} \otimes \mathbf{q}_{wb_{k-1}}^{-1}}_{\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T})^{-1} \otimes (\hat{\mathbf{q}}_{wb_k} \otimes \hat{\mathbf{q}}_{wb_{k-1}}^{-1}) \\ &= \hat{\mathbf{q}}_{wb_k} \otimes \hat{\mathbf{q}}_{wb_{k-1}}^{-1} \\ &= \delta\mathbf{q}_{wb_k}^{-1} \otimes \mathbf{q}_{wb_k} \otimes (\delta\mathbf{q}_{wb_{k-1}}^{-1} \otimes \mathbf{q}_{wb_{k-1}})^{-1} \\ &= \delta\mathbf{q}_{wb_k}^{-1} \otimes \underbrace{\mathbf{q}_{wb_k} \otimes \mathbf{q}_{wb_{k-1}}^{-1}}_{\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T} \otimes \delta\mathbf{q}_{wb_{k-1}} \\ &= \delta\mathbf{q}_{wb_k}^{-1} \otimes \delta\mathbf{q}_{wb_{k-1}} \\ &\approx \begin{bmatrix} 1 \\ \frac{-\delta\boldsymbol{\theta}_{wb_k} + \delta\boldsymbol{\theta}_{wb_{k-1}}}{2} \end{bmatrix} \end{aligned} \tag{6.98}$$

Here, the constraint that $\mathbf{z}_{\mathbf{q}_{wb_k}} = \mathbf{q}_{wb_k} \otimes \mathbf{q}_{wb_{k-1}}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ has been used to cancel the terms related to $\mathbf{q}_{wb_k}$ and $\mathbf{q}_{wb_{k-1}}$. Furthermore, the small angle assumption of error-quaternions from equation (6.3) has been utilised. Disregarding the real part of the error-quaternion in the residual yields:

$$\mathbf{r}_{\boldsymbol{\theta}_{wb_k}} = \frac{-\delta\boldsymbol{\theta}_{wb_k} + \delta\boldsymbol{\theta}_{wb_{k-1}}}{2} \tag{6.99}$$

The position residual between the last two augmented IMU states is given by:

$$\begin{aligned}\mathbf{r}_{\mathbf{p}^w_{b_k}} &= \mathbf{z}_{\mathbf{p}^w_{b_k}} - \hat{\mathbf{z}}_{\mathbf{p}^w_{b_k}} \\ &= \mathbf{p}^w_{b_k} - \mathbf{p}^w_{b_{k-1}} - (\hat{\mathbf{p}}^w_{b_k} - \hat{\mathbf{p}}^w_{b_{k-1}}) \\ &= \delta\mathbf{p}^w_{b_k} - \delta\mathbf{p}^w_{b_{k-1}}\end{aligned} \tag{6.100}$$

It should be noted that since $\mathbf{z}_{\mathbf{p}^w_{b_k}} = \mathbf{0}_{3\times1}$ by the definition of the constraint, the term can be dropped. The error-state formulation is given here to relate the residual to the error-state when the jacobian is derived.

The preceding residuals can be utilised to form the basis for a residual on the standard EKF form, $\mathbf{r} = \mathbf{H}\delta\mathbf{x} + \mathbf{v}$, where the jacobian with respect to the error-state is given by:

$$\begin{aligned}\mathbf{H}_z &= \frac{\partial \begin{bmatrix}\mathbf{r}^T_{\mathbf{v}^w_b} & \mathbf{r}^T_{\mathbf{q}_{wb_k}} & \mathbf{r}^T_{\mathbf{p}^w_{b_k}}\end{bmatrix}^T}{\partial\delta\mathbf{x}} \\ &= \begin{bmatrix}\mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{16\times3} & \cdots & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{16\times3} & \cdots & \frac{1}{2}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & -\frac{1}{2}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{16\times3} & \cdots & \mathbf{0}_{3\times3} & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3}\end{bmatrix}\end{aligned} \tag{6.101}$$

This yields that the update step for the stationary case is given by the following, where the assumed noise for the residual is given by $\mathbf{v}_z \sim \mathcal{N}(\mathbf{0}_{9\times9}, \mathbf{R}_z)$:

$$\mathbf{r}_z = \mathbf{H}_z\delta\mathbf{x} + \mathbf{v}_z \tag{6.102}$$

This allows for the calculation of the Kalman gain and the correction of the states to be done in the same way as outlined in table 5.2.

Chapter 7

Microcontroller Fundamentals

MCUs (microcontroller units) are small processors with usually a single core and far less memory (ranging from usually a few kilobytes to a few megabytes) than conventional consumer processors. Their ALUs (arithmetic logic units) have sizes of 8 bits, 16 bits or 32 bits, meaning the ALU can operate on 8-bit, 16-bit or 32-bit numbers. Registers are the data containers within the processor where the data from the memory is loaded into before the processor can perform arithmetic or operations on the data. The size of the registers follows the size of the ALU. Register access time can be assumed to be negligible compared to memory access. An 8-bit architecture does not limit MCUs from operating on 32-bit integers and floating point numbers, but more registers and instructions have to be used. Conventional processors usually have 64-bit ALUs.

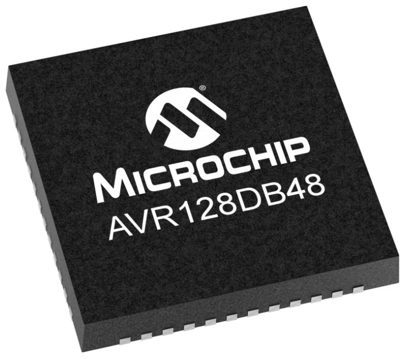


Figure 7.1: The 8-bit AVR128DB48 microcontroller from Microchip, from [75].

MCUs will generally use static memory (SRAM), which will retain its content as long as power is supplied. With dynamic memory (DRAM), which is typically found in consumer processors, the memory must be continuously refreshed. This makes it possible for the clock frequency of MCUs to not necessarily have a lower boundary. Some MCUs, such as the ATTiny85 by Microchip, have a specification which allows a minimum value of 0 Hz for the provided clock signal. Thus, MCUs have the functionality to be clocked fairly low to consume as little power as possible due to the staticness of the memory and, thus also the staticness of the state of the system. This is commonly used for remote sensing applications where the system has to operate for as long as possible on battery power. The upper bound on the clock frequency for MCUs typically ranges from tens to hundreds of MHz. To the author's knowledge at the writing of this master's thesis, the maximum clock speed achieved by a microcontroller is 1 GHz.

Due to the limited amount of memory and processing speed, MCUs are typically used for a single purpose or a rather finite set of purposes. That does not necessarily limit MCUs to computationally inexpensive tasks. The higher end MCUs with clock speed up to hundreds of MHz can perform quite computationally intensive tasks such as running small neural networks. They are also quite common in applications with hard real-time requirements, as timing of tasks on MCU can be done extremely precisely.

MCUs typically have capabilities for peripheral access with protocols such as *UART* (universal asynchronous receiver-transmitter), *I2C* (inter-integrated circuit), *USB* (universal serial bus) and *CAN* (controller area network). They are popularly used within everything from the automotive industry, to home appliances, to the space industry to supporting functions within consumer electronics.

Due to their limited program memory, MCUs do not run full-fledged operating systems. Small kernels exist, such as the FreeRTOS project [76] (typically occupying 5 KB - 9 KB of program memory). However, these projects can also be considered to reside under the bare-metal umbrella. The FreeRTOS kernel can be seen as a scheduler for different tasks, where mutexes, semaphores, notifications and message buffers are provided. Projects such as FreeRTOS allow for concurrency (and parallelism if there are multiple cores) with real-time constraints for MCUs.

7.1 Instruction Set

Microcontroller units usually have a *RISC* (reduced instruction set computer) architecture. The instruction set ranges over fewer instructions than available on *CISC* (complex instruction set computer) architectures. Examples of RISC are the Cortex-M and Cortex-A architecture designs provided by ARM, the AVR architecture by Microchip (formerly ATMEL), and RISC-V. An example of a CISC architecture is x86, found in Intel's and AMD's processors, which are ubiquitous in the conventional computer domain (laptops and desktops). A RISC architecture is simpler and can, to a greater extent, utilise *pipelining* (a way for the processor to prepare to run the next instruction before the current instruction is completed). In contrast, a CISC architecture can perform some specific operations in fewer clock cycles due to its greater instruction set.

7.1.1 Floating Point Instructions

Some MCUs (e.g. some variants of ARM Cortex) can do floating point arithmetic with a dedicated FPU (floating point unit). Floating point arithmetic is computationally expensive and can be significantly alleviated by these dedicated units. Such MCUs have an extension to their instruction for floating point instructions.

7.1.2 Single Instruction Multiple Data

SIMD instructions allow the processor to operate on more than a single element of data within one clock cycle. For a 32-bit microcontroller, SIMD allows operations on four 8-bit integers or two 16-bit integers in a single instruction. Such instructions can significantly improve performance if used with care.

The main idea of this can be seen in Figure 7.2, where four byte additions are done simultaneously. The difference here with 32-bit addition can be observed in the second column, where the carry of the addition of the leftmost bit is not propagated to the byte in the first column. SIMD instructions thus operate on multiple elements *independently*.

$$\begin{array}{ccccc}
 & 00000011 & 10000000 & 00010000 & 00001010 \\
+ & 00010000 & 10000000 & 00101000 & 00000011 \\
\hline
= & 00010011 & 00000000 & 00111000 & 00001101
\end{array}$$

Figure 7.2: Demonstration for a SIMD add instruction, where the 4 bytes from two 32-bit integers are added independently.

The SIMD instruction extension for ARM Cortex MCUs allows for a range of elementary operations: addition, subtraction, bit-shifts and multiplication. The elementary operations are mainly split into instructions for signed or unsigned integers and whether they should saturate the result or allow overflow. In figure 7.2, and overflow occurred in the second byte, but this can be prevented by using the saturating variant of the SIMD instruction, where the result would have been 11111111 instead. Furthermore, within the instruction set extension, there are more specialised instructions which do multiple elementary operations within one SIMD instruction. An example of this is the *UHADD8* instruction, which does 4 unsigned additions before halving the result for each addition. The instruction is read as *Unsigned Halved Addition 8-bit*, and is given by algorithm 2.

It should be stated that operating with SIMD instructions is not done by passing four and four bytes to the instructions. They operate on 32-bit integers, so the data has to be packed sequentially in memory.

Algorithm 2 Demonstration of the UHADD8 SIMD instruction. Note that the notation $[x : y]$ is used here to signify the bits from y to x.

```
Input: $A$, first group of bytes
Input: $B$, second group of bytes
Output: $R$, result

1: $R[7:0] \leftarrow \frac{A[7:0]+B[7:0]}{2}$
2: $R[15:8] \leftarrow \frac{A[15:8]+B[15:8]}{2}$
3: $R[23:16] \leftarrow \frac{A[23:16]+B[23:16]}{2}$
4: $R[31:24] \leftarrow \frac{A[31:24]+B[31:24]}{2}$
```

7.2 Memory

MCUs have the ability for supporting a range of memory configurations. The common factor however is usually a Harvard architecture, where the instructions of the program and the data are located in separate memory units. Usually, the memory is located *close* to the processor, meaning the memory is spatially close to the processor and not outside a possible cache. This allows the memory to be accessed quickly and often operate on the same frequency as the processor. Such memory is often referred to as *TCM* (tightly-coupled memory).

The program instructions are usually placed in non-volatile *FLASH* memory. However, there are exceptions to this. Some MCUs have the option to place both instructions and data in TCM as well as having the option to decide which parts of the program that is placed in FLASH memory.

SRAM is used for volatile memory such as TCM. SRAM is fast, but requires more physical space and is costlier than DRAM. The use of slower DRAM in conventional computers is alleviated by a range of caches (often SRAM) between the processor and the DRAM. This is also the case for the MCUs which have capabilities for external DRAM.

The width of the memory bus and the frequency it operates on will determine how quickly data can be loaded into the processor. These can vary: the bus width of TCM can be 64 bits even though the architecture is 32-bit, which speeds up retrieval of data and instructions. The clock speed of the memory bus can be to some degree — in a microcontroller setting at least — up to the application to define. Some MCUs have the ability to have several *clock domains*, which determine the speed of the buses and the peripherals the processor accesses. An example of this can be seen in figure 7.3, where there are three main domains: D1, D2 and D3. Within these domains, there are different buses, e.g. AHB (Advanced High-performance Bus), APB (Advanced Peripheral Bus) and AXI (Advanced Extensible Interface), which can be clocked differently.

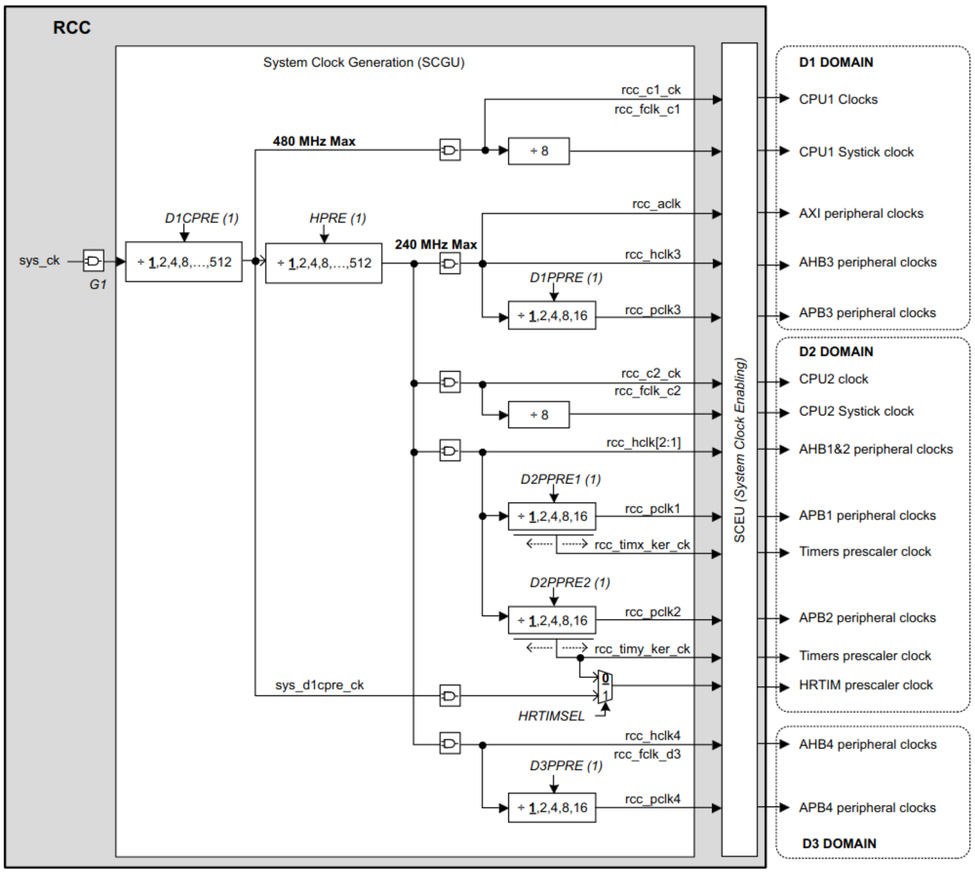


Figure 7.3: Clock tree of the STM32H747 MCU, from [77].

7.2.1 Cache

Cache alleviates slower memory that is not tightly coupled with the processor (both for instructions and data). In the microcontroller domain, the cache size can be in the range of a few tens of kilobytes. The cache will attempt to predict what memory the processor will access in the future, by spatial and temporal locality. Meaning, access to one address location is likely to be followed by access to adjacent memory addresses (e.g. iterating through a list) and access to one address location is likely to repeat within a short period of time (e.g. loop counters).

When the prediction fails, a *cache miss* occurs. The processor then has to wait for the memory to be retrieved. This is an area where significant optimisations can be made. Structuring programs around playing along with the cache and reducing the amount of cache misses can have a great effect on the performance of a program. This comes from the aspect that in modern computers, memory access can be a rather significant bottleneck. An example of this process can be seen in figure 7.4, where a read operation is outlined.

Write operations are similar, where the data can be written to the cache and either written to the memory instantly, called *write-through*, or the data is written to the memory at a later point in time when the whole cache block is replaced, called *write-back*. Write-through is simpler and maintains cache consistency, but it comes at the cost of write operations requiring more time. Write-back on the other hand reduces the amount of write operations to the memory, but comes of the cost of cache inconsistency. This can become problematic if the application utilises functionality for moving data without involving the CPU, called *direct memory access*. If DMA is utilised to move data from a peripheral unit to memory, the cache has to be *invalidated* for the respective addresses of the data, forcing the cache to grab the data from the memory when the processor reads it. Conversely, in order to force the cached data to be written to memory before a DMA operation, the cache is *cleaned* for the respective addresses.

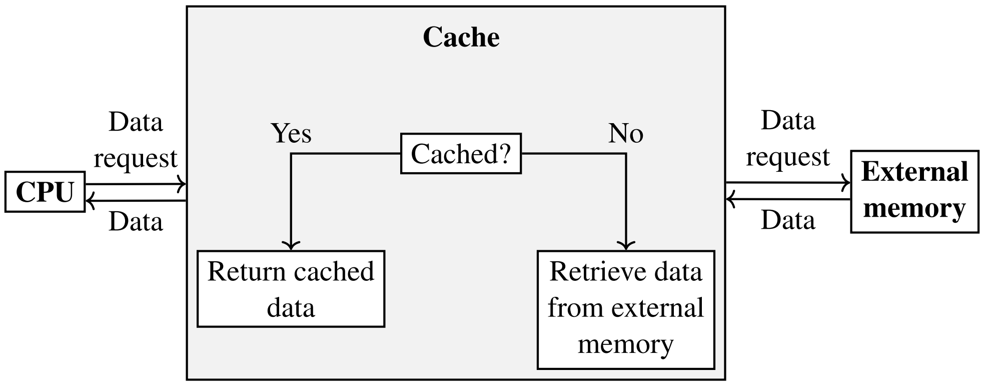


Figure 7.4: Flow chart of a read operation with cache.

7.2.2 Memory Layout

During the compilation and linkage of a program, different sections are defined. A program's global data will be either placed in the sections called *bss* (block started by symbol) or *data*, based on if the data is uninitialised or not. The code is placed in *text*. During runtime, variables allocated by the program either reside on the stack or the heap. Variables placed on the stack will be automatically popped from the stack when the scope of the given variable is left. Variables placed on the heap are up to the program to free explicitly.

For MCUs, there is a great deal of flexibility for where segments of code and data are placed. E.g. a particular function might be explicitly defined to reside in the ITCM (instruction tightly-coupled memory) of the microcontroller instead of external flash. The heap might be defined to reside on external RAM or within the DTCM (data tightly-coupled memory). An example of a memory map layout can be seen in figure 7.5, where the different data sections are defined with their respective address range. In this example, the physical locations of the memory regions are not specified, but they are arbitrary; they can reside in TCM or external RAM by their respective definitions in the linker file of the program.

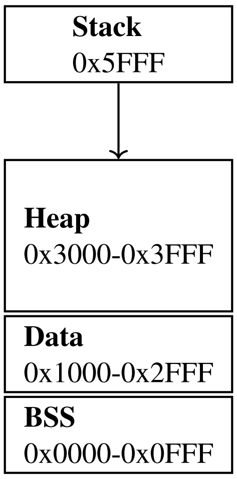


Figure 7.5: Example memory map layout. Note that the stack grows downwards in this example, which is the default behaviour on ARM Cortex-M.

If the stack grows into the other sections of the memory, a stack overflow occurs. This will not necessarily raise an error on MCUs, and can cause severely off-spec behaviour in the program. This can be detected by running compile-time analysation on the program and statically declaring most of the data. Furthermore, reducing the *depth* of function calls can also alleviate a growing stack. Before every function call, the *stack frame* is pushed onto the stack. The stack frame contains the current values of the registers, input values to the function and the address of the next instruction. This makes it possible for the processor to return to the state it was in after the function is executed. Having many nested function calls thus increases the required size of the stack.

If the program attempts to dynamically allocate more memory than is left available on the heap, the allocator will return a null pointer, which also causes havoc. In a conventional operating system, this would cause the program to crash, not propagating the error to the rest of the operating system. However, for MCUs, the whole application will either jump to a fault handler or continue in an undefined manner, forcing a reset of the whole system necessary. Therefore, the use of dynamic allocation on MCUs is a delicate topic, often disabled in its entirety or used with extreme care.

7.3 Interrupts

Interrupts are pieces of code that are executed when a specific condition occurs. This can be when an ADC (analogue to digital converter) conversion is done, a byte is received over UART, or a timer has finished counting to an arbitrary value. The code associated with interrupts is defined in *Interrupt Service Routines* (ISRs) and can be called by the processor at any time during the execution of the main program, given that the condition for the interrupt is fulfilled. When this happens, the processor interrupts the current execution of the program and jumps to the interrupt service routine, returning to where it was in the program after executing the code in the ISR.

7.4 Multi-Core Processing

A simple way to do multi-core processing on MCUs can be achieved by means of message passing. Typically, this is solved by having a dedicated *messaging unit* (MU), which will notify the respective cores through an interrupt when a message arrives. In this way, events can be raised that notify the other cores about starting a specific task. Various implementations build on this concept, e.g. *eRPC* (embedded remote procedure call) and *RPMsg* (remote processor messaging). For implementations such as eRPC and RPMsg, dedicated shared memory areas accessible from all cores are defined. These memory regions make it possible to store the messages, which then can be read from the other cores after a message interrupt occurs. An example of message passing can be seen in figure 7.6, where the main core notifies the second core about starting a task. When the second core has finished its task, it will notify the main core.

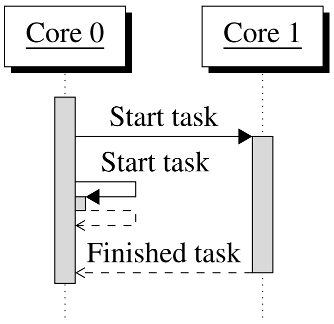


Figure 7.6: Multi-core message passing.

A conventional system based on threads is harder to replicate. This is due to having to do context switching if more threads than cores are spun up. Moreover, certain memory regions might only be accessible from certain cores, and the capabilities of the different cores might vary.

Chapter 8

Implementation

This chapter outlines the decisions made regarding the hardware platform and the implementation details for the frontend and the backend of the VIO pipeline.

8.1 Hardware

In a VIO pipeline, a substantial amount of data has to be processed in the frontend. Doing this at an acceptable frame rate requires a decent clock speed and a sufficient amount of fast memory. Modern processors alleviate access to slower memory with the use of cache. However, for microcontrollers, a far more suitable approach is to have a decent amount of memory tightly coupled with the processor — within the cache line — where access time is much quicker, and the memory operates at the same frequency as the processor.

A VGA image has a resolution of 640×480 pixels. With a bit depth of 8 bits, this results in 307.2 KB of data per image frame. Furthermore, for the frontend to be able to process image data of this size on, e.g. a frame rate of 20 Hz, the microcontroller has to be able to — at an absolute minimum — process $640 \times 480 \times 20 = 6,144,000$ data points per second. For each data point, there will be extra instruction cost, e.g. blurring the image with convolution has to be done before feature extraction to remove noise from the image. Separable convolution has an algorithmic runtime of $O(w \times h \times 2N)$, where w is the width of the image, h is the height of the image, and N is the kernel size. However, the instruction count will be a multiple of this algorithmic runtime as it does not account for loading the data into the registers and incrementing loop counters in addition to the actual multiplication of the kernel and the image values. With e.g. a kernel size of 3, 6 multiplications have to be performed for separable convolution. In addition, one can assume that 6 load instructions are needed to bring the data into the registers, accompanied by a store instruction to write the result into memory. Moreover, loop counters have to be updated (assuming at a minimum 2 instructions for this, one for incrementing the loop counter and one for branching). This results in: $640 \times 480 \times 20 \times (6 + 6 + 1 + 2) = 92,160,000$ instructions. This example is solely for one part of a VIO pipeline, emphasising that a clock frequency ranging well above a few hundred MHz is needed for such a pipeline to work under these specifications.

The backend in a VIO pipeline has to operate on a decent amount of floating-point data, e.g. for matrix multiplication. To do the matrix multiplication fast, as with the frontend, there should also be enough fast memory to alleviate memory access time. Moreover, as previously discussed, floating point arithmetic is also quite expensive without a dedicated FPU. It can thus be argued to be a rather hard requirement for a microcontroller doing VIO at an acceptable frame rate. This concludes that for a microcontroller to be able to run a VIO pipeline at an acceptable frame rate, it needs:

- Clock speed ranging far above a few hundred MHz
- Sufficient amount of fast memory closely coupled to the processor
- An FPU

This master's thesis bases its implementation on the NXP i.MX RT1170 microcontroller (particularly the NXP i.MX RT1170 evaluation kit), which is a dual-core microcontroller with a 1 GHz ARM Cortex-M7 and a 400 MHz ARM Cortex-M4. The M7 has 512 KB of TCM, and the M4 has 256 KB of TCM. Moreover, the microcontroller has 1.25 MB on-chip RAM, and the evaluation kit comes bundled with 64 MB external SDRAM running at 200MHz. The SDRAM and OCRAM are connected to the cores' cache, which is 32 KB and 16 KB for the M7 and M4 core, respectively. The i.MX RT1170 has an FPU, as well as a GPU that can be used for image processing acceleration. The evaluation kit also comes bundled with 16 MB of non-volatile program memory. The NXP i.MX RT1170 is one of the most performant microcontroller units out on the market as of the writing of this master's thesis, and to the author's knowledge, the only microcontroller which has broken the GHz barrier in clock speed. Basing the implementation on the i.MX RT1170 was therefore considered the most promising choice of hardware for this master's thesis.

The Arduino Nicla Vision was initially considered, which shares many similarities with the i.MX RT1170 and is based on the STM32H7 microcontroller. It is not, however, as performant as the i.MX RT1170. The specialisation project preceding this master's thesis was based on the i.MX RT1160, which can be considered as the i.MX RT1170's little brother. It was chosen due to chip shortage and long lead times for the i.MX RT1170. Thus, for this master's thesis, the associated code from the specialisation project was ported over to the i.MX RT1170.

The i.MX RT1170 consists of two cores with the *ARMv7E-M* architecture, which has capabilities for floating-point and SIMD instructions. The TCM of both cores contains ITCM and DTCM, and the distribution can be adjusted such that, e.g. more of the TCM is utilised for DTCM. However, this adjustment can only be made in banks of 32 KB. A valid configuration for the M7-core's TCM could be 64 KB for ITCM and 448 KB for DTCM. The outline of the RAM can be seen in table 8.1.

As one can observe in table 8.1, the TCM operates on the processor's clock speed, whereas the OCRAM operates on the main bus frequency of 240 MHz. The slower speed of the OCRAM is alleviated with cache and a wide bus width. The external

RAM	Within cache line	Accessible from	Size	Bus frequency	Bus width
M7 TCM	Yes	M7-core	512 KB	Core speed (1 GHz)	64-bit/2x32-bit
M4 TCM	Yes	Both cores	256 KB	Core speed (400 MHz)	2x32-bit
On-Chip RAM	No	Both cores	1.25 MB	Main bus frequency, 240 MHz	64-bit
External SDRAM	No	Both cores	64 MB	200 MHz	16-bit

Table 8.1: Different types of RAM available on the i.MX RT1170 evaluation board. Note that the TCM bus for the M7 core is 64-bit for instructions and 2x32-bit for data, where there is one bus for odd-numbered data addresses and one bus for even-numbered data addresses.

SDRAM is the slowest, with the lowest bus width. A write and read speed demonstration can be seen in table 8.2, where a test of reading and writing a 100 KB buffer 10 000 times was performed with *memcpy*. The cache on the M7 core of the i.MX RT1170 is 32 KB, yielding that the cache had to be updated throughout the iterations of reading as it does not have the capacity for the whole buffer.

RAM	Read without cache	Read with cache	Write without cache	Write with cache
M7 TCM	3.58 GB/s	3.58 GB/s	3.58 GB/s	3.58 GB/s
On-Chip RAM	0.09 GB/s	0.60 GB/s	1.83 GB/s	1.83 GB/s
External SDRAM	0.03 GB/s	0.18 GB/s	0.42 GB/s	0.43 GB/s

Table 8.2: Write and read speed for the different memories available on the i.MX RT1170 evaluation kit. The cache was invalidated before each set of read operations and cleaned after each set of write operations.

In this example, a write-through policy was used for the cache, which is why the cache does not affect the write operations. Furthermore, one can see that the TCM is unaffected by whether the cache is enabled due to being tightly coupled with the processor and within the cache line. This also shows how much faster writing to memory is than reading when the CPU operates on a higher frequency than the memory. When writing, the CPU issues the write command and can continue with the next instruction. However, when reading, the CPU has to stop and wait for the data, effectively wasting clock cycles.

Table 8.2 also shows the effect of the bus width, where one can observe that for the operations with the TCM, the CPU can seemingly read and write 4 bytes per clock cycle. This is due to the 2x32-bit bus width of the TCM, enabling 4 bytes on the bus simultaneously for every even/odd address. This is also prevalent when comparing the SDRAM and the OCRAM. There is roughly a 3x data rate for reading from the OCRAM when the cache is disabled, even though the two memories operate on comparable frequencies.

8.2 Libraries Utilised

The implementation discussed in the following section uses Eigen [5] for linear algebra and the Embedded Template Library (ETL) [6] for containers such as vectors. The ETL replicates the standard library for C++ but focuses on compile-time known sizes of its containers. Thus, an upper boundary for how much memory the implementation consumes can be found at compile-time.

8.3 Frontend

The frontend developed in this thesis builds on the feature extractor and tracker built in the preceding specialisation project — which was inspired by the frontend in OpenVINS [4] — where FAST [40, 41] is utilised as the feature extractor and Lucas-Kanade [42] with image pyramids is utilised for feature tracking. The choice in the specialisation project to build the components for a direct method was due to reducing the computational cost. It should be stated that the original implementation of MSCKF by Mourikis and Roumeliotis [13] utilises SIFT, which is rather computationally demanding but can be more robust.

The following sections outline how the feature extractor and feature tracker were built in the preceding specialisation project. It will also outline adaptions made to the frontend to make it more robust against track outliers. Lastly, a high-level outline of the frontend will be shown.

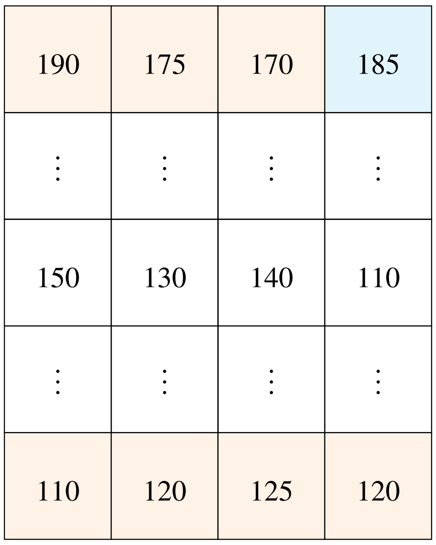
8.3.1 FAST

The FAST implementation constructed in the specialisation project became significantly targeted towards the ARMv7E-M architecture of the microcontroller. Most prevalent with the use of SIMD instructions to quickly scan through an image. Furthermore, lookup tables were used to determine whether a given pixel was within the FAST threshold. The images and the respective data associated with them were stored in DTCM for fast memory access.

SIMD Rejection Test

It was found in the specialisation project that the rejection of candidate pixels was one of the main bottlenecks, simply due to the sheer amount of data. As an image resides naturally sequentially in memory, row after row, SIMD could be utilised. The images have a bit depth of 8, yielding that SIMD could be used for operations on four pixels in a single instruction. For the rejection of candidate pixels, this was utilised such that a group of four pixels could be examined simultaneously. The main idea can be seen in figure 8.1, where an excerpt from a feature candidate test with the opposite pixel pairs at the top and bottom are shown. The idea of testing opposite pixel pairs in the FAST pattern comes from the fact that one of the opposite pixel pairs has to be *outside* the threshold range for the pixel to be a feature candidate, as outlined in section 4.5.4. In figure 8.1, a FAST threshold of 50 is used, rendering that the only pixel value in the pattern being outside the range of the corresponding centre pixel intensity ± the threshold is the top right-most pattern pixel. The three left-most candidates can thus be discarded without any further processing.

The gist of the SIMD candidate rejection test developed in the specialisation project can be seen in algorithm 3, where a test of one of the opposite pixel pair groups, denoted by the index i, is shown. Firstly, two 32-bit integers are constructed, which take the current pixel pointer value converted to a 32-bit number (so that it essentially is packed with 4 pixel candidates), here denoted C, and add/subtract a 32-bit



190	175	170	185
⋮	⋮	⋮	⋮
150	130	140	110
⋮	⋮	⋮	⋮
110	120	125	120

Figure 8.1: Idea behind SIMD feature candidate test: processing groups of 4 pixels. The four feature candidates in the middle are compared against the top row and then against the bottom row. Here a FAST threshold of 50 is used. Only the right-most pixel in the top row passes the test. The first three candidates can thus be discarded without any further processing.

integer filled with the threshold at each byte. With this, one essentially has two 32-bit integers with 4 pixel candidate values ± the threshold. The instructions used here are *UQADD8* and *UQSUB8*, which will clamp the value for each byte to an unsigned value between 0 and 255.

After that, a check is performed for whether the pixel candidates plus the threshold are not all at the maximum value. If they are, it does not matter what the pattern pixel values are; they cannot be greater than 255, no matter what. If this is not the case, the algorithm checks if the pattern pixels are greater than the pixel candidates plus the threshold by means of subtraction. Meaning, if a given pattern pixel value minus the candidate pixel value plus the threshold is greater than zero, it has passed the test. Here the *USUB8* SIMD instruction is used, which will set underflow flags within the processor. If an underflow occurs (which happens if the pattern pixel value is greater than the centre pixel plus the threshold), these can be read out with the *SEL* instruction.

The SEL instruction with the associated parameters will return 0xFF for the subtractions where underflow occurs and 0x00 otherwise, at the corresponding byte positions. This will thus result in the pattern pixels being above the threshold range having their indices within the m_+ mask filled with 0xFF. The same logic is performed for the pattern pixels below the threshold range, as seen in lines 11 to 16. In the specialisation project, it was found that doing two such checks for the top-to-bottom pattern pixels and left-to-right pattern pixels yielded the most performant implementation, ruling out most pixels.

Algorithm 3 SIMD feature candidate test

```
Input: $C$, candidate pixels' intensity (32-bit)
Input: $P_0 \dots P_{15}$, pattern pixel intensities (32-bit)
Input: $i$, pattern pixel index
Input: $t$, intensity threshold, repeated threshold at each byte (32-bit)
Output: $m$, mask determining if opposite pixel pairs are above or below threshold
    range
 1: $C_+ \leftarrow$ UQADD8($C$, $t$)
 2: $C_- \leftarrow$ UQSUB8($C$, $t$)
 3: $m_+ \leftarrow 0$
 4: $m_- \leftarrow 0$

 5: if $C_+ <$ 0xFFFFFFFF then
 6:     USUB8($P_i$, $C_+$)
 7:     $m_+ \leftarrow$ SEL(0xFFFFFFFF, 0x0)
 8:     USUB8($P_{i+8}$, $C_+$)
 9:     $m_+ \leftarrow m_+$ OR SEL(0xFFFFFFFF, 0x0)
10: end if

11: if $C_- >$ 0x0 then
12:     USUB8($C_-$, $P_i$)
13:     $m_- \leftarrow$ SEL(0xFFFFFFFF, 0x0)
14:     USUB8($C_-$, $P_{i+8}$)
15:     $m_- \leftarrow m_+$ OR SEL(0xFFFFFFFF, 0x0)
16: end if

17: $m \leftarrow (m_+$ OR $m_-)$
```

Lookup Table Rejection Test

When a candidate passes the SIMD rejection tests, it will be evaluated further with a lookup table test, where t is the FAST threshold.

Index	**Value**
$[0, 255-t)$	1
$[255-t, 255+t)$	0
$[255+t, 512)$	2

Table 8.3: Lookup table used for rejection test.

The test is then based on using the lookup table with the following pseudocode:

$$\mathbf{l}[255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_p)] \tag{8.1}$$

where $\mathbf{I}(\mathbf{p}_c)$ is the pixel value of the candidate being examined, $\mathbf{I}(\mathbf{p}_p)$ is a pattern pixel value and $\mathbf{l}$ is the lookup table. An example with $t = 100$, a candidate pixel value of 116 and different pattern pixel values can be seen in table 8.4. In this example, the first and third entries are outside the threshold range and should be examined further.

$\mathbf{I}(\mathbf{p}_p)$	$255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_p)$	**Lookup Table Value**	**Description**
6	145	1	Below threshold range
40	179	0	Within threshold range
220	359	2	Above threshold range

Table 8.4: FAST pattern lookup table example

The lookup table allows for creating a mask for every opposite pixel pair by utilising logical OR. The mask is then logically AND-ed with the rest of the pairs. If the mask is 0 at any point after a lookup table test between two opposite pixel pairs, the candidate can be rejected since at least one of them has to be outside the threshold range for the candidate to be a potential feature. An excerpt of this can be seen in algorithm 4. Note that in the implementation, the lookup table test happens on a per-pixel basis and the pattern pixels were also stored in a lookup table for fast access.

Algorithm 4 Lookup table feature candidate test

```
Input: $\mathbf{l}$, lookup table
Input: $\mathbf{I}$, image
Input: $\mathbf{p}_c$, position of the candidate pixel
Input: $\mathbf{p}_0 \dots \mathbf{p}_{15}$, pattern pixel positions

1: $m \leftarrow \mathbf{l}[255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_0)]$ OR $\mathbf{l}[255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_8)]$
2: if $m = 0$ then
3:     reject candidate
4: end if

5: $m \leftarrow m$ AND $(\mathbf{l}[255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_4)]$ OR $\mathbf{l}[255 - \mathbf{I}(\mathbf{p}_c) + \mathbf{I}(\mathbf{p}_{12})])$
6: if $m = 0$ then
7:     reject candidate
8: end if

9: ...
```

If a given candidate passes all the lookup table tests, it will be examined in detail for 9 consecutive pixels being either above or below the threshold range, using the information from the lookup table test to know which condition to check for.

Results From the Specialisation Project

The custom implementation of FAST from the specialisation project was tested against OpenCV's implementation, using the EuRoC dataset. The images in the EuRoC dataset have a resolution of 752×480. The custom implementation was found to correspond to OpenCV for 99.8% of the features with an average runtime of 4.962 ms for an average of 86.4 features per image (using a FAST threshold of 80).

8.3.2 Lucas-Kanade

The algorithmic implementation of Lucas-Kanade from the specialisation project was not implemented with specific dependence on the MCU's instruction set, simply due to there being fewer aspects of the algorithm which could be alleviated by e.g. SIMD. That said, how the algorithm operated on memory and how to reduce the amount of memory needed was explored to reduce the runtime and required memory.

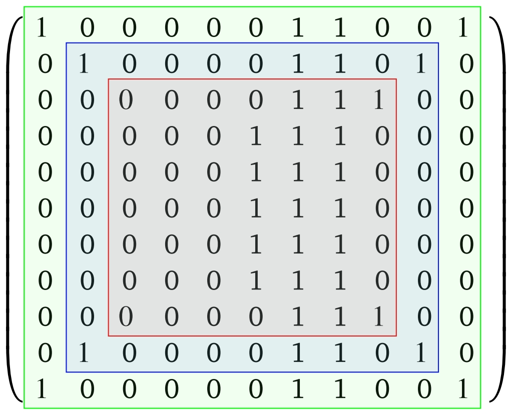
The gist of the algorithm developed in the specialisation project can be inspected in algorithm 5. The implementation used image pyramids to increase robustness (as outlined in section 4.5.5). However, in classical Lucas-Kanade implementations, one usually constructs image pyramids for the two images with which the tracking is done. An image pyramid with 5 levels of an image from the EuRoC dataset (at a resolution of 752×480), where each level is halved in width and height from the previous layer, requires $\sum_{i=0}^{4} \frac{752}{2^i} \cdot \frac{480}{2^i} = 480,810$ bytes (bit depth of 8). Thus, using two such image pyramids for the two images would require 122.2% of the available RAM on the M7 core (when excluding the memory reserved for the M4 core) on the i.MX RT1160 (which was used in the specialisation project). Thus, the implementation could not use this approach. In the specialisation project, the implementation rather used local pyramids around the neighbourhood — the patches — of the features. In the following, $\mathbf{P}_{\text{pyr}}^{k}$ is the patch pyramid of image $\mathbf{I}^k$.

Algorithm 5 Lucas-Kanade pipeline

```
Input: $\mathbf{P}_{\text{pyr}}^{k-1}$, patch pyramid from previous frame
Input: $\mathbf{I}^k$, image from the current frame
Output: $\mathbf{P}_{\text{pyr}}^{k}$, patch pyramid used in the next frame
Output: $\mathbf{U} = \begin{bmatrix} \mathbf{u}_0 & \dots & \mathbf{u}_M \end{bmatrix}$, list of features

1: $\mathbf{I}_{\text{pyr}}^{k} \leftarrow$ CreateImagePyramid($\mathbf{I}^k$)
2: $\mathbf{I}_{\text{pyr}}^{k} \leftarrow$ BlurImagePyramid($\mathbf{I}_{\text{pyr}}^{k}$)

3: $\mathbf{U} \leftarrow$ TrackFeatures($\mathbf{I}_{\text{pyr}}^{k}, \mathbf{P}_{\text{pyr}}^{k-1}$) (according to algorithm 1)

4: $\mathbf{P}_{\text{pyr}}^{k} \leftarrow$ ConstructPatchPyramid($\mathbf{I}_{\text{pyr}}^{k}, \mathbf{U}$)
```

As seen in algorithm 1, section 4.5.5, the data of the patches from the first image need not be re-sampled and are constant during the algorithm; they can thus be precomputed from a set of features, which happens when the patch pyramid is constructed (step 4 in algorithm 5). This allows only storing MLP_s^2 bytes of data from the first image, where M is the number of features, L is the number of pyramid levels, and P_s is the patch size. With a patch size of 7, 5 pyramid levels and N features, this results in $M \cdot 5 \cdot 7^2 = 245M$ bytes of data.

In the implementation from the specialisation project, borders around the patches were used to allow convolution at the edge of the patch and sub-pixel accuracy through bilinear filtering. This can be observed in figure 8.2, where the red square is the patch, which needs the data in the blue square to apply the blur without shrinking in size. However, since bilinear filtering is used, all the values in the blue square need to be blurred, thus requiring an additional border *during creation* to not shrink in size when convoluted, which in the illustration is the green square. Thus, the final size of the patch will consist of the blue square, but the values within the green square are only used during the initialisation of the patch.



$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{pmatrix}$$

Figure 8.2: The structure of the patch with the borders.

Floating point numbers are needed for more accurate results when using bilinear filtering, which led to 4 bytes per value in the patch. In addition, a sub-pixel coordinate for the patch's location had to be stored, resulting in an additional 2 floating point numbers per patch. With this, the amount of bytes required for N patches with 5 pyramid levels became $M \cdot 5 \cdot ((7+2)^2 + 2) \cdot 4 = 1660M$.

Results From the Specialisation Project

When compared with an OpenCV implementation, the custom implementation yielded the same track for 83% of the features over the whole EuRoC dataset, having an average runtime of 23.8 ms for an average of 20.3 features. However, comparing this way did not state how accurate the implementation in reality was. This was not the case with the FAST implementation since there is a fixed set of rules for the FAST pattern detection, which could be verified. Therefore, for this master's thesis, the Lucas-Kanade implementation was tested against the KITTI flow 2015 dataset [7, 8].

Eigenvalue Rejection

Following the work in the specialisation project, an eigenvalue check was imposed for the structure tensor to make the implementation more robust, as outlined in section 4.5.5 and figure 4.9. However, as shown in the results given by figure 10.1 and table 10.1, the custom implementation for Lucas-Kanade was found to be less robust than OpenCV's implementation. Some attempts with affine transformations within the tracker were made to alleviate this, but they did not yield improved results.

8.3.3 Rotated BRIEF Outlier Rejection

In order to reject outliers, the frontend incorporates rotated BRIEF descriptors, making it possible to detect when a track divergence occurs. The BRIEF implementation uses a patch size of 31×31 and a pattern size of 256 pixel coordinate pairs, yielding a descriptor length of 256 bits. To still support extracting feature candidates near the edges of the image, a reflected border around the image is often used, alleviating that the patch can reach outside the boundary of the original image. An example of this can be seen in Figure 8.3. Regularly, this is achieved by storing a separate image with an appended border. This implementation of BRIEF separates itself from the conventional implementations by doing on-the-fly reflection instead.

Image edge

…	60	105	140	85	140	105	60	…
…	45	30	40	35	40	30	45	…
…	30	40	25	14	25	40	30	…

Figure 8.3: Reflection demonstration, where the elements left of the edge are the reflection of the elements right of the edge.

On-the-fly Reflection

Instead of storing an image with an appended border, this implementation of BRIEF does on-the-fly reflection. This is motivated by the fact that the i.MX RT1170 does not have enough memory for a separate image in the faster memory regions without shuffling some of the other data the frontend and backend use around, yielding that it would have had to be stored in SDRAM. By doing on-the-fly reflection, the implementation can operate on the same image data as the rest of the frontend, which resides in DTCM. This makes computing the BRIEF descriptors faster even though there is an extra instruction cost due to the on-the-fly reflection.

Let $\mathbf{p}_r$ be the reflected coordinate and $\mathbf{p}$ be the original coordinate. Here, the origin of the image is given by the top-left corner. Reflection along the left and top edge in the image thus becomes:

$$\mathbf{p}_r = -\mathbf{p} \tag{8.2}$$

For the right and bottom edges of the image, the width and height of the image have to be taken into consideration. Let W be the width and H be the height. Building from the fact that the distance to the edge has to be equal for both points, the reflected point becomes what is shown in equation (8.3) (note here that zero indexing is used, such that the horizontal edge is given by $x = W - 1$ and the vertical edge is given by $y = H - 1$).

$$\begin{aligned} \begin{bmatrix} W-1 \\ H-1 \end{bmatrix} - \mathbf{p}_r &= \mathbf{p} - \begin{bmatrix} W-1 \\ H-1 \end{bmatrix} \\ \mathbf{p}_r &= 2\begin{bmatrix} W-1 \\ H-1 \end{bmatrix} - \mathbf{p} \end{aligned} \tag{8.3}$$

By utilising equation (8.2) and equation (8.3), reflection can thus be done on-the-fly. It should be stated that the extra instruction cost could yield this being in-favourable for a much greater set of features. That said, the on-the-fly reflection is a small part of the total amount of instructions, with the cost of 16 instructions per reflection. This yields that in a hypothetical case where reflection happens for all the coordinates of the BRIEF pattern, the total number of additional instructions would be $16 \cdot M \cdot 512 = 8192M$. Here M is the maximum number of features, and 512 comes from the number of pixel coordinates in the BRIEF pattern.

Comparing this against pre-computing the image with an appended border — with a required size of 31 pixels due to the patch size — the extra instruction cost from constructing the borders becomes $2 \cdot 752 \cdot 31 \cdot 16 + 2 \cdot 480 \cdot 31 \cdot 16 = 1222144$ instructions. Here, the resolution of the images in the EuRoC dataset is used: 752×480, and the multiplication by two reflects that a border has to be constructed for both vertical and horizontal edges. Thus, roughly, this leads to ≈ 149 features. However, this estimate is far from the truth, as it does not consider the extra clock cycles required for copying over the rest of the image. Moreover, this also does not consider the increased memory access time for the slower external SDRAM.

The on-the-fly reflection was tested against conventional pre-computing of the reflected image (stored in SDRAM), with a maximum of 50 features in the frontend. This reduced the computational time of the whole frontend (doing feature extraction, tracking and outlier rejection with the BRIEF implementation) by 9 ms: from 31 ms to 22 ms.

8.3.4 High-Level Overview

The high-level overview of the frontend implemented can be observed in algorithm 6. Note that when features are tracked, the features from the previous frame are implicit in the patch pyramid $\mathbf{P}_{\text{pyr}}^{k-1}$, as discussed in section 8.3.2. Here, $\mathbf{D}$ denotes the BRIEF descriptors. Before features are tracked, the angular velocity between the previous and the current camera frame is integrated to provide an estimate for where the features are in the current frame. The other variables, which are not inputs or outputs, reside in the frontend module. Moreover, as shown in the overview, feature extraction considers the maximum amount of features allowed in the pipeline, denoted by M_{max}.

Algorithm 6 High-level overview of frontend

```
Input: $\mathbf{I}^k$, image at time step $k$
Input: $\mathbf{B}^k$, IMU measurements between previous and current frame
Output: $\mathbf{U} = \begin{bmatrix}\mathbf{u}_0 & \dots & \mathbf{u}_M\end{bmatrix}$, list of feature observations

 1: $\mathbf{I}_{\text{pyr}}^k \leftarrow$ CreateImagePyramid($\mathbf{I}^k$)
 2: $\mathbf{I}_{\text{pyr}}^k \leftarrow$ BlurImagePyramid($\mathbf{I}_{\text{pyr}}^k$)

 3: $\mathbf{R}_{b_k b_{k-1}} \leftarrow$ IntegrateIMUData($\mathbf{B}^k$)
 4: $\mathbf{U} \leftarrow$ TrackFeatures($\mathbf{I}_{\text{pyr}}^k, \mathbf{P}_{\text{pyr}}^{k-1}, \mathbf{R}_{b_k b_{k-1}}$)
 5: $\mathbf{U} \leftarrow$ FindInliersUsingBRIEF($\mathbf{U}, \mathbf{D}$)

 6: if Length($\mathbf{U}$) $< M_{\text{max}}$ then
 7:     $\mathbf{U}^k \leftarrow$ ExtractFeatures($\mathbf{I}^k, M_{\text{max}} -$ Length($\mathbf{U}$))
 8:     $\mathbf{D} \leftarrow \mathbf{D} \cup$ ConstructBRIEFDescriptors($\mathbf{I}^k, \mathbf{U}^k$)
 9:     $\mathbf{U} \leftarrow \mathbf{U} \cup \mathbf{U}^k$
10: end if

11: $\mathbf{P}_{\text{pyr}}^k \leftarrow$ ConstructPatchPyramid($\mathbf{I}_{\text{pyr}}^k, \mathbf{U}$)
```

8.4 Backend

The backend implemented for the VIO pipeline is based mainly on the LARVIO [2, 3] project, which is an MSCKF implementation including a static initialiser, IMU-camera extrinsics estimation, time stamp compensation and stationary detection.

8.4.1 High-Level Overview

The overview of the backend can be observed in algorithm 7, where $\mathbf{S}_{\text{imu}}$ is the IMU state, $\mathbf{S}_{\text{aug}}$ is the set of augmented IMU states, and $\mathbf{T}$ is the set of *feature tracks*. Each feature track contains a list of observations of the given feature. The feature observations passed from the frontend are not only coordinates; they have an identifier to add the observation to the corresponding feature track.

Algorithm 7 High-level overview over the backend

```
Input: $\mathbf{U}$, list of feature observations from frontend
Input: $\mathbf{B}^k$, IMU measurements between previous and current frame
 1: if not initialised then
 2:     $(\text{initialised}, \mathbf{S}_{\text{imu}}) \leftarrow \text{AttemptStaticInitisliation}(\mathbf{U}, \mathbf{B}^k)$
 3: end if

 4: if initialised then
 5:     $\mathbf{S}_{\text{imu}} \leftarrow \text{PropagateIMUState}(\mathbf{S}_{\text{imu}}, \mathbf{B}^k)$
 6:     $\mathbf{S}_{\text{aug}} \leftarrow \mathbf{S}_{\text{aug}} \cup \mathbf{S}_{\text{imu}}$

 7:     $\mathbf{T} \leftarrow \text{AddFeatureObservations}(\mathbf{T}, \mathbf{U})$
 8:     $\mathbf{S}_{\text{aug}} \leftarrow \mathbf{S}_{\text{aug}} \setminus \text{FindUnusedStates}(\mathbf{S}_{\text{aug}}, \mathbf{T})$

 9:     if IsStationary($\mathbf{T}$) then
10:         $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow \text{StationaryUpdate}(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}})$
11:     end if

12:     $\mathbf{T}_{\text{update}} \leftarrow \text{FindFeatureTracksReadyForUpdate}(\mathbf{T})$
13:     if $\mathbf{T}_{\text{update}} \neq \emptyset$ then
14:         $\mathbf{S}_a \leftarrow \text{FindAssociatedAugmentedStates}(\mathbf{T}_{\text{update}})$
15:         $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow \text{MeasurementUpdate}(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}, \mathbf{S}_a, \mathbf{T}_{\text{update}})$
16:     end if

17:     if Length($\mathbf{S}_{\text{aug}}$) $= N_{\text{max}}$ then
18:         $\mathbf{S}_l \leftarrow \text{FindAugmentedStatesWithLeastMovement}(\mathbf{S}_{\text{aug}})$
19:         $\mathbf{T}_a \leftarrow \text{FindAssociatedFeatures}(\mathbf{S}_l)$
20:         $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow \text{MeasurementUpdate}(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}, \mathbf{S}_l, \mathbf{T}_a)$
21:     end if
22: end if
```

Static Initialisation

```
**if not** initialised **then**
    $(\text{initialised}, \mathbf{S}_{\text{imu}}) \leftarrow \text{AttemptStaticInitisliation}(\mathbf{U}, \mathbf{B}^k)$
**end if**
```

The backend initialises by means of static initialisation, requiring the system to be stationary initially. The static initialisation was set to require at least 20 stationary frames. It looks at the Euclidean distance between the feature observations at the first frame and the current frame to determine whether the system is static. If the system is not deemed static, the initialisation is reset.

If the system is determined static, the average of the measured accelerations from the IMU measurements is calculated. This is used to construct the initial orientation estimate by comparing it against the assumed world frame gravity vector. Moreover, an initial bias for the angular velocity is estimated by the average measured angular velocity during the static frames.

Propagating & Augmenting the State

```
$\mathbf{S}_{\text{imu}} \leftarrow \text{PropagateIMUState}(\mathbf{S}_{\text{imu}}, \mathbf{B}^k)$
$\mathbf{S}_{\text{aug}} \leftarrow \mathbf{S}_{\text{aug}} \cup \mathbf{S}_{\text{imu}}$
```

During the propagation step, the set of IMU measurements $\mathbf{B}^k$ between the previous and current camera frames is processed. The propagation of the nominal IMU state is done by means of Runga-Kutta numerical integration (Appendix B) and follows the dynamics outlined in section 6.3. The IMU state covariance is propagated by finding the error-state transition matrix, as outlined in section 6.5.2.

After the propagation, the current estimate for the IMU state $\mathbf{S}_{\text{imu}}$ is augmented into $\mathbf{S}_{\text{aug}}$ according to section 6.6.

Adding Feature Observations & Removing Unused States

```
$\mathbf{T} \leftarrow \text{AddFeatureObservations}(\mathbf{T}, \mathbf{U})$
$\mathbf{S}_{\text{aug}} \leftarrow \mathbf{S}_{\text{aug}} \setminus \text{FindUnusedStates}(\mathbf{S}_{\text{aug}}, \mathbf{T})$
```

When new feature observations arrive, they are iterated through and added to the feature tracks $\mathbf{T}$. The backend operates with identifiers for the IMU state, yielding that the set of feature observations will be associated with the current state when added. This allows for checking if there are states which do not have any associated features and can be removed to reduce computational cost.

Stationary Detection & Update

```
if IsStationary($\mathbf{T}$) then
    $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow$ StationaryUpdate($\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}$)
end if
```

Stationary detection is based on the same principle as the static initialisation, checking for minimal movement for each feature in the feature tracks. The stationary update follows the derivations in section 6.10.

Updating the Filter With Features

```
$\mathbf{T}_{\text{update}} \leftarrow$ FindFeatureTracksReadyForUpdate($\mathbf{T}$)
if $\mathbf{T}_{\text{update}} \neq \emptyset$ then
    $\mathbf{S}_a \leftarrow$ FindAssociatedAugmentedStates($\mathbf{T}_{\text{update}}$)
    $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow$ MeasurementUpdate($\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}, \mathbf{S}_a, \mathbf{T}_{\text{update}}$)
end if
```

Features are used to update the filter under the given conditions:

- There are features which have been tracked past the maximum track length and can be triangulated.
- There are features which have gone out of frame (out of the image) and can be triangulated.

If the feature has gone out of frame and cannot be triangulated, it will be discarded. As previously mentioned, due to associating feature observations with augmented IMU state identifiers, the associated states can be found. This is used to construct the jacobian outlined in section 6.7.1 with the associated null-space projection. The filter is then updated according to the derivations in section 6.8.

Pruning Augmented IMU States

```
if Length($\mathbf{S}_{\text{aug}}$) $= N_{\max}$ then
    $\mathbf{S}_l \leftarrow$ FindAugmentedStatesWithLeastMovement($\mathbf{S}_{\text{aug}}$)
    $\mathbf{T}_a \leftarrow$ FindAssociatedFeatures($\mathbf{S}_l$)
    $(\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}) \leftarrow$ MeasurementUpdate($\mathbf{S}_{\text{imu}}, \mathbf{S}_{\text{aug}}, \mathbf{S}_l, \mathbf{T}_a$)
end if
```

A forced filter update is performed if the set of augmented IMU states is at the maximum allowed capacity for the sliding window. It is based on finding augmented IMU states with little movement from a reference state. All the features observed from these states are then used in the forced filter update.

8.4.2 Adaptations & Optimisations

This implementation of an MSCKF backend separates itself from conventional implementations by the following aspects:

- Explicitly placing heavily used data in fast memory regions
- Chaining the error-state transition matrix to reduce matrix multiplication
- Using a custom allocator for Eigen

Moreover, as with other implementations such as OpenVINS [4], the implementation makes heavy use of the triangular functionality in Eigen to limit the computations needed for symmetric matrices such as the state covariance matrix.

The following sections will demonstrate how these changes decreased the runtime of the backend. The *Machine Room 1* part of the EuRoC dataset [9] was used as a baseline for the runtime of the backend (with a maximum of 50 features and a maximum sliding window of 15). The runtime of the static initialisation is omitted from the analysis due to not happening on a frequent interval in the backend and being a small contribution to the overall runtime. The backend had an initial RMSE of ≈ 0.42m and a final percentage error of 0.92% of the total track length.

The reader should be aware that BRIEF was not used for outlier rejection in the frontend during these tests due to being incorporated into the implementation later. This does not, however, have any implications on where the major bottlenecks in the backend were. The initial runtime can be observed in table 8.5, where there was no use of TCM.

Component	**Average Time (ms)**	**Max Time (ms)**
Propagating state	20	47
Augmenting state	0	3
Adding feature observations	0	1
Removing unused states	0	3
Stationary detection & update	0	5
Updating filter with features	30	342
Pruning augmented IMU states	1	82
Backend in total	53	442

Table 8.5: Initial runtime of the backend.

In the results, the great spikes in processing time occurred due to the filter update, where QR decomposition and matrix multiplication were the two main contributors to increased runtime.

It should be noted that due to how the timing was calculated, using the *SysTick* component of the microcontroller instead of the DWT as was done with the frontend in the preceding specialisation project, there are some slight inaccuracies which can be observed with the total sum not being equal to the summed average of the components as well as the restricted resolution to only 1 ms, which the reader should bear in mind in the following sections. The reason for doing the timing this way was to have nested timing: testing the range of components in the backend whilst also profiling the backend as a whole, which would not be possible with the DWT. This was not deemed a problem for determining where optimisations had to be made, as the estimates provided a thorough insight into the bottlenecks. It should also be noted that the total time of the backend not adding up to the individual components' maximum is due to the maximum of the individual components' runtime happening at different instances in time.

Explicitly Placing Heavily Used Data in Fast Memory Regions

The implementation stores the state covariance in DTCM as it is frequently used in the backend. The total size of the DTCM configured for the implementation is 480 KB, where $752 \cdot 480 = 360960$ bytes is reserved for the image being processed in the frontend, and 32 KB is reserved for the stack. The total size of the state covariance is given by:

$$T = (22 + 6N) \cdot (22 + 6N) \cdot 4 \text{ bytes} \tag{8.4}$$

where the IMU state occupies 22 entries and N is the number of augmented IMU states, as outlined in section 6.1.3. Moreover, the size is multiplied by 4 due to using 32-bit floating point numbers. With a sliding window of $N = 15$, this results in a maximum of:

$$T = (22 + 6 \cdot 15) \cdot (22 + 6 \cdot 15) \cdot 4 = 50176 \text{ bytes} \tag{8.5}$$

which leaves 47616 bytes unused in DTCM. To use a static buffer with the Eigen matrices, Eigen's map functionality is used in the implementation. Incorporating this resulted in the runtime which can be observed in table 8.6, where one can see the main contribution to the decreased runtime came from the propagation step, where the state covariance is heavily used in matrix multiplication. Furthermore, one can also observe a decreased runtime for the components updating of the filter, where the state covariance is also used.

Component	**Average Time [ms]**	**Max Time [ms]**
Propagating state	15	27
Augmenting state	0	2
Adding feature observations	0	1
Removing unused states	0	1
Stationary detection & update	0	4
Updating filter with features	28	338
Pruning augmented IMU states	1	62
Backend in total	46	396

Table 8.6: Runtime of the backend after placing the state covariance matrix in DTCM

Exploiting Symmetric Matrices

As with OpenVINS [4], this implementation makes heavily use of the triangular functionality in Eigen to reduce matrix multiplication, effectively allowing to only calculate the upper triangular part of a matrix and then mirroring the result for the lower triangular part. Making use of this resulted in the runtime observed in table 8.7, where one can observe this change affecting mainly the components related to the update step of the filter.

Component	**Average Time [ms]**	**Max Time [ms]**
Propagating state	14	26
Augmenting state	0	2
Adding feature observations	0	1
Removing unused states	0	1
Stationary detection & update	0	4
Updating filter with features	19	216
Pruning augmented IMU states	0	31
Backend in total	35	262

Table 8.7: Runtime of the backend after placing intermediary matrix products in a buffer in DTCM and using triangular matrix multiplication for symmetric matrices

Chaining the Error-state Transition Matrix to Reduce Matrix Multiplication

As outlined in the section 6.5, when the state covariance is predicted, the transition matrix $\mathbf{\Phi}(t_{k+1}, t_k)$ is calculated. The transition matrix has the following property (see [70], lemma 1173):

$$\mathbf{\Phi}(t_{k+n}, t_k) = \mathbf{\Phi}(t_{k+n}, t_{k+(n-1)})\mathbf{\Phi}(t_{k+(n-1)}, t_{k+(n-2)})\ldots\mathbf{\Phi}(t_{k+1}, t_k) \tag{8.6}$$

The transition matrix between error-state estimates can thus be chained, allowing for only needing to calculate the chain when propagating the covariance from a range of IMU measurements between two camera frames. This allows for a single propagation of the state covariance with the chained transition matrix instead of propagating it for every measurement. With this, equation (6.63) becomes:

$$\begin{aligned}\mathbf{P}^-_{II_{k+n}} &= \mathbf{\Phi}(t_{k+n}, t_k)\mathbf{P}_{II_k}\mathbf{\Phi}(t_{k+n}, t_k)^T \\ &+ \mathbf{\Phi}(t_{k+n}, t_k)\mathbf{G}(t_k)\mathbf{Q}(t_k)\mathbf{G}(t_k)^T\mathbf{\Phi}(t_{k+n}, t_k)^T\Delta t \cdot n\end{aligned} \tag{8.7}$$

This reduces the amount of matrix multiplication when predicting the nominal state and the covariance. The result from incorporating this into the implementation can be seen in table 8.8, where one can observe that this change led to an 11 ms decrease in runtime for propagating the state on average. This does yield a numerically different system as a whole. However, this was found not to hurt the filter's accuracy, achieving an RMSE of 0.39 m and a final percentage error of 0.80% of the total track length, which was an improvement from the initial system. However, these values should be taken with a grain of salt but show that the filter was behaving similarly. It should also be noted that the maximum value of the filter increased, which came from the slightly changed numerical values in the filter from time point to time point, leading to changes in the update step of the filter.

Component	**Average Time [ms]**	**Max Time [ms]**
Propagating state	3	4
Augmenting state	0	2
Adding feature observations	0	1
Removing unused states	0	1
Stationary detection & update	0	4
Updating filter with features	19	261
Pruning augmented IMU states	0	34
Backend in total	24	284

Table 8.8: Runtime of the backend after using the transition matrix chain to reduce matrix multiplication

Custom Allocator

The implementation uses a custom allocator prioritising for faster memory regions for Eigen's dynamic matrices. This significantly speeds up all use of Eigen in the implementation. Regularly, the data for these matrices are stored on the heap. For the implementation, this became problematic due to two reasons: both DTCM and OCRAM have rather limited capacity (yielding that storing the heap in one of them can result in running out of space) and storing the heap on the external SDRAM results in slow access times for the matrix products. This implementation thus uses a custom allocator to alleviate this. The proposed custom allocator allows for spanning a heap over two distinct memory regions, which in the implementation is used to prioritise allocations in DTCM and fallback to OCRAM.

The custom allocator is not a replacement for the system's dynamic memory (which the respective calls to *malloc*, *realloc* and *free*). Instead, the implementation uses a modified version of Eigen that allows for calling an externally defined custom allocator. This allows for not limiting the allocator to Eigen solely.

The allocator is built with redundancy in mind, allowing, as stated, to fall back to slower memory with more capacity if the faster memory reaches full capacity. The custom allocator is designed as a standard heap, allowing for allocating, reallocating and freeing memory. Moreover, it is implemented as a doubly linked list of nodes spanning over the two memory regions, which can be observed in figure 8.4.

The allocator is implemented after a *first-fit* strategy with splitting, where an allocation will be placed in the first available/unused block when traversing the doubly linked list, with a split of the block occurring after to maintain space efficiency. This strategy mimics a *best-fit* strategy, where the block with the lowest capacity but still

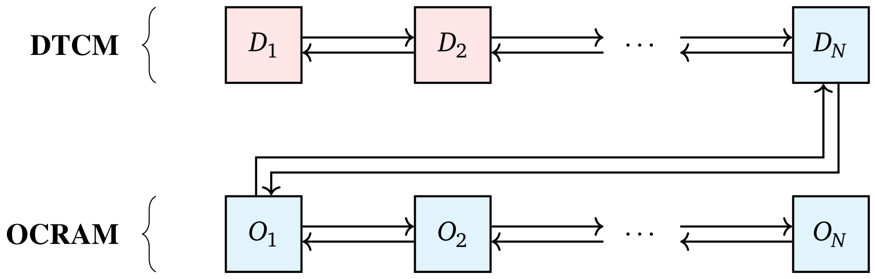


Figure 8.4: Structure of allocator spanning over the two memory regions DTCM and OCRAM, with a doubly linked list combining the blocks. The memory write and access speed decreases downward. The used blocks are denoted in red, whereas the unused blocks are denoted in blue. Note that the blocks here are of equal size, which is only for illustration purposes, as the blocks can greatly vary in size.

enough space for the allocation is chosen. However, best-fit has to traverse the whole doubly linked list due to this aspect. Both strategies have an algorithmic runtime of $\mathcal{O}(n)$, whereas the first-fit will, on average, be lower. There are various other allocation strategies, but first-fit with splitting was chosen due to firstly being fast and secondly being space efficient, which is one of the primary goals of the allocator: placing as much of the data in DTCM. The first-fit with splitting strategy can be observed in figure 8.5.

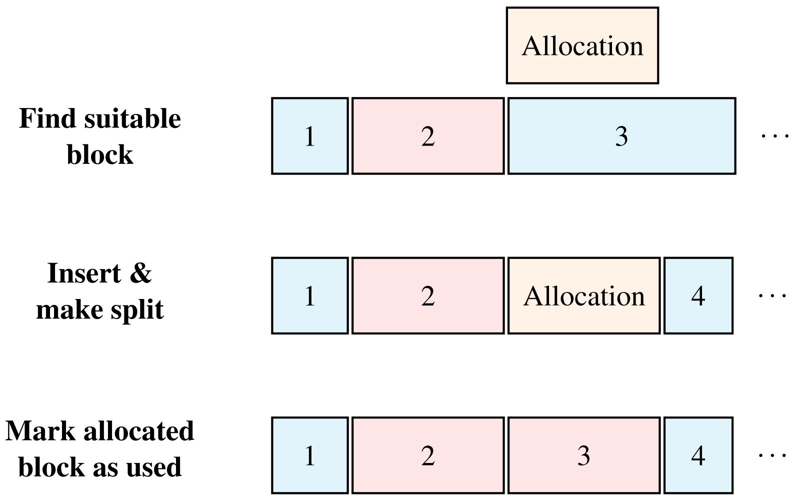


Figure 8.5: Search for available space for allocation in first-fit with splitting.

Functionality for merging blocks is also incorporated into the allocator to have greater space efficiency, where neighbouring unused blocks are merged when a free occurs, as demonstrated in figure 8.6. This prevents the heap from being split into smaller and smaller pieces, converging to a state where larger allocations cannot happen.

When laid out in memory, each block needs metadata for the size of the data, whether it is used or available for allocation and pointers for the previous and the

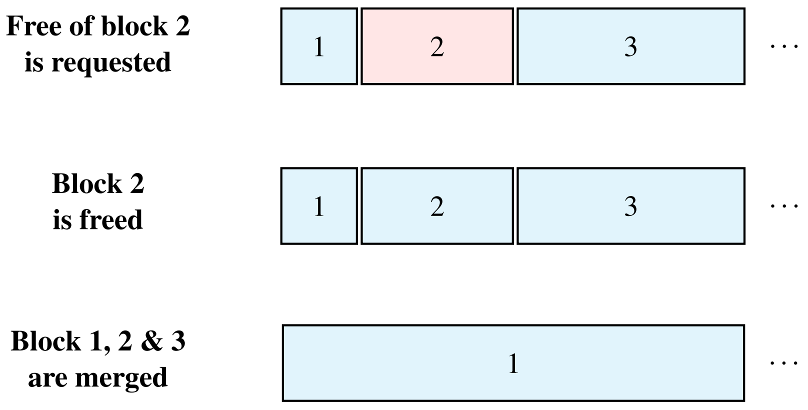


Figure 8.6: Merging unused blocks after a free.

next block. Since the architecture is 32-bit, each pointer occupies 4 bytes. The size variable is 32-bit, and the boolean flag for whether the block is used or not is 8-bit. However, due to alignment, which the compiler introduces for faster access times, the boolean flag will be padded with 3 bytes. This results in the metadata for each block being 16 bytes, which can be observed in figure 8.7. It should be stressed that this metadata is stored along with the actual data in the respective memory regions.

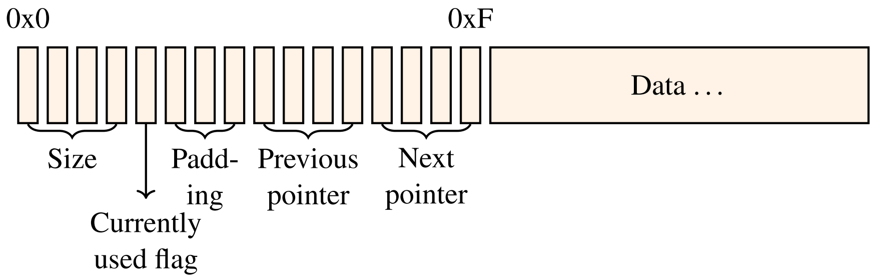


Figure 8.7: Structure of a block.

For the allocations to not grow beyond the capacity of the allocator, it needs to have the capacity within one memory region at least as big as the scope with the largest amount of allocation. This was found by examining the matrix sizes, mainly during the filter update, where the most dynamic data is allocated. Before the update step of the filter, the jacobian $\mathbf{H}_{\hat{\mathbf{x}},n}$ (equation (6.81)) and the corresponding residual $\mathbf{r}_n$ is constructed, both which can have a maximum of $2M_{\max} - 3$ rows, where $M_{\max}$ is the maximum number of feature observations. Furthermore, the jacobian can have a maximum of $22 + 6N_{\max}$ columns, where $N_{\max}$ is the maximum allowed size of the sliding window.

During the update step, a QR decomposition is performed if the number of rows of the jacobian is greater than the number of columns, reducing the maximum number of rows to $N^*_{\max} := 22 + 6N_{\max}$ (and reducing the allocated data) for both the jacobian and the residual. With that being said, the resize will require an allocation of the new resized data before the original data is moved to the resized data, with a maximum size of $N^*_{\max} \cdot N^*_{\max}$ for the jacobian and $N^*_{\max}$ for the residual. Here, this updated jacobian is given by $\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}$ and the corresponding residual is given by $\mathbf{r}_Q$ (see section 6.8.2). Furthermore, when the Kalman gain is computed, the inverse innovation covariance is found (equation (6.92)):

$$\mathbf{S}_{k+1} = \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \mathbf{P}^-_k \mathbf{T}^T_{\mathbf{H}_{\hat{\mathbf{x}},n}} + \sigma^2_{im} \mathbf{I}_{\mathrm{rows}(\mathbf{Q}^T_1) \times \mathrm{rows}(\mathbf{Q}^T_1)} \tag{8.8}$$

The calculation of the Kalman gain for when the maximum amount of memory is used is then given by the procedure shown in algorithm 8.

Algorithm 8 Calculation of Kalman gain

```
1: $\mathbf{S}^{-1}_{k+1} \leftarrow \mathrm{Allocate}(N^*_{\max}, N^*_{\max})$
2:     $\mathbf{S}_{k+1} \leftarrow \mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}} \mathbf{P}^-_k \mathbf{T}^T_{\mathbf{H}_{\hat{\mathbf{x}},n}}$
3:     $\mathbf{S}_{k+1} \leftarrow \mathbf{S}_{k+1} + \sigma^2_{im} \mathbf{I}_{N^*_{\max} \times N^*_{\max}}$
4:     $\mathbf{S}^{-1}_{k+1} \leftarrow \mathrm{Inverse}(\mathbf{S}_{k+1})$
5: $\mathbf{K}_{k+1} = \mathbf{P}^-_k \mathbf{T}^T_{\mathbf{H}_{\hat{\mathbf{x}},n}} \mathbf{S}^{-1}_{k+1}$
```

During step 2, two allocations occur: one for the transpose of $\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}$ and one to store the result itself. During step 3 one allocation occurs for $\mathbf{I}_{N^*_{\max} \times N^*_{\max}}$, as the addition is performed in-place. However, during these steps, the intermediary matrices given by the transpose and the identity are freed directly after the operation. As shown by the indentation in algorithm 8, these allocations are placed within a nested scope, such that $\mathbf{S}_{k+1}$ is deallocated when this scope is left and only the allocated data of $\mathbf{S}^{-1}_k$ is left. Thus, to summarise for calculation of the innovation covariance, only two extra matrix allocations of maximum size $N^*_{\max} \cdot N^*_{\max}$ are active at any time.

During the calculation of the Kalman gain in step 5, the allocation of the data for the Kalman gain itself, and the transpose of the measurement jacobian occurs. The transpose is, however, freed instantly after the product is calculated. Thus, for the calculation of the Kalman gain, the maximum allocated data at any time point is given by $3 \cdot N^*_{\max} \cdot N^*_{\max}$.

Furthermore, when the state covariance is updated according to equation (6.94), it is updated in-place, such that only one allocation for an intermediary product occurs, with the same maximum size as the preceding matrices. At this point, both $\mathbf{S}^{-1}_{k+1}$ and $\mathbf{K}_{k+1}$ are in scope, and only one extra matrix is allocated since the state covariance is mapped to a buffer in DTCM residing outside the allocator, as discussed in section 8.4.2. This yields again a maximum of $3 \cdot N^*_{\max} \cdot N^*_{\max}$.

This concludes that a rough upper bound for the capacity needed by the allocator is given either by the allocated data before and during the QR decomposition or after. In the following, the multiplication by 4 is needed as the matrices use 32-bit floating point numbers. Moreover, due to the allocation happening during the resize in the

QR decomposition, extra entries have to be added with the maximum sizes of for the reduced jacobian $\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}$ and the reduced residual $\mathbf{r}_Q$.

$$\begin{aligned}
A_{\max} &= 4 \cdot \max\Big(\max \text{size}(\mathbf{H}_{\hat{\mathbf{x}},n}) + \max \text{size}(\mathbf{r}_n) + \max \text{size}(\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}) + \max \text{size}(\mathbf{r}_Q), \\
&\qquad \max \text{size}(\mathbf{T}_{\mathbf{H}_{\hat{\mathbf{x}},n}}) + \max \text{size}(\mathbf{r}_Q) + 3 \cdot (N^*_{\max} \cdot N^*_{\max})\Big) \\
&= 4 \cdot \max\Big((2M_{\max} - 3)(22 + 6N_{\max}) + (22 + 6N_{\max})^2 + 2 \cdot (22 + 6N_{\max}), \\
&\qquad (N^*_{\max} \cdot N^*_{\max}) + N^*_{\max} + 3 \cdot (N^*_{\max} \cdot N^*_{\max})\Big) \\
&= 4 \cdot \max\Big((2M_{\max} - 1)(22 + 6N_{\max}) + (22 + 6N_{\max})^2, \\
&\qquad 4 \cdot (22 + 6N_{\max})^2 + (22 + 6N_{\max})\Big)
\end{aligned} \tag{8.9}$$

The implementation uses a sliding window of 15 and a maximum number of features in the frontend set to 50. However, to allow overlap between the frames, e.g. in the edge case of all the current features going out of frame and the frontend extracting 50 new features in the same frame, the maximum size of the features in the backend needs to be $2 \cdot 50$. However, this does not imply that the maximum size of e.g. the measurement jacobian has to abide by this, as the number of tracked features will still only be 50, and only tracked features are passed to the update step of the filter. Thus, the maximum number of observations in total becomes $M_{\max} = 50 \cdot 15 = 750$. This is thus the extreme case where 50 features have been tracked throughout the whole sliding window, and all features pass the outlier rejection test (as discussed in section 6.8.1). The rough estimate for the maximum needed capacity thus becomes:

$$\begin{aligned}
A_{\max} &= 4 \cdot \max\Big((2 \cdot 750 - 1)(22 + 6 \cdot 15) + (22 + 6 \cdot 15)^2, \\
&\qquad 4 \cdot (22 + 6 \cdot 15)^2 + (22 + 6 \cdot 15)\Big) \\
&= 4 \cdot \max\Big(180432, 50288\Big) \\
&= 721728 \text{ bytes}
\end{aligned} \tag{8.10}$$

The reader should be aware that this is an *estimate* of the upper boundary, as some small allocations are happening when for example the state is updated in the update step of the filter. With that being said, the estimate given by $\mathbf{A}_{\max}$ plus some safety margin for alignment and other smaller allocations provides a rough idea of the needed memory for the allocator. To get a more precise estimate, the update step was performed with the maximum sizes for the measurement jacobian and the residual whilst recording the used capacity of the allocator, yielding an upper bound of 726996 bytes, not far from the calculated value of $A_{\max}$.

It should, however, be emphasised that this estimate is for a continuous memory region. If the primary memory region is less than this estimate, fragmentation will occur over the two memory regions. That said, as long as the OCRAM section — the secondary memory region within the allocator — has at least this capacity, the allocator will not run out of memory.

The runtime with the custom allocator can be observed in table 8.9, where it can be seen that the custom allocator made the backend run 1.5x faster in average and 2.86x faster for the maximum case. Using the custom allocator did not change the algorithm's accuracy — as it should not — where the pipeline still had an RMSE of 0.39 m and a final percentage error of 0.80% of the total track length.

Component	**Average Time [ms]**	**Max Time [ms]**
Propagating state	4	5
Augmenting state	0	1
Adding feature observations	0	1
Removing unused states	0	1
Stationary detection & update	0	3
Removing lost features	11	76
Pruning augmented IMU states	0	29
Backend in total	16	93

Table 8.9: Runtime of the backend with the custom allocator.

This aspect of the implementation signifies how great a bottleneck memory can be and how much there is to gain by utilising the faster memory regions for all they are worth. It should be noted that to allow for most of DTCM to be used for the custom allocator, the image data that the frontend uses was moved from DTCM to OCRAM. This had, however, a negligible effect on the runtime for the frontend, increasing from 21 ms on average to 22 ms and increasing from the maximum of 33 ms to 34 ms (with a FAST threshold of 80 and a maximum of 50 features).

This concludes the optimisations and adaptions made for the backend, yielding a 2.875x speed increase on average and a 4.258x speed increase in the maximum case compared to the initial implementation.

8.5 Closing Remarks

The preceding sections conclude the implementation details, overviews and optimisations made to the VIO pipeline. That said, the reader should be aware that an update rate for the backend was introduced into the implementation to limit the number of calls to the backend. With this, the frontend would operate on a set frequency dictated by the camera rate, whereas the backend could operate on an equal or an arbitrarily lower frequency.

Chapter 9

Test Setup

9.1 Lucas-Kanade Implementation

The Lucas-Kanade implementation was tested against the KITTI flow 2015 dataset [7, 8]. The images from the training dataset were used with the accompanying flow vectors. The calculated ground truth was given by the extracted features' position in the first frame added with the flow vector. OpenCV's FAST implementation was used to have a common baseline for the extracted features, such that solely the Lucas-Kanade implementation was tested. A FAST threshold of 75 was used, with a maximum of 30 iterations in the Lucas-Kanade implementation and a stationary threshold for the tracking at 0.01. Furthermore, a maximum amount of features was set to 1000 to be able to extract all the features in the image at the given FAST threshold. The following parts of the dataset were used:

- Images from the *image 2* folder
- Flow vectors from the *flow noc* folder

9.2 Pipeline

The frontend and backend were tested against the EuRoC [9] dataset, using TCP/IP to stream the images, IMU measurements and the ground truth over the 1 Gbit/s Ethernet link of the i.MX RT1170 evaluation kit. The intrinsics and extrinsics for the *cam0* and *imu0* sensors were used. The following folders were utilised for each part of the dataset:

- Images: the *cam 0* folder
- Ground truth: the *state ground-truth estimate 0* folder, which provides ground-truth for the position, orientation, velocity, gyroscope bias and accelerometer bias
- IMU measurements: the *imu 0* folder

Furthermore, an initialisation frame was set to the current ground truth pose when the filter initialised. With this, the estimate could be transformed into this initialisation frame and operate from the same starting point as the ground truth.

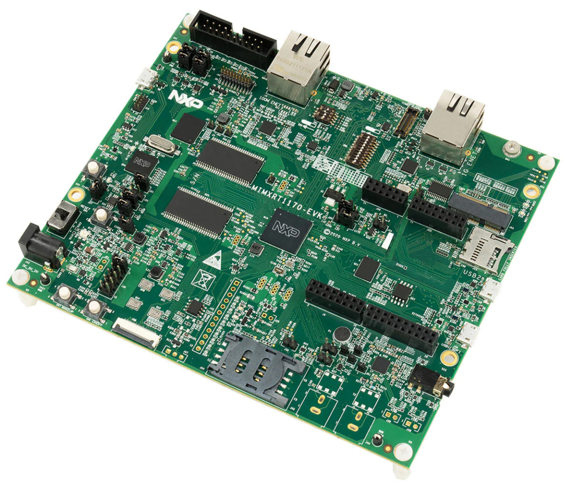


Figure 9.1: The i.MX RT1170 evaluation kit, from [78].

The following metrics were used to evaluate the implementation:

$$
\begin{aligned}
\text{RMSE}_{\text{position}} &= \sqrt{\sum_{k=0}^{K} \frac{||\mathbf{p}_{b_k}^{w} - \hat{\mathbf{p}}_{b_k}^{w}||_2^2}{K}} \\
\text{RMSE}_{\text{orientation}} &= \sqrt{\sum_{k=0}^{K} \frac{||\delta\boldsymbol{\theta}_{wb_k}||_2^2}{K}} \\
\text{RMSE}_{\text{velocity}} &= \sqrt{\sum_{k=0}^{K} \frac{||\mathbf{v}_{b_k}^{w} - \hat{\mathbf{v}}_{b_k}^{w}||_2^2}{K}} \\
\text{RMSE}_{\text{gyroscope bias}} &= \sqrt{\sum_{k=0}^{K} \frac{||\mathbf{b}_{a_k}^{b} - \hat{\mathbf{b}}_{a_k}^{b}||_2^2}{K}} \\
\text{RMSE}_{\text{accelerometer bias}} &= \sqrt{\sum_{k=0}^{K} \frac{||\mathbf{b}_{a_k}^{b} - \hat{\mathbf{b}}_{a_k}^{b}||_2^2}{K}}
\end{aligned}
\tag{9.1}
$$

Here, $\delta\boldsymbol{\theta}_{wb_k}$ is the axis-angle representation of the error-quaternion $\delta\mathbf{q}_{wb_k} = \mathbf{q}_{wb_k} \otimes \hat{\mathbf{q}}_{wb_k}^{-1}$. Moreover, the *average normalised estimated error squared* (ANEES) metric was used, which is directly related to the Mahalanobis distance test (section 5.3.1). Let $\delta\mathbf{x_k}$ be a 3×1 error-vector, then the ANEES is given by:

$$\text{ANEES} = \frac{1}{K}\sum_{k=0}^{K} \delta\mathbf{x}_k^T \mathbf{P}^{-1} \delta\mathbf{x}_k \tag{9.2}$$

Here $\sum_{k=0}^{K} \delta\mathbf{x}_k^T \mathbf{P}^{-1} \delta\mathbf{x}_k \sim \chi^2(3K)$, such that a confidence interval for the ANEES is given by:

$$r_1 = \frac{1}{K}F^{-1}(\frac{\alpha}{2}, 3K) \qquad r_2 = \frac{1}{K}F^{-1}(\frac{\alpha}{2}, 3K) \tag{9.3}$$

where F^{-1} is the inverse cumulative distribution function of the χ^2 distribution and α is the confidence level.

For the results in chapter 10, a FAST threshold of 80, a BRIEF threshold of 70 and a maximum of 50 features were used for the frontend. The backend operated on a sliding window of a maximum of 15 augmented IMU states and an update frequency set to half the frequency of the frontend. Furthermore, runtime results were calculated using the SysTick module on the microcontroller, set to increment every millisecond, yielding a resolution of 1 ms.

9.2.1 Dataset Sections

The dataset sections are categorised by the following:

Section	Description
Machine Hall 1 (MH 01)	Medium displacement, slow movements, bright scene, good texture
Machine hall 2 (MH 02)	Equivalent with MH 01
Machine hall 3 (MH 03)	Medium displacement, fast movements, bright scene, good texture
Machine hall 4 (MH 04)	Large displacement, fast movements, dimly lit scene
Machine hall 5 (MH 05)	Equivalent with MH 04
Vicon room 1 1 (V1 01)	Small displacement, slow movements, medium amount of rotation, bright scene
Vicon room 1 2 (V1 02)	Small displacement, fast movements, high amount of rotation, bright scene
Vicon room 1 3 (V1 03)	Small displacement, fast movements, high amount of rotations, motion blur, exposure changes
Vicon room 2 1 (V2 01)	Medium displacement, slow movements, low amount of rotations, bright scene
Vicon room 2 2 (V2 02)	Medium displacement, fast movements, high amount of fast rotations, bright scene
Vicon room 2 3 (V2 03)	Medium displacement, fast movements, high amount of fast rotations, motion blur, bright scene

Table 9.1: The properties of the various dataset sections in EuRoC.

9.2.2 Memory Usage

The memory usage of the implementation was found by invoking the linker with the *--print-memory-usage* flag.

9.2.3 Power Consumption

The power consumption was measured on the whole 3.3 V domain of the evaluation kit. In particular, across the J41 jumper. The evaluation kit has dedicated measuring points for the MCU and the FLASH power domains. However, this is not the case for the SDRAM. It was therefore deemed better to measure the whole 3.3 V domain to at least get a precise upper boundary, even though this upper boundary will be much higher than for a dedicated PCB (printed circuit board) solution, where only the necessary components are included. The whole 3.3 domain spans the MCU, the external

SDRAM, the various FLASH program memories, the two Ethernet transceivers, the CAN transceiver, the Wi-Fi module, as well as other components.

Chapter 10

Results

10.1 Lucas-Kanade

The following shows the pixel error from the tracking compared with the ground truth from the KITTI flow 2015 dataset [7, 8]. The outliers for both implementations have been omitted from the box plot to provide a clearer picture of the nominal behaviour and can rather be deduced from table 10.1.

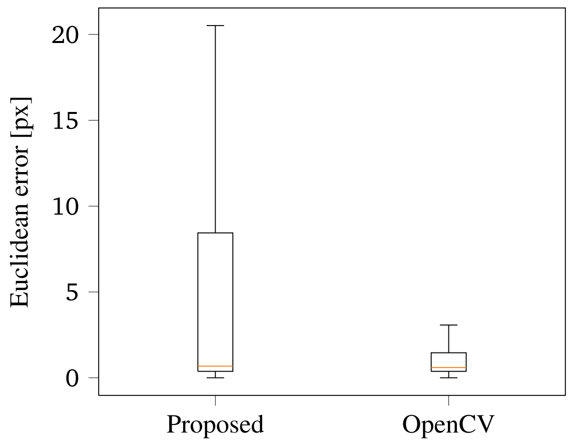


Figure 10.1: Box plot comparing the proposed Lucas-Kanade implementation and OpenCV's Lucas-Kanade implementation.

Implementation	**85%**	**95%**	**100%**
Proposed	≤ 34.0	≤ 85.4	≤ 596.4
OpenCV	≤ 7.8	≤ 57.4	≤ 883.3

Table 10.1: Upper-level quantiles for the proposed Lucas-Kanade implementation and OpenCV's Lucas-Kanade implementation.

10.2 Pipeline Runtime

Dataset section	Average [ms]	Max [ms]	Average FPS	Min FPS	# frames	# frames above 50 ms
MH 01	27.7	79	36.1	12.7	3254	9
MH 02	27.4	52	36.5	19.2	2508	5
MH 03	27.4	58	36.5	17.2	2652	10
MH 04	27.2	75	36.8	13.3	1999	15
MH 05	26.6	69	37.6	14.5	2244	9
V1 01	23.3	86	42.9	11.6	2831	3
V1 02	22.8	61	43.9	16.4	1688	5
V1 03	21.6	63	46.3	15.9	2113	3
V2 01	24.6	52	40.7	19.2	2255	1
V2 02	24.8	56	40.3	17.9	2323	3
V2 03	23.7	65	42.2	15.4	1839	4

Table 10.2: Runtime of the proposed pipeline (frontend + backend) on the dataset sections in EuRoC.

	Frontend		Backend	
Dataset section	Average	Max	Average	Max
MH 01	31.5	49	25.6	49
MH 02	30.0	48	23.2	48
MH 03	26.4	47	23.5	47
MH 04	24.5	48	21.5	48
MH 05	24.6	47	21.6	47
V1 01	15.4	44	13.0	44
V1 02	13.4	50	11.9	50
V1 03	10.8	48	9.8	48
V2 01	17.8	42	15.5	42
V2 02	16.1	45	14.4	45
V2 03	11.8	45	10.1	45

Table 10.3: Metrics for the number of features tracked in the frontend and number of features used in the backend.

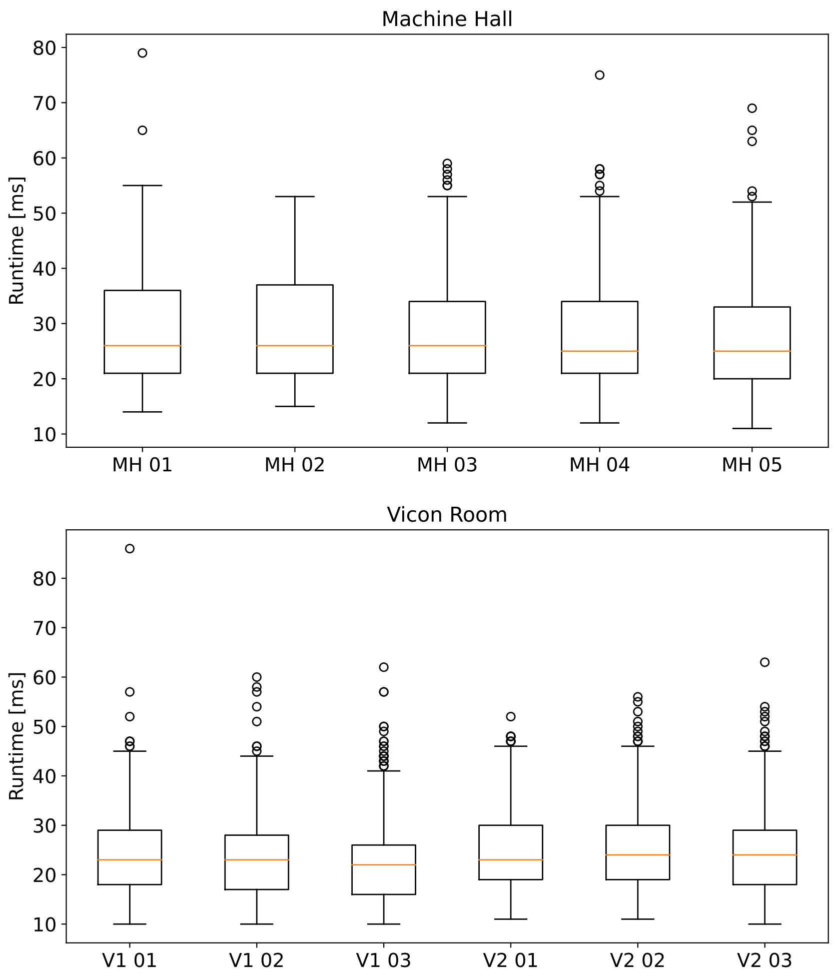
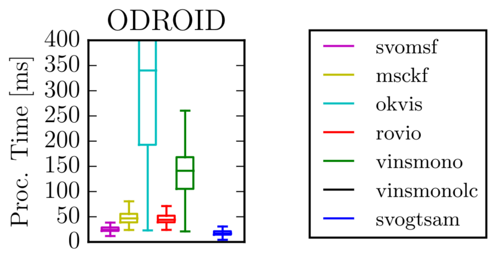


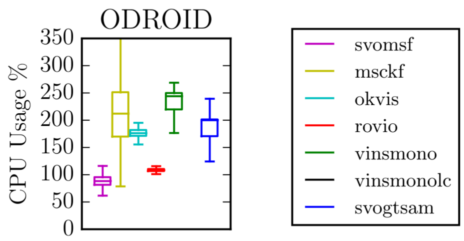
Figure 10.2: Box plot summarising the runtimes for the pipeline in the different dataset sections of EuRoC.

Comparision With State-Of-The-Art

The following comparisons are from Delmerico and Scaramuzza's benchmark comparison paper [35] on state-of-the-art VIO methods. The hardware platform is an ODROID XU4 with a quad-core 1.3 GHz ARM Cortex-A7 and a quad-core 1.9 GHz ARM Cortex-A15 (Samsung Exynos 5 Octa), being one of the most similar architectures to compare against due to the similar instruction set and somewhat comparable clock speed.



(a) Processing time per iteration of the respective comparison pipelines.



(b) CPU utilisation as a percentage of a single core for the respective comparison pipelines.

Figure 10.3: Runtime results for state-of-the-art VIO methods running on an ODROID XU4 [35] (figures are from Delmerico and Scaramuzza's public author version of the paper [79]). Note that the proposed solution runs on one core.

10.3 Pipeline Accuracy & Filter Confidence

	OROID Quad-core 1.3 GHz A7 Quad-core 1.9 GHz A15						**i.MX RT1170 EVK 1 GHz M7 400 MHz M4**
Dataset section	**SVO MSF**	**MSCKF**	**OK-VIS**	**RO-VIO**	**VINS Mono**	**SVO GTSAM**	**Proposed**
MH 01	0.22	0.47	0.15	0.36	**0.13**	0.15	0.28*
MH 02	0.24	0.63	0.20	0.23	0.08	**0.05**	0.40*
MH 03	0.52	0.47	x	0.58	0.58	**0.12**	0.61
MH 04	2.28	0.64	0.42	0.81	**0.12**	x	1.02
MH 05	1.12	0.48	0.62	0.78	0.21	**0.12**	0.71
V1 01	0.43	0.21	0.09	0.15	0.11	**0.07**	0.55
V1 02	0.81	0.21	x	0.24	**0.11**	0.14	0.31
V1 03	x	1.52	x	0.20	**0.11**	x	0.70
V2 01	0.15	0.25	0.11	0.14	**0.08**	0.15	0.18
V2 02	0.46	0.19	0.26	0.17	**0.06**	x	0.22
V2 03	x	1.09	x	0.23	**0.16**	x	0.34

Table 10.4: Absolute translational RMSE in meters for every dataset section in EuRoC compared with VIO pipelines running on an ODROID XU4 from Delmerico and Scaramuzza's benchmark comparison paper [35]. Sections in which the compared methods were unable to finish the dataset section are given by x. For the i.MX RT1170 Evaluation Kit, only the M7 core is utilised. *The pipeline initialised *after* the section of movement in the start of MH 01 and MH 02.

Dataset section	**Orientation [deg]**	**Velocity [m/s]**	**Gyroscope bias [deg/s]**	**Accelerometer bias [m/s^2]**
MH 01	3.20°	0.07	0.056°	0.085
MH 02	1.25°	0.06	0.051°	0.106
MH 03	2.95°	0.24	0.055°	0.064
MH 04	1.40°	1.98	0.049°	0.087
MH 05	1.81°	1.84	0.045°	0.075
V1 01	9.02°	0.89	0.173°	0.446
V1 02	2.50°	1.89	0.085°	0.064
V1 03	6.27°	1.60	0.169°	0.098
V2 01	1.42°	0.06	0.052°	0.063
V2 02	3.57°	0.09	0.107°	0.096
V2 03	3.48°	0.11	0.097°	0.062

Table 10.5: RMSE for orientation, velocity, gyroscope bias and accelerometer bias, calculated according to equation (9.1).

Dataset section	**# of samples**	**90% confidence interval**	**Position**	**Orientation**	**Velocity**	**Gyroscope bias**	**Accelerometer bias**
MH 01	3254	$[2.93, 3.07]$	0.06	57.00	1.01	0.88	2.40
MH 02	2508	$[2.92, 3.08]$	0.13	4.02	0.91	0.26	5.67
MH 03	2652	$[2.92, 3.08]$	0.24	43.39	8.69	0.53	1.85
MH 04	1999	$[2.91, 3.09]$	0.61	5.54	650.00	0.23	4.75
MH 05	2244	$[2.92, 3.09]$	0.31	13.41	574.84	0.38	3.23
V1 01	2831	$[2.92, 3.08]$	0.22	440.02	214.49	6.08	451.41
V1 02	1688	$[2.90, 3.10]$	0.08	30.27	800.98	0.73	1.92
V1 03	2113	$[2.91, 3.09]$	0.38	205.69	489.45	1.94	9.02
V2 01	2255	$[2.92, 3.09]$	0.03	11.25	0.86	0.41	3.36
V2 02	2323	$[2.91, 3.08]$	0.04	76.95	1.72	4.27	35.00
V2 03	1839	$[2.91, 3.09]$	0.08	62.81	2.39	1.01	7.88

Table 10.6: ANEES calculated from the ground truth and the estimated state. Note that this does not include the extrinsics, the time-stamp compensation and augmented IMU states from the state vector due to there not being a ground truth for them. The confidence interval is given by equation (9.3), where K is set to the number of samples (number of camera frames after initialisation) and a 90% confidence level was used. Each entry in the table is a 3×1 vector, where the axis-angle error was used for the orientation.

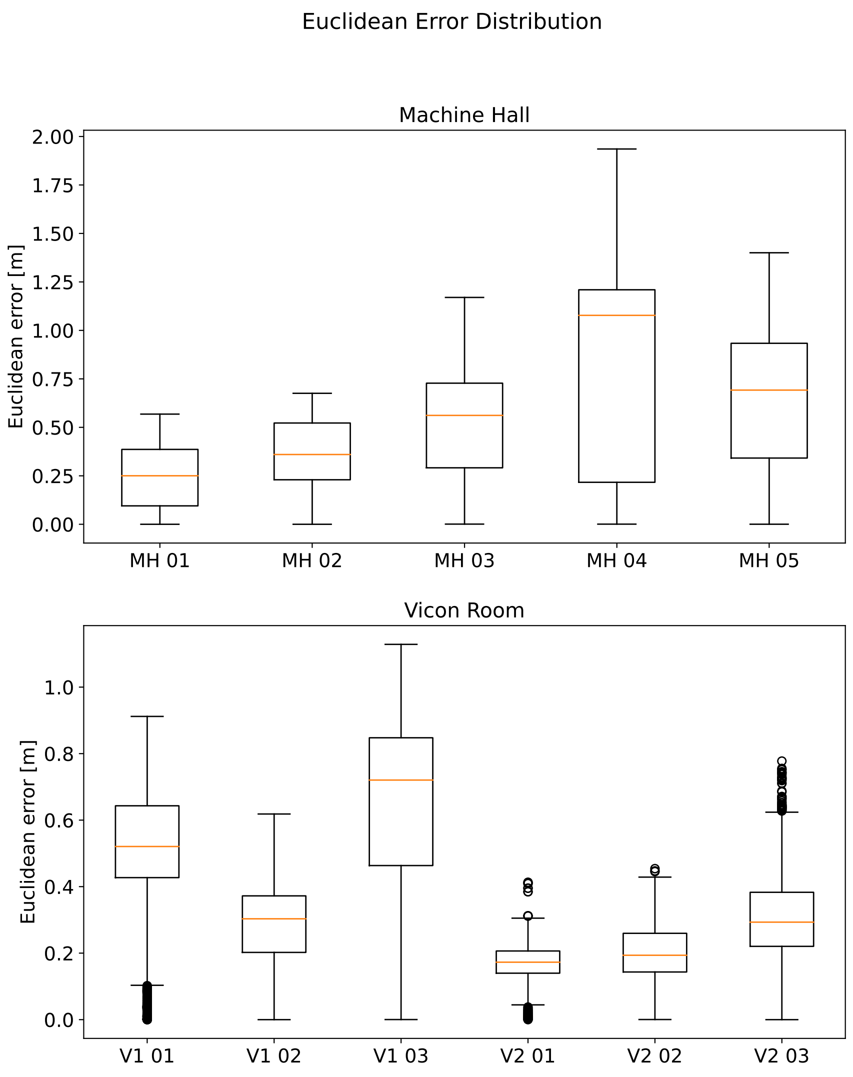
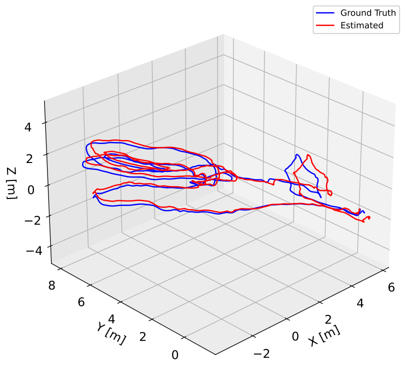
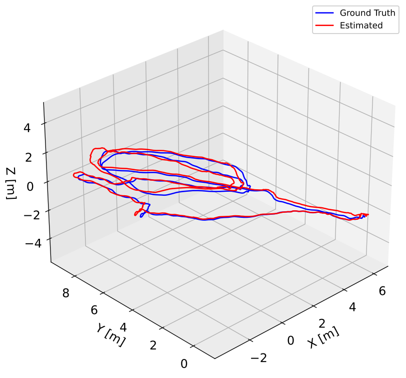


Figure 10.4: Box plot of Euclidean translation error for the proposed methods on the different dataset sections of EuRoC.

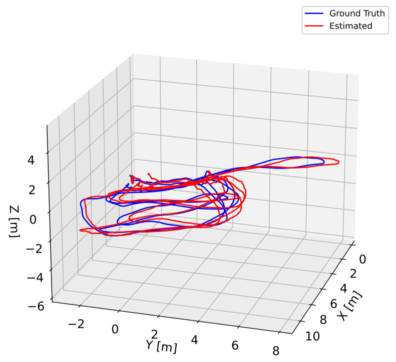
10.4 Example Trajectories



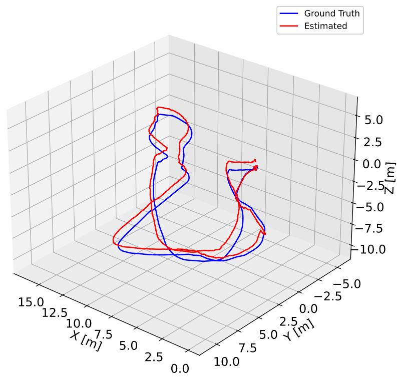
(a) Machine Hall 1



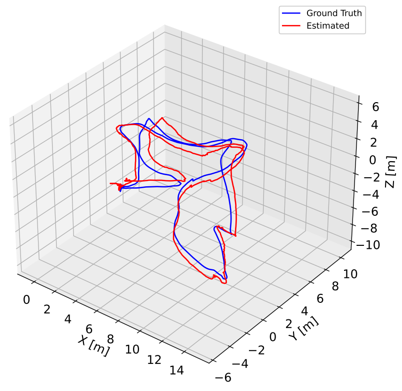
(b) Machine Hall 2



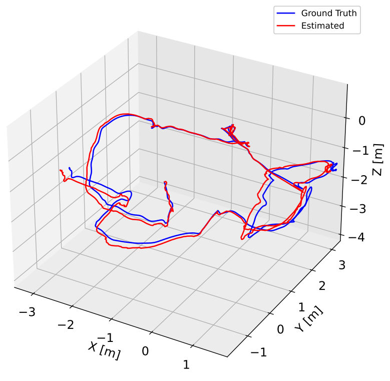
(a) Machine Hall 3



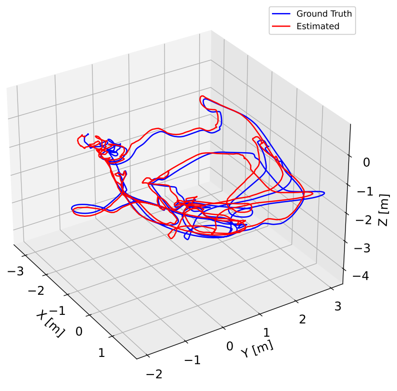
(b) Machine Hall 4



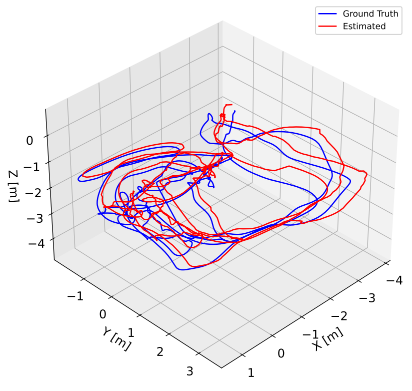
(a) Machine Hall 5



(b) Vicon Room 2, 1st section



(a) Vicon Room 2, 2nd section



(b) Vicon room 2, 3rd section

10.5 Memory Usage

Region	Used	Capacity
FLASH	2202.9 KB	16256 KB
ITCM	3.2 KB	32 KB
Total	2206.1 KB/2.2 MB	16288 KB/15.9 MB
DTCM	469.3 KB	480 KB
OCRAM	1178.2 KB	1280 KB
SDRAM*	1451.4 KB	46875 KB
Total	3098.9 KB/3 MB	48635 KB/47.5 MB

Table 10.7: Code and data usage for the implementation. Code resides in FLASH and ITCM, whereas data resides in DTCM, OCRAM and SDRAM. Here, 1 KB = 1024 B and 1 MB = 1024 KB. *The reduced capacity for the SDRAM (it has a capacity of 64 MB on the evaluation kit) comes from reserving parts of it for the data streaming functionality during the tests. This is thus not directly related to the memory required for the VIO pipeline. The code size of this functionality is included in the FLASH entry in the table, yielding a smaller program size in an implementation without this.

Comparison With State-Of-The-Art

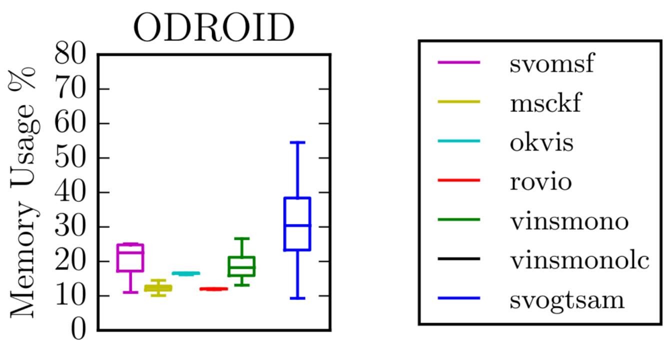


Figure 10.9: Memory usage results for state-of-the-art VIO methods running on an ODROID XU4 with 2 GB of memory (figure is from Delmerico and Scaramuzza's public author version of their benchmark comparison paper [79])

10.6 Power Consumption

	Running [mA]	**Idle [mA]**
M7+M4	380	310
M7	350	280

Table 10.8: Power consumption of the whole 3.3 V domain of the i.MX RT1170 evaluation kit — including the various components on the board that are not utilised — whilst running the pipeline and idling. The two rows signify the difference between when the M4 core is enabled (but not used for anything) and when it is disabled.

Chapter 11

Discussion

This chapter summarises and discusses the results obtained from evaluating the Lucas-Kanade implementation and the VIO pipeline as a whole.

11.1 Lucas-Kanade

From the results seen in figure 10.1 and table 10.1, the proposed implementation of pyramidal Lucas-Kanade is far less robust than OpenCV's implementation in general. Both have a comparable median error, but the proposed implementation suffers from larger quantiles.

The images within the KITTI dataset have a large baseline, which the proposed implementation handles poorly. OpenCV's implementation includes affine transformation for warping the patches used in the tracking, increasing the robustness for large baselines or heavy rotation. With that being said, a high camera frame rate can also alleviate this, effectively circumventing the problem to a certain degree.

11.2 Pipeline Runtime

The proposed VIO pipeline achieves comparable runtime even though the hardware it runs on is far weaker when compared with the ODROID XU4. With a maximum of 86 ms per frame, the pipeline can run at a standard camera frame rate of 10 Hz. Moreover, the results are from solely utilising the M7 core of the i.MX RT1170; the second core was not used in the implementation. Only two of the state-of-the-art methods tested on the ODROID XU4 have a CPU utilisation close to or lower than 100 % (in other words, they only use one core): SVO+MSF and ROVIO. When comparing the processing time for these two methods against the proposed solution, it shows that it is comparable to ROVIO, whereas SVO+MSF is faster. With that being said, the ODROID does operate at 30 % faster clock speed for the Cortex-A7 MPU and 90 % faster clock speed for the Cortex-A15 MPU.

The processing times from figure 10.3a are calculated from the incoming camera frame to when the state is updated. This is not the case in the proposed implementation, where the backend operates on half the frequency of the frontend (section 8.5). Thus, the runtime would increase if the backend was set to operate on the same frequency.

Furthermore, the maximum frame time largely depends on the number of feature tracks in the backend. As shown in table 10.3, the average number of features is well below the maximum allowed features set to 50, which is due to the relatively high FAST threshold and the outlier rejection with BRIEF. This positively influences the runtime but can make the whole pipeline less robust.

The proposed solution does not achieve realtime-ness on the EuRoC dataset, where the camera frame rate is 20 Hz. In a real-world use case, the camera frame rate thus has to be around 10 Hz, or frames have to be dropped. With that being said, in the test case with EuRoC, few frames would have had to be dropped, which can be observed in the last column of table 10.2 and in figure 10.2. This could, of course, negatively impact the accuracy of the pipeline.

11.3 Pipeline Accuracy & Filter Confidence

The proposed pipeline has comparable accuracy to the state-of-the-art methods tested on the ODROID XU4. Moreover, it does not lose track, such as OKVIS, SVO+MSF and SVO+GTSAM. With that being said, it is definitely in the lower level of accuracy, but comparable to the MSCKF implementation used by Delmerico and Scaramuzza [35] and SVO+MSF.

For a more accurate implementation, the required processing time of the pipeline would increase. That said, improving certain parts, such as making the Lucas-Kanade tracker more robust, could greatly improve the accuracy and not necessarily have that significant penalty on the needed processing time. More features and increased outlier rejection with inverse Lucas-Kanade and RANSAC could also positively impact the accuracy of the pipeline but will increase the required processing time.

In general, the pipeline struggles on the dataset sections with larger displacement, such as MH 04 and MH 05, which might indicate some scale inaccuracy due to little excitation of the system or inaccurate triangulation. It was observed that for sections where there occurred large displacements followed by back-tracing, the pipeline would be able to arrive at the same starting point but had an increasing error during the forward-displacement.

Moreover, MH 04 and MH 05 are dimly lit during certain parts of the trajectory. This could be alleviated by histogram equalisation before the image is processed in the frontend, but this would again increase the runtime. This is closely related to the reduced accuracy for V1 03 — where exposure changes happen — making it harder for the pipeline to track features across frames as it breaks with the Lucas-Kanade assumption of intensity consistency.

The pipeline generally has more features to work with for the vicon room 2 part of the dataset, seemingly increasing the accuracy compared with the vicon room 1 part. The two parts are somewhat equivalent, yielding that the pipeline is sensitive to a low amount of features (and possibly outliers which are not detected). It can thus be argued that the pipeline can tackle large and fast rotations as long as the number of features is relatively high.

11.3.1 Filter Confidence

From the ANEES observed in table 10.6, it can be seen that the filter, in general, is too confident in its estimates for the orientation and the velocity. For the velocity, one can observe that this happens particularly for MH 04, MH 05 and in the V1 part of the dataset. This will naturally propagate to the position and indicates that the pipeline is not as robust for quick and large displacements where, e.g. motion blur could also be a factor. This further emphasises that a more robust feature tracker is needed.

The filter becomes highly overconfident in its orientation during the V1 part of the dataset, which can be argued to be from the low feature count and having to rely more on the IMU.

The high variety of the ANEES indicates that the filter is sensitive for different environments, for which features are used and for various different movements. It needs more tuning, and possibly the pipeline should be appended with components for increased robustness.

11.4 Pipeline Memory Usage

From the memory usage observed in table 10.7 it can be argued that the pipeline is relatively lightweight. It should, however, be emphasised that the implementation does not fit within the DTCM and the OCRAM alone, yielding that external RAM is required should the proposed solution be used with a custom PCB. Moreover, the i.MX RT1170 does not have non-volatile program memory on the chip (ITCM is volatile), rendering some external FLASH necessary.

The comparison presented in figure 10.9 shows that the state-of-the-art method with the lowest upper bound for memory usage is ROVIO, at 10 % of the total capacity of the ODROID XU4. This yields $\approx$ 204.8 MB of memory usage (here 1 GB = 1024^3 B). Thus, the proposed implementation requires roughly 68x less memory. This estimate is highly uncertain because ROVIO is built around ROS (robot operating system). Profiling the memory usage of the implementation does not solely profile the VIO aspects but includes other parts of the system, which skews the result.

11.5 Pipeline Power Consumption

From the current consumption observed in table 10.8, it can be seen that a (rather high) upper bound is given by $\approx$ 1.115 W when the pipeline is running (and only using the M7 core). It should again be emphasised that this is for the whole 3.3 V domain of the evaluation kit. The power consumption will naturally be lower for a custom PCB with only the MCU, some external RAM and some external program memory. The difference in power consumption between when the system is running the pipeline and when it is idling (but not clocked differently) is likely to come from using the external Ethernet module on the evaluation kit; quite a significant amount of data is transferred every second.

An exact estimate for the required power on the ODROID XU4 is hard to retrieve. The user manual [80] states anywhere between 10 W and 20 W. Non-verified sources claim an *idle* power consumption from $1.7 - 2.9$ W. The data sheet for the chip itself (the Samsung Exynos 5 Octa) is under an NDA and not something the author could retrieve. The author had to depend on non-verified sources estimating the maximum power ranging from 3 W to 4 W. As shown in figure 10.3, the CPU utilisation is well below that of three cores for all methods except the particular implementation of MSCKF, yielding that the used power from the chip itself will be well below this maximum value. Furthermore, throttling and adaptive clock frequency adjustment will happen on a platform like the ODROID XU4. The chip itself can also swap between the Cortex-A7 and the Cortex-A15 depending on the load, where the Cortex-A7 is more efficient than the Cortex-A15. That said, when comparing the kits themselves and nominal values, it is highly likely that the i.MX RT1170 evaluation kit will require far less power than the ODROID XU4. For dedicated PCBs with the i.MX RT1170 and the Samsung Exynos 5 Octa one only have the maximum power ratings to work from, where the i.MX RT1170 — with a maximum of 437.025 mW at 25°C when *both* cores are enabled (see [81], table 13) — is more efficient. The exact comparison can, however, only be verified with actual tests.

When compared against the Navion ASIC chip [29], the i.MX RT1170 in itself will require far more power. The data sheet for the i.MX RT1170 specifies a maximum of 366.075 mW at 25°C when *only* the M7 core at 1 GHz is enabled (table 14 in [81]). The Navion chip uses 2 mW when processing camera frames at 20 Hz. This is expected, as an ASIC solution will naturally require far less power when designed particularly for the problem at hand.

11.6 Closing Remarks

The proposed VIO pipeline can achieve comparable accuracy at comparable processing times in light of state-of-the-art VIO methods running on the ODROID XU4. It is likely to draw far less power, but it is hard to draw exact conclusions without more tests. The pipeline is, however, sensitive to different environments and changes in illumination. It would potentially be improved by a more robust feature tracker, utilising more features and improving outlier rejection.

Chapter 12

Conclusion

This master's thesis — built on the preceding specialisation project — has proposed a MSCKF-based visual-inertial odometry pipeline running on a microcontroller. The proposed pipeline has been tested on the EuRoC dataset, achieving comparable accuracy and runtime to some of the state-of-the-art methods, with the requirement of 3 MB of RAM and at a potentially far lower power budget.

The implementation has explored algorithmic aspects to reduce the needed operations on data and utilises SIMD to operate on data faster. The major improvements in runtime in the implementation stem from how the proposed VIO pipeline interacts with memory and where the data the implementation utilises resides in memory. The VIO pipeline greatly prioritises faster memory regions of the microcontroller to achieve comparable runtime performance to state-of-the-art methods running on comparable but much more performant hardware.

Furthermore, the implementation has explored how popularly used components in state-of-the-art VIO methods can use less memory, with the use of patch pyramids for Lucas-Kanade tracking from the specialisation project and the use of in-place reflection for BRIEF. The implementation has also introduced a custom memory allocator spanning over two memory regions, which can be used to prioritise data for a faster memory region whilst having redundancy in a slower memory region.

Further Work

Further work of the proposed VIO implementation resides mainly in improving the robustness and accuracy, where a more robust pyramidal Lucas-Kanade tracker could potentially greatly improve the accuracy of the pipeline. Furthermore, further work could also explore how the pipeline could become more robust against exposure changes, dimly lit scenes and motion blur.

This master's thesis has not investigated incorporating guarantees for real-time requirements, such as never lagging behind the camera frame rate. This is also an aspect which could be explored in further work.

Whilst not mentioned, the author also explored utilising the GPU on the i.MX

RT1170 for accelerated image processing. However, due to the tight coupling between the provided GPU and display drivers, a solution for working solely with a frame buffer in memory was not found. This was thus deemed as an area for future work.

Bibliography

[1] S. Gangstad, ‘Bare-Metal Feature Extraction & Tracking,’ 2022.

[2] X. Qiu, H. Zhang, W. Fu, C. Zhao and Y. Jin, ‘Monocular visual-inertial odometry with an unbiased linear system model and robust feature tracking front-end,’ *Sensors (Switzerland)*, vol. 19, 8 Apr. 2019, issn: 14248220. doi: 10.3390/s19081941.

[3] X. Qiu, H. Zhang and W. Fu, ‘Lightweight hybrid visual-inertial odometry with closed-form zero velocity update,’ *Chinese Journal of Aeronautics*, vol. 33, pp. 3344–3359, 12 Dec. 2020, issn: 10009361. doi: 10.1016/j.cja.2020.03.008.

[4] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang and G. Huang, ‘OpenVINS: A Research Platform for Visual-Inertial Estimation,’ in *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020. [Online]. Available: https://github.com/rpng/open_vins.

[5] G. Guennebaud, B. Jacob *et al.*, *Eigen v3*, 2010. [Online]. Available: http://eigen.tuxfamily.org.

[6] J. Wellbelove, *Embedded Template Library*. [Online]. Available: https://www.etlcpp.com.

[7] M. Menze, C. Heipke and A. Geiger, ‘Object Scene Flow,’ *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[8] M. Menze, C. Heipke and A. Geiger, ‘Joint 3D Estimation of Vehicles and Scene Flow,’ in *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[9] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik and R. Siegwart, ‘The EuRoC micro aerial vehicle datasets,’ *The International Journal of Robotics Research*, 2016. doi: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[10] H. C. Longuet-Higgins, 'A computer algorithm for reconstructing a scene from two projections,' *Nature*, vol. 293, pp. 133–135, 1981.

[11] C. G. Harris and J. M. Pike, '3D positional integration from image sequences,' *Image and Vision Computing*, vol. 6, pp. 87–90, 2 1988.

[12] H. P. Moravec, 'Obstacle avoidance and navigation in the real world by a seeing robot rover,' Jan. 1980. doi: `10.1184/R1/6557033.v1`. [Online]. Available: `https://frc.ri.cmu.edu/~hpm/project.archive/robot.papers/1975.cart/1980.html.thesis/index.html`.

[13] A. I. Mourikis and S. I. Roumeliotis, 'A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation,' in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572. doi: `10.1109/ROBOT.2007.364024`.

[14] M. Bloesch, S. Omari, M. Hutter and R. Siegwart, 'Robust visual inertial odometry using a direct EKF-based approach,' *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 298–304, 2015, issn: 21530866. doi: `10.1109/IROS.2015.7353389`.

[15] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart and P. Furgale, 'Keyframe-based visual-inertial odometry using nonlinear optimization,' *International Journal of Robotics Research*, vol. 34, pp. 314–334, 3 2015, issn: 17413176. doi: `10.1177/0278364914554813`.

[16] C. Forster, M. Pizzoli and D. Scaramuzza, 'SVO: Fast semi-direct monocular visual odometry,' *IEEE Robotics and Automation Magazine*, p. 22, 2014. doi: `10.1109/ICRA.2014.6906584`.

[17] J. Engel, T. Schöps and D. Cremers, 'LSD-SLAM: Large-scale direct monocular SLAM,' in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13*, Springer, 2014, pp. 834–849.

[18] R. Mur-Artal, J. M. M. Montiel and J. D. Tardos, 'ORB-SLAM: a versatile and accurate monocular SLAM system,' *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[19] R. Mur-Artal and J. D. Tardos, 'ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,' *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, 5 Oct. 2017, issn: 15523098. doi: `10.1109/TRO.2017.2705103`.

[20] T. Qin, P. Li and S. Shen, 'VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator,' *IEEE Transactions on Robotics*, vol. 34, pp. 1004–1020, 4 2018, issn: 15523098. doi: `10.1109/TRO.2018.2853729`.

[21] C. Campos, R. Elvira, J. J. Rodriguez, J. M. Montiel and J. D. Tardos, 'ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM,' *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, 6 2021, issn: 19410468. doi: 10.1109/TRO.2021.3075644.

[22] R. Watt, *Black Hornet Nano Helicopter UAV*, https://photos.defenceimagery.mod.uk/fotoweb/cache/v2/-/S/Archive/Archive/Army/45155/45155077.jpg.iCfi5rDUMWATAA.E-WFVvpqE-.jpg, Open Government License. Accessed: 2023-06-02.

[23] Z. Zhang, A. Suleiman, L. Carlone, V. Sze and S. Karaman, 'Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach,' in *Robotics: Science and Systems*, 2017.

[24] Y. He, Y. Wang, C. Liu and L. Zhang, 'PicoVO: A Lightweight RGB-D Visual Odometry Targeting Resource-Constrained IoT Devices,' vol. 2021-May, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 5567–5573, isbn: 9781728190778. doi: 10.1109/ICRA48506.2021.9561285.

[25] G. Decroon, M. Percin, B. Remes, R. Ruijsink and C. De Wagter, *The delfly: Design, aerodynamics, and artificial intelligence of a flapping wing robot*. Jan. 2015, pp. 1–218. doi: 10.1007/978-94-017-9208-0.

[26] H. Müller, D. Palossi, S. Mach, F. Conti and L. Benini, 'Fünfiiber-Drone: A Modular Open-Platform 18-grams Autonomous Nano-Drone,' in *2021 Design, Automation & Test in Europe Conference & Exhibition*, 2021, pp. 1610–1615. doi: 10.23919/DATE51398.2021.9474262.

[27] R. G. Valenti, I. Dryanovski, C. Jaramillo, D. P. Ström and J. Xiao, 'Autonomous quadrotor flight using onboard RGB-D visual odometry,' in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 5233–5238.

[28] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen and M. Pollefeys, 'Vision-based autonomous mapping and exploration using a quadrotor MAV,' in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4557–4564.

[29] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman and V. Sze, 'Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones,' *IEEE Journal of Solid-State Circuits*, vol. 54, pp. 1106–1119, 4 Apr. 2019, issn: 00189200. doi: 10.1109/JSSC.2018.2886342.

[30] D. K. Mandal, S. Jandhyala, O. J. Omer, G. S. Kalsi, B. George, G. Neela, S. K. Rethinagiri, S. Subramoney, L. Hacking, J. Radford, E. Jones, B. Kuttanna and H. Wang, 'Visual Inertial Odometry At the Edge: A Hardware-Software Co-design Approach for Ultra-low Latency and Power,' *2019 Design, Automation and Test in Europe Conference and Exhibition*, pp. 960–963, 2019, issn: 1558-1101. doi: 10.23919/2019.8714921.

[31] R. Liu, J. Yang, Y. Chen and W. Zhao, 'ESLAM: An energy-efficient accelerator for real-time ORB-SLAM on FPGA platform,' in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[32] Z. Xu, J. Yu, C. Yu, H. Shen, Y. Wang and H. Yang, 'CNN-based Feature-point Extraction for Real-time Visual SLAM on Embedded FPGA,' in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2020, pp. 33–37.

[33] Z. Wan, B. Yu, T. Y. Li, J. Tang, Y. Zhu, Y. Wang, A. Raychowdhury and S. Liu, 'A Survey of FPGA-Based Robotic Computing,' *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 48–74, 2021. doi: 10.1109/MCAS.2021.3071609.

[34] Teledyne FLIR LLC, *Black Hornet PRS datasheet*, https://flir.netx.net/file/asset/14121/original/attachment, Accessed: 2023-06-14.

[35] J. Delmerico and D. Scaramuzza, 'A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots,' Institute of Electrical and Electronics Engineers Inc., Sep. 2018, pp. 2502–2509, isbn: 9781538630815. doi: 10.1109/ICRA.2018.8460664.

[36] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli and R. Siegwart, 'A robust and modular multi-sensor fusion approach applied to MAV navigation,' in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 3923–3929. doi: 10.1109/IROS.2013.6696917.

[37] C. Forster, L. Carlone, F. Dellaert and D. Scaramuzza, 'On-Manifold Preintegration for Real-Time Visual-Inertial Odometry,' *IEEE Transactions on Robotics*, vol. 33, pp. 1–21, 1 Feb. 2017, issn: 15523098. doi: 10.1109/TRO.2016.2597321.

[38] F. Dellaert, 'Factor Graphs and GTSAM: A Hands-on Introduction,' 2012.

[39] D. Scaramuzza and F. Fraundorfer, 'Tutorial: Visual odometry,' *IEEE Robotics and Automation Magazine*, vol. 18, pp. 80–92, 4 2011, issn: 10709932. doi: 10.1109/MRA.2011.943233.

[40] E. Rosten and T. 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. [Online]. Available: http://www.edwardrosten.com/work/rosten_2005_tracking.pdf.

[41] E. Rosten and T. 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. [Online]. Available: http://www.edwardrosten.com/work/rosten_2006_machine.pdf.

[42] B. D. Lucas and T. Kanade, ‘An Iterative Image Registration Technique with an Application to Stereo Vision,’ in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[43] D. Lowe, ‘Object recognition from local scale-invariant features,’ in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. doi: 10.1109/ICCV.1999.790410.

[44] S. Gangstad, *TinyVIO*, https://github.com/simengangstad/TinyVIO.

[45] J. Solà, ‘Quaternion kinematics for the error-state Kalman filter,’ Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.02508.

[46] R. Szeliski, *Computer Vision: Algorithms and Applications*, First. 2010. doi: 10.1007/978-1-84882-935-0.

[47] L. E. Osterman, N. D. Opdyke, P. N. Webb, T. E. Ronan, L. J. H. Jr and T. E. Delaca, ‘A computer algorithm for reconstructing a scene from two projections,’ 1981, pp. 206–212.

[48] R. I. Hartley, ‘In Defense of the Eight-Point Algorithm,’ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, 6 1997.

[49] K. Levenberg, ‘A method for the solution of certain non-linear problems in least squares,’ *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[50] J. Nocedal and S. J. Wright, *Numerical Optimization*, Second. Springer, 2006, isbn: 978-0387-30303-1.

[51] P. Sturm, S. Ramalingam, J.-P. Tardif, S. Gasparini and J. Barreto, ‘Camera Models and Fundamental Concepts Used in Geometric Computer Vision,’ *Foundations and Trends® in Computer Graphics and Vision*, vol. 6, no. 1–2, pp. 1–183, 2011, issn: 1572-2740. doi: 10.1561/0600000023. [Online]. Available: http://dx.doi.org/10.1561/0600000023.

[52] C. S. Fraser, ‘Digital camera self-calibration,’ *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 52, no. 4, pp. 149–159, 1997, issn: 0924-2716. doi: https://doi.org/10.1016/S0924-2716(97)00005-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271697000051.

[53] WolfWings, *Barrel distortion*, https://commons.wikimedia.org/wiki/File:Barrel_distortion.svg, Public domain, via Wikimedia Commons. Accessed: 2023-06-07.

[54] WolfWings, *Pincushion distortion*, https://commons.wikimedia.org/wiki/File:Pincushion_distortion.svg, Public domain, via Wikimedia Commons. Accessed: 2023-06-07.

[55] J. Wang, F. Shi, J. Zhang and Y. Liu, ‘A new calibration model of camera lens distortion,’ *Pattern Recognition*, vol. 41, pp. 607–615, 2 Feb. 2008, issn: 00313203. doi: 10.1016/j.patcog.2007.06.012.

[56] C. B. Duane, 'Close-range camera calibration,' *Photogramm. Eng*, vol. 37, no. 8, pp. 855–866, 1971.

[57] P. Drap and J. Lefèvre, 'An Exact Formula for Calculating Inverse Radial Lens Distortions,' *Sensors*, vol. 16, 6 Jun. 2016, issn: 14248220. doi: `10.3390/s16060807`.

[58] H. Bay, T. Tuytelaars and L. Van Gool, 'SURF: Speeded Up Robust Features,' in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417, isbn: 978-3-540-33833-8.

[59] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, 'ORB: An efficient alternative to SIFT or SURF,' in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. doi: `10.1109/ICCV.2011.6126544`.

[60] M. Calonder, V. Lepetit, C. Strecha and P. Fua, 'BRIEF: Binary Robust Independent Elementary Features,' in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos and N. Paragios, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792, isbn: 978-3-642-15561-1.

[61] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, 'ORB: an efficient alternative to SIFT or SURF,' Nov. 2011, pp. 2564–2571. doi: `10.1109/ICCV.2011.6126544`.

[62] E. Rosten, *FAST pattern*, `https://www.edwardrosten.com/work/fast.html`, Accessed: 2023-06-02.

[63] OpenCV, *Optical Flow*, `https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html`, Accessed: 2023-06-07.

[64] C. Harris, M. Stephens *et al.*, 'A combined corner and edge detector,' in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 147–151.

[65] J. Shi and C. Tomasi, 'Good features to track,' in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. doi: `10.1109/CVPR.1994.323794`.

[66] Cmglee, *Image pyramid*, `https://commons.wikimedia.org/wiki/File:Image_pyramid.svg`, CC BY-SA 3.0, via Wikimedia Commons. Accessed: 2023-06-07.

[67] P. C. Mahalanobis, 'On the generalized distance in statistics,' *Sankhy A: The Indian Journal of Statistics, Series A (2008-)*, vol. 80, S1–S7, 2018.

[68] R. E. Kálmán, 'A New Approach to Linear Filtering and Prediction Problems,' 1960. [Online]. Available: `http://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf`.

[69] S. Särkkä and A. Solin, *Applied Stochastic Differential Equations* (Institute of Mathematical Statistics Textbooks). Cambridge University Press, 2019, isbn: 9781108693448. [Online]. Available: `https://doi.org/10.1017/9781108186735`.

[70] H. Bourlès and B. Marinescu, *Linear Time-Varying Systems - Algebraic-Analytic Approach*. Apr. 2011, isbn: 978-3-642-19726-0. doi: `10.13140/2.1.2149.5042`.

[71] B. Øksendal, *Stochastic Differential Equations, An Introduction with Applications*, Sixth. Springer Berlin, Heidelberg, 2014, isbn: 978-3-642-14394-6. [Online]. Available: `https://doi.org/10.1007/978-3-642-14394-6`.

[72] M. Li and A. I. Mourikis, 'High-precision, consistent EKF-based visual-inertial odometry,' *The International Journal of Robotics Research*, vol. 32, pp. 690–711, 6 2013, issn: 0278-3649. doi: `10.1177/0278364913481251`.

[73] D. Lay, *Linear Algebra and Its Applications*, Fifth. Pearson Education Limited 2016, 2016, isbn: 978-1-292-09223-2.

[74] M. Li and A. I. Mourikis, 'Improving the accuracy of EKF-based visual-inertial odometry,' in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 828–835. doi: `10.1109/ICRA.2012.6225229`.

[75] Microchip Technology Inc., *AVR128DB48*, `https://www.microchip.com/en-us/product/avr128db48`, Accessed: 2023-06-09.

[76] Amazon Web Services Inc., *FreeRTOS*. [Online]. Available: `https://github.com/FreeRTOS/FreeRTOS-Kernel`.

[77] STMicroelectronics N.V., *RM0399: STM32H745/755 and STM32H747/757 advanced Arm-based 32-bit MCUs*, `https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html`, Accessed: 2023-06-09.

[78] NXP Semiconductors N.V., *i.MX RT1170 Evaluation Kit*, `https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-rt1170-evaluation-kit:MIMXRT1170-EVK`, Accessed: 2023-06-09.

[79] J. Delmerico and D. Scaramuzza, 'A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots, Author Version,' [Online]. Available: `https://rpg.ifi.uzh.ch/docs/ICRA18_Delmerico.pdf`.

[80] Hardkernel, *ODROID-XU4 User Manual*, `https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf`, Accessed: 2023-06-06.

[81] NXP Semiconductors, *i.MX RT1170 Crossover Processors Data Sheet for Consumer Products*, `https://www.nxp.com/docs/en/data-sheet/IMXRT1170CEC.pdf`, Accessed: 2023-06-06.

Appendices

Appendix A

Jacobians of Rotations

In the following derivations, the global angular error is used, which coincides with the theory in chapter 6.

Let the rotation of a vector $\mathbf{v}^b$ in frame b to frame a be given by $\mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^b$. The rotation axis is given by $\boldsymbol{\theta} = \phi\mathbf{u}$, where ϕ is the angle and $\mathbf{u}$ is the axis of rotation. Moreover, when $\boldsymbol{\theta} \rightarrow \mathbf{0} \Rightarrow \sin(\phi) \approx \phi, 1-\cos(\phi) \approx 0$. The jacobian with respect to $\mathbf{q}_{ab}$ is then given by:

$$
\begin{aligned}
\frac{\partial \mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a}{\partial \mathbf{q}_{ab}} &= \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} \frac{(\boldsymbol{\theta} \oplus \mathbf{R}(\mathbf{q}_{ab}))\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} \frac{(\mathrm{Exp}(\boldsymbol{\theta})\mathbf{R}(\mathbf{q}_{ab}))\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} \frac{((\mathbf{I}_{3\times3} + \sin(\phi)[\mathbf{u}]_\times + (1-\cos(\phi))[\mathbf{u}]_\times^2)\mathbf{R}(\mathbf{q}_{ab}))\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} \frac{((\cancel{\mathbf{I}_{3\times3}} + \phi[\mathbf{u}]_\times)\mathbf{R}(\mathbf{q}_{ab}))\mathbf{v}^a - \cancel{\mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a}}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} \frac{[\boldsymbol{\theta}]_\times\mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\rightarrow\mathbf{0}} -\frac{\boldsymbol{\theta}[\mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a]_\times}{\boldsymbol{\theta}} \\
&\approx -[\mathbf{R}(\mathbf{q}_{ab})\mathbf{v}^a]_\times
\end{aligned} \tag{A.1}
$$

Let the rotation of a vector $\mathbf{v}^a$ in frame a to frame b be given by $\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a$. The jacobian with respect to $\mathbf{q}_{ab}$ is then given by:

$$\begin{aligned}
\frac{\partial \mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\partial \mathbf{q}_{ab}} &= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{(\boldsymbol{\theta}\oplus\mathbf{R}(\mathbf{q}_{ab}))^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{(\mathrm{Exp}(\boldsymbol{\theta})\mathbf{R}(\mathbf{q}_{ab}))^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{((\mathbf{I}_{3\times3} + \sin(\phi)[\mathbf{u}]_\times + (1-\cos(\phi))[\mathbf{u}]_\times^2)\mathbf{R}(\mathbf{q}_{ab}))^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{((\cancel{\mathbf{I}_{3\times3}} + \phi[\mathbf{u}]_\times)\mathbf{R}(\mathbf{q}_{ab}))^T\mathbf{v}^a - \cancel{\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{([\boldsymbol{\theta}]_\times\mathbf{R}(\mathbf{q}_{ab}))^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{\mathbf{R}(\mathbf{q}_{ab})^T[\boldsymbol{\theta}]_\times^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} -\frac{\mathbf{R}(\mathbf{q}_{ab})^T[\boldsymbol{\theta}]_\times\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{\mathbf{R}(\mathbf{q}_{ab})^T\boldsymbol{\theta}[\mathbf{v}^a]_\times}{\boldsymbol{\theta}} \\
&\approx \mathbf{R}(\mathbf{q}_{ab})^T[\mathbf{v}^a]_\times
\end{aligned} \tag{A.2}$$

Let the rotation of a vector $\mathbf{v}^a$ in frame a to frame c be given by $\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a$. The jacobian with respect to $\mathbf{q}_{cb}$ is then given by:

$$\begin{aligned}
\frac{\partial \mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\partial \mathbf{q}_{cb}} &= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{(\boldsymbol{\theta}\oplus\mathbf{R}(\mathbf{q}_{cb}))\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{\mathrm{Exp}(\boldsymbol{\theta})\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&= \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{(\mathbf{I}_{3\times3} + \sin(\phi)[\mathbf{u}]_\times + (1-\cos(\phi))[\mathbf{u}]_\times^2)\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a - \mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{(\cancel{\mathbf{I}_{3\times3}} + [\boldsymbol{\theta}]_\times)\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a - \cancel{\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} \frac{[\boldsymbol{\theta}]_\times\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a}{\boldsymbol{\theta}} \\
&\approx \lim_{\boldsymbol{\theta}\to\mathbf{0}} -\frac{\boldsymbol{\theta}[\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a]_\times}{\boldsymbol{\theta}} \\
&\approx -[\mathbf{R}(\mathbf{q}_{cb})\mathbf{R}(\mathbf{q}_{ab})^T\mathbf{v}^a]_\times
\end{aligned} \tag{A.3}$$

Appendix B

Runge-Kutta

Runge-Kutta is a numerical integration method to estimate the function $y(t_k)$, given the known derivative $\dot{y}(t_k) = f(t_k, y(t_k))$ and initial condition $y_0 = y(t_0)$. In this section, the fourth-order Runga-Kutta method will be outlined.

Let the step size between two consecutive frames k and $k+1$ be given by Δt, then:

$$y(t_{k+1}) = y(t_k) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{B.1}$$

where:

$$\begin{align} k_1 &= f(t_k, y_k) \tag{B.2} \\ k_2 &= f(t_k + \frac{\Delta t}{2}, y_k + \Delta t \frac{k_1}{2}) \tag{B.3} \\ k_3 &= f(t_k + \frac{\Delta t}{2}, y_k + \Delta t \frac{k_2}{2}) \tag{B.4} \\ k_4 &= f(t_k + \Delta t, y_k + \Delta t k_3) \tag{B.5} \\ & \tag{B.6} \end{align}$$