TÉCNICO LISBOA

Visual Inertial Odometry with Event Cameras

José Pedro Ribeiro Gomes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors

Professor José António da Cruz Pinto Gaspar
Professor Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Professor João Fernando Cardoso Silva Sequeira
Supervisor: Professor José António da Cruz Pinto Gaspar
Member of the Committee: Professor Pedro Daniel dos Santos Miraldo

September 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

It is with great pleasure and a sense of accomplishment that I write this last part of my thesis. It has been a long and difficult journey, that would not have been possible without a large group of people that supported me throughout, and I would like to acknowledge their help along the way.

First, I want to thank my supervisors, Prof. José Gaspar and Prof. Alexandre Bernardino, for guiding me through this journey (during these particularly unusual circumstances), for their support and insight, and for tolerating my idiosyncrasies and side projects throughout this journey. Also, a word of appreciation for the members of the ORIENT project and fellow students working on their respective thesis, in particular Prof. John Opstal, Gonçalo, Duarte and Bernardo.

I also wish to thank my friends and colleagues, in particular Gil Serrano, Teresa Alves, Renato Dias, Afonso Luís, Pedro Martins, Gonçalo Pereira, André Ribeiro, and Francisco Lopes, for their friendship and support throughout multiple periods of my life, and for making this journey far more fun and enjoyable (and keeping my mental health during the periods of lockdown).

I want to thank my school and university teachers, which, by showing me their passion for their respective subjects, have allowed me to discover my areas of interest and have provided the knowledge to keep studying them. In particular, I want to thank Prof. Luísa Falcão for her constant motivation and genuine interest in the students, which far surpassed the 90-minute classes, and Prof. Rui Marreiros, who has allowed me to start tinkering with robotics and participate in competitions early on.

Through my internships, I have come into contact with many interesting people who have taught and influenced me greatly, of which I want to highlight the group at Ericsson Stockholm, in particular José, Ananya, Amir, Diego, and Ioannis, for their support, allowing me to delve into a myriad of topics (many of which eventually ended up in this work, either directly or indirectly, without realizing at the time), as well as multiple discussions regarding areas of interest and possible future academic and professional paths.

Last, but most certainly not least, I want to express my profound gratitude towards my family, in particular my parents, Ana and Luís, and my brother, David, whose constant support, motivation and advice has allowed me to open many doors and make the most of many different situations. Words fall short to explain my gratitude.

To all, a sincere and heartfelt *Thank You!*

Resumo

Câmaras de eventos são sensores de imagem inspirados pelo funcionamento da retina dos animais, reportando mudanças de brilho ao nível de cada pixel, denominados “eventos”, que são reportados de forma assíncrona, com um ritmo de amostragem variável e dependente da cena, com uma resolução temporal muito elevada. Com efeito, estas câmaras permitem contrastes claro/escuro elevados numa dada cena e não sofrem de *motion blur* (distorção causada por movimento). Trabalhos recentes mostram a viabilidade do uso deste tipo de câmaras no contexto de odometria visual-inercial, podendo mesmo ultrapassar o desempenho de câmaras convencionais, em particular em cenas com movimento elevado. Neste trabalho, propomos e analisamos o desempenho de câmaras de eventos no contexto da estimação de pose, tirando proveito de fotogramas, eventos, e uma Unidade de Medida Inercial (UMI/IMU). Mostramos que o nosso método proposto, usando um Unscented Kalman Filter com representação de estado baseado em grupos de Lie é comparável, e pode mesmo superar um sistema convencional equivalente. Tanto quanto é do nosso conhecimento, este é o primeiro trabalho propondo a fusão de informação visual e inercial usando câmaras de eventos usando um Unscented Kalman Filter, bem como o seguidor de Kanade-Lucas-Tomasi baseado em Eventos (EKLT) no contexto de estimação de pose. Para além da proposta inicial apresentada, propomos uma segunda abordagem, que melhora a primeira, onde a pose estimada atual é realimentada no seguidor e detetor de caracteríticas (*features*), desta forma ajudando o estimador. Esta ideia é inovadora no contexto de câmaras de eventos. As propostas feitas são validadas com simulações, bem como *datasets* disponíveis publicamente e disponibilizados pelo Robotics and Perception Group de ETH Zurich, que se têm vindo a tornar um padrão para comparação de métodos no contexto de câmaras de eventos, e gravações recorrendo ao braço robótico Kinova para trajetórias precisas.

Palavras chave: Câmaras de eventos, Câmaras Neuromórficas, Sensores de Visão Dinâmicos (DVS), Estimação de Pose, Localização e Mapeamento Simultâneo (SLAM), Odometria Visual e Inercial (VIO), Unscented Kalman Filter (UKF), Grupos de Lie.

Abstract

Event cameras are image sensors inspired by animal's retina, reporting pixel-wise changes in brightness independently, called "events", which are streamed asynchronously, and at a varied refresh rate, with very high temporal resolution. Furthermore, these cameras allow for high contrasts within a scene (high dynamic range) and do not suffer motion blur. Recent works have shown the viability of these cameras in providing visual-inertial odometry information, that may even outperform conventional cameras in high-movement scenes. In this work, we propose and analyse the performance of event cameras in the context of pose estimation, by leveraging frames, events and Inertial Measurement Unit (IMU) information. We show that our proposed method using an Unscented Kalman Filter with Lie group embedding for state representation is comparable to, and can even outperform, an equivalent conventional approach to pose estimation. To the best of our knowledge, this is the first work proposing the fusion of visual with inertial information using events cameras by means of an Unscented Kalman Filter, as well as the use of the Event-based Kanade-Lucas-Tomasi tracker (EKLT) in the context of pose estimation. Furthermore, we propose a second approach, as an improvement to the previous method, where the current estimate of the state is fed back into the feature extractor in order to better track the features, which in turn helps with the pose estimation. Though the idea is not novel, its use in the context of pose estimation with event cameras, to the best of our knowledge, is. The proposed approaches are validated using a set of computer generated simulations generated for this work, as well as public event camera datasets available online from the Robotics and Perception Group from ETH Zurich, which have gradually become a benchmark for comparison in the context of event cameras, and a dataset created using a Kinova robot arm for precise trajectories that we created for this work.

Keywords: Event cameras, Neuromorphic camera, Dynamic Vision Sensor (DVS), Pose estimation, Simultaneous Localization and Mapping (SLAM), Visual Inertial Odometry (VIO), Unscented Kalman Filter (UKF), Lie groups.

Contents

List of Figures, Tables, Acronyms, Glossary and Nomenclature

List of Figures

List of Tables

Acronyms

DOF	Degrees of Freedom.
DoG	Difference of Gaussian.
EKF	Extended Kalman Filter.
EKLT	Event-Based Kanade–Lucas–Tomasi (EKLT) feature tracker.
fps	Frames per Second.
HDR	High Dynamic Range.
ICP	Iterative Closest Points.
IMU	Inertial Measurement Unit.
KF	Kalman Filter.
KLT	Kanade–Lucas–Tomasi (KLT) feature tracker.
RMSE	Root Mean Square Error.
SAE	Surface of Active Events.
SIFT	Scale Invariant Feature Transform.
SLAM	Simultaneous Localization and Mapping.
SSD	Sum of Squared Distances.
SURF	Speed-Up Robust Features.
UKF	Unscented Kalman Filter.

VIO	Visual-Inertial Odometry.

Glossary

Ego-motion estimation	The problem of trying to understand the movement of a system, in particular a camera, by how the features being captured by said camera move over time. It is particularly relevant for the concept of odometry.
Ego-motion	The movement of a system, in particular a camera, which produces changes to the way the world is captured. It is particularly relevant for the concept of odometry.
Event-based Kanade–Lucas–Tomasi (EKLT) feature tracker	Event-based approach to feature extraction and tracking, based on the KLT. The same method of patch comparison is used, but their origin is different. In particular, mixing frames with events.
Event cameras	Also called neuromorphic cameras, and dynamic vision sensors, are a type of camera that reports changes in brightness in a pixel level, rather than outputting a stream of frames at a constant frame rate, with the whole scene captured, as conventional cameras do.
Harris corner detector	Corner detector operator that extracts corner features from an image.
Inertial Measurement Unit (IMU)	Sensor that measures and reports a body's specific force (by means of accelerometers), angular rate (by means of gyroscopes), and sometimes the orientation of the body (by means of magnetometers). As such, reports information on the movement of the body being measured.
Kanade–Lucas–Tomasi (KLT) feature tracker	KLT is an approach to feature extraction and tracking, by means of comparing patches of images (around a feature of interest) between two consecutive frames. It can also be used to estimate the optical flow.

Motion field	Corresponds to the projection of 3D relative velocity vectors onto the 2D image plane. It is not necessarily measurable.
Odometry	Odometry refers to the use of data from motion sensors to estimate the change in position of a system over time.
Optical flow	Corresponds to the observed 2D displacements of brightness patterns in the image (that can result from the motion field (movement in the image), as well as moving light sources, for example).
Pseudo-frame	Image frames provide features at a constant rate. Event cameras do not. In this work, we call pseudo-frames to an accumulation of features from events over time, to create an analogous to conventional features to frames. One can think of it as a batch of features that are given the same timestamp.
Simultaneous Localization and Mapping (SLAM)	Simultaneous Localization and Mapping is a problem in mobile autonomous systems that tries to construct or update a map of the surrounds of the system, while simultaneously placing this system in said map.
Surface of Active Events (SAE)	Surface of Active Events is a visual method of representing events such that each entry in a frame of dimensions (W,H) contains the timestamp of the most recent acquired event at that location. This frame is then discretised and each pixel given a color based on how old the latest event received is.
Visual Inertial Odometry (VIO)	Visual Inertial Odometry (VIO) refers to the use of visual (cameras) and inertial (accelerometers, gyroscopes, IMU) sensors, in particular, to estimate the change in position of a system over time.

Nomenclature

δt	Time interval between two instants
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^N$	Real vector space of dimension N

$\mathbf{I}_N$	Identity matrix of size $N \times N$
$\mathcal{G}$	General Lie Group
$\mathcal{T}_x\mathcal{G}$	Tangent space of Lie Group $\mathcal{G}$ at point $x \in \mathcal{G}$
$\mathfrak{se}(3)$	Lie Algebra associated with $SE(3)$
$\mathfrak{se}_{2+p}(3)$	Lie Algebra associated with $SE(3)$
$\mathfrak{so}(3)$	Lie Algebra associated with $SO(3)$
$SE(3)$	Special Euclidean Group of dimension 3
$SE_{2+p}(3)$	Special Euclidean Group used in this work
$SO(3)$	Special Orthogonal Group of dimension 3

Chapter 1

Introduction

"Begin at the beginning," the King said gravely, "and go on till you come to the end: then stop."

— Lewis Carroll, Alice in Wonderland

Vision plays a very important role in the animal kingdom, and virtually every higher order animal has developed some sort of visual system to improve their chance of survival. Eyes at the retinal layer possess cells that are sensitive to light, namely photoreceptor cells that perceive colours - cones - and brightness - rods, which give the organism the sense of sight (see Fig. 1.1).

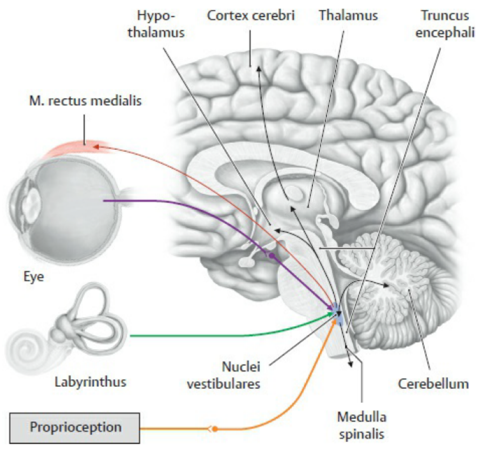


Figure 1.1: Coupling between the vestibular and ocular systems. Integration of information in the vestibular nucleus and corresponding output to adjacent cortical centres. From [Schuenke et al., 2020].

As such, it is only natural that sensors that can equip artificial systems with the sense of sight have been created, in particular cameras (what we call throughout this work as "conventional cameras"[1], to distinguish from event cameras, explained briefly). However, conventional cameras do not exactly mimic animal's visual system. They

[1]In this work, conventional cameras refer to cameras that produce images (frames) at a constant framerate, as opposed to event cameras, that produce events. Though some authors may consider this group too broad (for instance, that fisheye cameras should not be treated as "conventional"), we merely focus on output creation.

are much slower (typically conventional cameras produce around 20-30 fps), produce redundant information and are much more energy-costly. Furthermore, they are not very good with scenes with high contrast (as detail is lost in bright and dark areas), or with high movement (as images produced suffer from motion blur).

Neuromorphic hardware appears as a bio-inspired approach to hardware development that tries to replicate the advantages of animal systems, either in speed, energy efficiency, or any other positive or desirable attribute. Examples of such hardware are audio sensors ([Liu et al., 2013]), signal processors ([Giulioni et al., 2015] and [Qiao et al., 2015]), and vision sensors ([Mead and Mahowald, 1988]).

The work described within this dissertation focuses on the Dynamic Vision Sensors type of neuromorphic cameras (DVS cameras [Lichtsteiner et al., 2008]), which are a type of event cameras that report changes in the brightness captured by each pixel (precisely, the log-intensity of the brightness captured by each pixel). Unlike conventional cameras (that record a sequence of the intensity of all pixels in the scene, and therefore produce redundant information, and are not energy efficient), event cameras produce “events”, which contain the timestamp, pixel location, and polarity of the change in the pixel.

This approach has multiple advantages, such as 1) lower latency, because there is no need for video compression at the camera level, 2) higher energy efficiency, 3) no redundant information, 4) higher temporal resolution, in the order of microseconds, as opposed to milliseconds of conventional cameras, and 5) higher dynamic range, to name a few.

Of course, it is not without its disadvantages. First of which is the different mindset that needs to be embraced. Data is not received at a constant rate, for starters, and the nature of data itself (events) is foreign and takes some time to adapt. Conventional cameras “like” to be still in the sense that there is no movement contaminating the readings, and allows for feature extraction and initialization of the system, for instance. This is the complete opposite for event cameras: no movement means no data (assuming constant brightness in the scene), and output is dominated by (salt and pepper, mostly) noise. Furthermore, though it is possible to produce frames from events by means of image reconstruction, the result is not perfect (see Fig. 1.2 for an idea). Lastly, a new type of noise appears - time noise - where events have some deviation from the exact moment at which it was generated.

With the advantages of event cameras in mind, we set out to develop a system that is able to estimate the pose (position and orientation) of a system based on event cameras. Though this problem is well-documented and studied using conventional cameras, approaches using event cameras are still being researched, and no solution is exactly “trivial”, as events are not yet very familiar, and how to use them is not obvious. For reference, Fig.1.2 shows the accumulation of events produced when a pen is moving in front of a camera, and when a person is waving their hand.

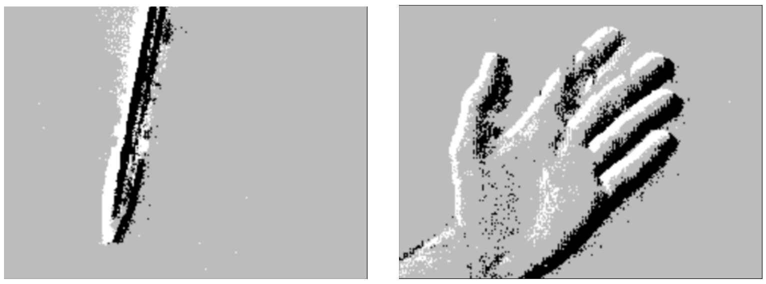


Figure 1.2: Outputs of event cameras under various situations, a pen moving in front of the camera (left), and a person’s hand waving in front of the camera (right). White and black denote the polarity of the event being represented, in particular positive and negative, respectively.

This work appears integrated in the ORIENT project, that focuses "[$\cdots$] on Neuroscience, with a goal to better understand how the brain coordinates movements in the eyes and head, in order for humans to orient themselves in the world and in relation to any object that might be around" [2]. In particular, this work appears as a study on the possibility of using an event camera to estimate the orientation of the eye.

1.1 Overview of Related Works

Event cameras have demonstrated to be useful in multiple tasks where speed is paramount, of which quadrotor control comes immediately to mind ([Mueggler et al., 2014], [Falanga et al., 2020], [Sun et al., 2021], and [Sanket et al., 2020]). They have also been adopted in areas such as flow estimation ([Akolkar et al., 2018] and [Ieng et al., 2017]), image and video reconstruction ([Rebecq et al., 2019]), and depth estimation ([Rebecq et al., 2018a]), to name a few areas.

The problem of pose estimation shares some goals and similarities with SLAM (Simultaneous Localization and Mapping), as one of the problems is that of localization, which relies on a correct estimation of the pose of the system. According to [Gallego et al., 2019], the first work on camera tracking with an event camera was presented in [Weikersdorfer and Conradt, 2012], and proposed an implementation based on particle filters, but was limited to a planar motion. [Censi and Scaramuzza, 2014] proposed an approach based on Bayesian filters, and the use of a second, conventional camera, but subject to the same limitations of planar motions.

[Cook et al., 2011], [Kim et al., 2008], [Gallego and Scaramuzza, 2017], and [Reinbacher et al., 2017] proposed several implementations which, event though limited to rotation, and therefore without the need of translation or depth, paved the way for more complex implementations, as the ones in [Gallego et al., 2018], [Kim et al., 2016], [Rebecq et al., 2017], and [Vidal et al., 2018], of which we consider the last to be the state of the art in terms of localization using event cameras.

Multiple authors have opted for approaches that try to rely on bridging the "classic" approaches with event cameras. For example, [Zihao Zhu et al., 2017] relies on features tracked by [Zhu et al., 2017], and combines them with IMU information by means of the Extended Kalman Filter (EKF) presented in [Mourikis and Roumeliotis, 2007]. Our proposed approach borrows from this idea. Other methods ([Rebecq et al., 2017]) rely on image reconstruction from [Gallego and Scaramuzza, 2017], and then use classical approaches ([Lucas et al., 1981] and [Rosten and Drummond, 2006]).

The lack of a standardised dataset for event cameras makes it difficult to compare different methods, however [Mueggler et al., 2017b] introduces a public dataset that can be used to compare methods, that some authors have chosen to use.

On the "conventional" side of pose estimation, multiple SLAM approaches based on vision (Visual SLAM, or vSLAM) are worth mentioning, such as FastSLAM ([Montemerlo et al., 2002]), ORB-SLAM ([Mur-Artal and Tardós, 2017]), EKF-SLAM, and GraphSLAM ([Thrun, 2002]). These approaches rely on conventional cameras to provide features that are used to construct a map and also be used as reference for the estimation of the pose of the system.

1.2 Problem Formulation

This work appears in the sequence of the previous works in the ORIENT group (in particular [Martins, 2019]) where the orientation of an eye is estimated by means of visual odometry. The eye produces very fast (under

[2] https://welcome.isr.tecnico.ulisboa.pt/orient-project-collaboration/

200 ms) and short (usually under 25 deg) movements, called saccades, on the order of 700 deg/s ([Luo, 2015]). This poses some challenges for conventional cameras, that are susceptible to motion blur.

It is the goal of this work to design and suggest methods that may be used to solve this problem of eye orientation, by focusing on the larger problem of localization using event cameras. For instance, though not relevant in the context of an eye, we decided to include translation estimation in the state, not only to allow comparisons of our approaches with others, but also to try and contribute with a more versatile approach, eventually even to the state of the art.

We propose an approach that leverages visual and inertial information, by means of events cameras with frames, events and Inertial Measurement Unit (IMU), in order to estimate the pose of the system using an Unscented Kalman Filter (UKF) based on Lie groups. We then propose a second approach as an improvement of the first, that leverages the information of the current estimated pose to improve the quality of the visual information.

1.3 Motivation for our Approach

Our approach is inspired by SLAM approaches, that simultaneously create a map while localizing the system in it, of which ORB-SLAM and EKFSLAM are common examples. Though our implementation is not that of SLAM, since no map is being generated (only a local map relevant for localization), most concepts are still valid, in particular we borrowed the idea of a state that contains the pose of the system (and some other useful variables) that is constantly being estimated.

Multiple formulations could have been used to tackle this problem, so why did we choose the approach here presented? In particular, why combine visual and inertial information, why use event cameras, why the choice for Kalman filtering, ..., the list goes on. We want to explain some of our choices in a few paragraphs.

Why event cameras? The first question that is interesting to address is why use event cameras. Throughout Section 2.2, many of the advantages (and current limitations) of event cameras are presented. This work chose to use event cameras as a way to analyse what is currently possible, and try to contribute to the state of the art. Part of the reason for event cameras has to do with the working principle, and the high temporal resolution, which becomes an important factor in the context of eye saccades.

Part of this work's objective is to evaluate the feasibility of using event cameras in the context of the estimation of saccadic movement. Saccades have a very high angular velocity (in the order of hundreds of degrees per second). Normal cameras cannot keep up with this speed, and the resulting images are corrupted by motion blur.

As such, event cameras appear as an interesting alternative that helps solve this limitation of conventional cameras (at the expense of the need for newer (or, at least, different) methodology).

Why visual inertial odometry? As stated, this work is part of the ORIENT project, where there is a strong emphasis placed on modelling, describing and/or mimicking biologically inspired systems, both to gain insight into the biological system itself, in particular the eye, and to leverage as much of natural evolution as possible into artificial systems.

With this in mind, we turn to the biological solution that animals (in particular humans) have converged to. Oversimplifying, humans have multiple ways to know where they are in relation to the world. We have proprioception, which means we have a sense of where our limbs are even when we are not actively observing them (we do not need to look at our legs to know how much we have walked), we have much prior information (we know how fast we move based on our gait, leg length, walk/run), acquired through years of learning, and we have some other "tricks" to accurately position us in the world.

However, we want to highlight two systems in particular: the vestibular system, and the visual (ocular) system, both of which are partially explained (Sections 3.2 and 3.3).

The interesting aspect of these two systems is that they are not isolated, and evolution has coupled them together through the vestibulo-ocular pathway (evidenced in the vestibulo-ocular reflex). This reflex is easy to demonstrate: if a person focuses a point in the world, and moves their head, the point remains in focus, and our eyes move to compensate the head movement. This happens because the vestibular system detects the movement, sends this information to the vestibular nucleus in the brainstem, and this information is used to make the eyes compensate the movement and maintain the point in focus (Fig. 1.1).

This coupling between these two systems is so tight, that when there is a mismatch, our brain believes it was poisoned, hence sea sickness, for instance. Furthermore, dizziness after multiple spins (pirouetting) is caused by persistent movement of crystal in the vestibular systems after continuous stimulation, which is interpreted as movement, and the eyes try to compensate this (non-existent) movement, resulting in the world rotating around us.

Another reason to use visual and inertial information is that they complement each other. In Section 3.2, the major limitation of purely inertial odometry systems is that there are no correction mechanisms, and they drift over time. However, the visual component can serve as a correction (for example, imagine you are blindfolded, riding in a car; turns and accelerations can be detected (inertial information), and you can have a sense of where you are, but eventually, you lose your bearings and get lost; however, if you were allowed to peek at your surroundings from time to time, you could correct the drift you suffered over time), which would increase the reliability of the inertial system over time, by turning it into a visual inertial odometry system.

Furthermore, a monocular (single camera) system is not capable of obtaining the absolute dimensions of a scene, only the relative dimension. The inertial system, on the other hand, is able to have a sense of the absolute values of the movement. Combining both systems should produce better estimation with a better sense of dimensions. As such, there is a synergy to be explored by combining these two systems.

Finally, the IMU is already embedded in the event camera, and there is no *a priori* reason not to take advantage of it.

Why not just vision? Some of the reasons are explained in Sections 3.2 and 3.3, but ultimately culminate in the synergy between complementary systems that allow them to overcome their individual limitations. From the biological standpoint, the coupling between these two systems is evident, shown by the vestibulo-ocular pathway, as explained previously.

From our standpoint, we wish to leverage the inertial system, which responds very rapidly, but is prone to drifting, and the visual system, which is slower, but less precise. Another relevant factor is the "unknowns" associated with event cameras. The features extracted can be less robust (or at least, less in number) than conventional features, which means that a vision-only approach may not be reliable on its own. As such, a vision-only approach, though possible and very successful using conventional cameras ([Mur-Artal and Tardós, 2017]), was not selected for this work.

Why the need for data fusion? We want to use two data sources (the visual and the inertial components) in a way that the resulting system is more accurate or consistent. This means that the information from each system needs to be somehow combined. Multiple solutions can be used for this fusion, of which filtering is a common approach. Again, in the field of filtering, multiple filters can be used with varying degrees of complexity. We choose to approach this problem with an UKF, as it is a technique that is known to produce good results, and which, to the best of our knowledge, has not been used in combination with event cameras.

1.4 Contributions

We propose a Lie group-based UKF approach to solve the pose estimation problem which, to the best of our knowledge, is a novel approach in the context of event cameras. This approach attempts to combine visual information in the form of events and frames, with inertial information obtained from an IMU, by means of an Unscented Kalman Filter.

Also, we propose the use of the Event-based Kanade-Lucas-Tomasi tracker (EKLT, [Gehrig et al., 2020]) in the context of pose estimation, which, to the best of our knowledge, has not yet been done before.

Furthermore, we list, implement and compare possible ways to improve said tracker by taking into account the current estimation of the pose, in effect improving sensor output by combining it locally (at the sensor level) with measurements from outside the sensor, which is an unusual approach to improving sensor reading.

Through this work, multiple tools have been developed, such as trajectory generators for simulations, motion capture scripts, simulations, datasets, and the UKF implementation itself, which we hope can be of use for future works.

1.5 Thesis Structure

Chapter 1 introduces the problem to approach in the thesis, in particular presents a short discussion on the problem of pose estimation being tackled, the state of the art on event cameras, and pose estimation. Chapter 2 presents the mathematical and engineering background needed to understand the concepts being introduced and/or used in the context of this work. Chapter 3 and Chapter 4 introduce our proposed approaches. Chapter 5 provides an overview of the different experiments executed as well as the results attained and a critical analysis of these results. Chapter 6 summarises the work performed and highlights the main achievements. Moreover, this chapter proposes further work to extend the activities described in this document.

Chapter 2

Background and State of the Art

"Ex nihilo nihil fit" (Nothing comes from nothing)

— Parmenides

In this chapter, we provide an overview of the necessary mathematical basis for the understanding of this thesis, as well as the works being done in the area that either helped this work, or serve as a possible comparison. In particular, Section 2.1 explains how cameras in general work, and introduces the concept of image features (what they are, how they are detected, and how they are tracked across multiple camera frames). Section 2.2 introduces event cameras, novel cameras that have a different working principle than conventional cameras, and are the basis of this work. Section 2.3 introduces the Inertial Measurement Unit (IMU) and its sensor readings. Section 2.4 introduces the problem of pose estimation, and the way camera movement changes the information captured by the camera. Lastly, Section 2.5 contains the state of the art in terms of pose estimation, with emphasis on event cameras.

2.1 Camera Projection Model and Imaging Features

In this section we introduce the typical pinhole camera projection model that is the basis for image formation, and introduce the concept of image features, which are keypoints in the image that are particularly distinctive and useful.

2.1.1 Camera Model

An important concept to take into account is that of camera model, which models the correspondence between points in the world and their position in image space. A common model used is the perspective projection model, in particular the pinhole model (Fig. 2.1). In this model, the mapping from world (3D) to camera (2D) is performed by tracing a ray of light from the world, through an infinitesimally small aperture (pinhole), and into the image plane. Important parameters in this model are the focal distance (distance from aperture to image plane), and image (or optical) centre (centre of the image plane), which are both intrinsic parameters.

From this model, we can derive the perspective projection geometry (Fig. 2.2), where the image plane is modelled in front of the optical centre, and the optical axis is orthogonal to the image plane. In this model, lines are

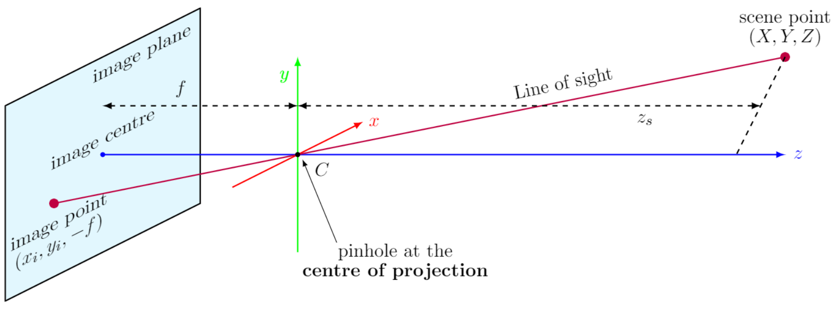


Figure 2.1: Pinhole camera model

projected to lines, collinear features remain collinear, and tangents and intersections are preserved, but parallel lines (in the world) eventually meet at a vanishing point, since angles are not preserved.

This geometry makes it clear that projection between world and image are given by the triangular similarity

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.1}$$

where X, Y, and Z represent 3D coordinates in the real world, x and y their corresponding projected 2D points, and f the focus length of the camera.

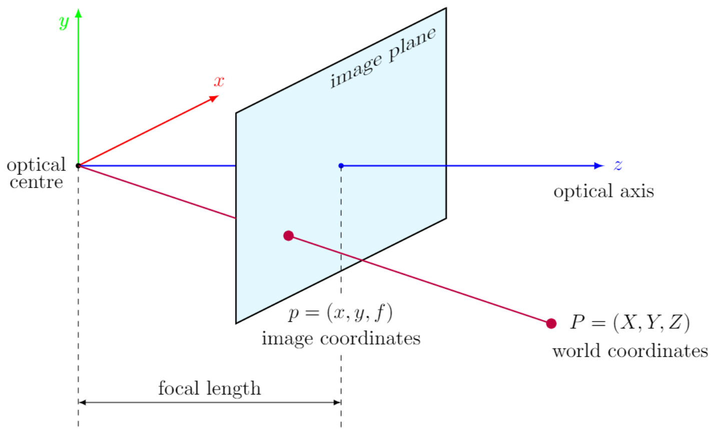


Figure 2.2: Projective geometry

The intrinsic parameters combine these properties, namely focal length (f), offset to the optical centre (c_x and c_y), and skew (s, from non-orthogonality between optical axis and image plane), and constitute the intrinsic parameters matrix K, defined as

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

This matrix is also present in the projective matrix P, which relates the points in world space to their corresponding image plane position, given by

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

which is critical for computer vision.

Assuming camera rotation and translation (extrinsic parameters, since image and world frames are not centred), the projective can be expanded to include rotation R and translation t, and becoming completely generic, in particular

$$\mathbf{x} = K \begin{bmatrix} R & t \end{bmatrix} \mathbf{X} \quad . \tag{2.4}$$

This is called the camera matrix. The calibration process to obtain this matrix is explained in Appendix C. Though the calibration process for both cameras (conventional and event) follow the same general idea of finding a set of points in the image from multiple points of view, and use them to estimate the parameters, the acquisition of these points differs a bit (due to the nature of the cameras, and also their resolution). The appendix explains these changes.

2.1.2 Feature Detection and Tracking

In the context of computer vision and image processing, image features are distinctive landmarks in the images, preferably insusceptible to point-of-view, scale, and the aperture problem. Features are important as they provide information on the image, which can be used for recognition, matching, reconstruction, and tracking, among many other applications. Many types of features can be considered, such as edges, corners, blobs, ridges, and shapes. In this section, we aim to explain the choice of features that are typically used, from the conventional cameras perspective. Approaches for event cameras are described in Section 2.2.2.

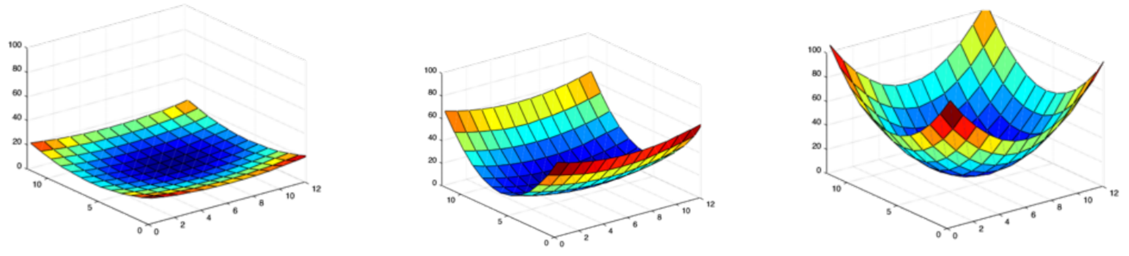
2.1.2.1 Feature Detection

The process of identifying features in an image is called feature detection, and multiple detectors have been described in the literature, dependent on the features of interest, such as Canny and Sobel detectors (for edges), Hough transform (for shapes), Laplacian operator (for blobs), and Harris detector (for corners).

Corners and Gaussian Curvature Let us consider the surfaces $E(u, v)$, defined as

$$\begin{aligned} E\left(u, v\right) &= \begin{bmatrix} u \, v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \\ M &= \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \end{aligned} \tag{2.5}$$

and apply it to an image. Three particularly interesting situations can arise, as shown in Fig. 2.3. In particular, uncharacteristic areas in the image produce flat zones, since the gradients of such zones are small. Edges have one direction of greater change, as one of the gradients is high, but not the other, producing a “valley-like” surface. Finally, since corners represent the intersection of edges, two directions of gradient produce a higher response, resulting in a clear depression with the corner in the center.



(a) flat zone (b) edge zone (c) corner zone

Figure 2.3: Gaussian curvature around different points. (a) shows a flat zone, two low-value eigenvalues, (b) shows a edge zone, one significantly-large eigenvalue, and (c) shows a corner, two significantly-large eigenvalues.

The Gaussian curvature then measures the principal curvatures of a surface at each point, effectively measuring how each surface bends. As such, corners are points where the surface bends in both directions.

Harris Corner Detector The typical example for a classic corner detector is the Harris Corner Detector ([Harris et al., 1988]). It works using the following steps: 1) Compute the x-wise $I_x(x, y)$ and y-wise $I_y(x, y)$ partial image derivatives, 2) Compute the second-order derivatives $I_x^2(x, y)$ and $I_y^2(x, y)$, and cross-derivatives $I_xI_y(x, y)$, 3) Compute the second-moment matrix $M(x, y)$, 4) Compute the Harris score, and 5) Detect local extrema whose Harris score is greater than the set threshold.

The partial derivatives are computed by applying a Sobel derivative kernel (usually 3x3 or 5x5 kernels) to the whole image, producing the x-wise $I_x(x, y)$ and y-wise $I_y(x, y)$ partial image derivatives. From these derivatives, we can define the vector $\nabla I(x, y) = (I_x(x, y), I_y(x, y))^T$ and the second-moment matrix for each pixel ($M(x, y)$), defined as $M(x, y) = \sum_{(x,y)\in patch} g(x, y)\nabla I(x, y)\nabla I^T(x, y)$, where $g(x, y)$ is a Gaussian weighting function centred around (x, y), which controls the “sharpness” of the edge.

Then, we can compute the Harris score as defined by

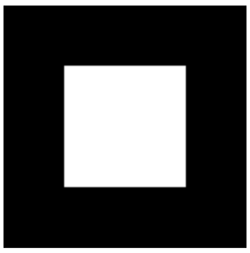
$$H(x, y) = \lambda_1\lambda_2 - k \times (\lambda_1 + \lambda_2)^2 = \text{det}(M) - k \times \text{trace}(M)^2 \tag{2.6}$$

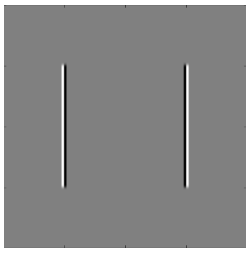
where k is an empirical value, $k \in [0.04; 0.06]$.

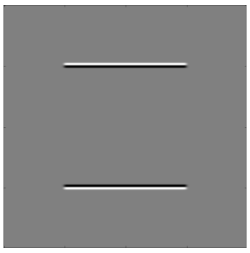
Finally, if $H(x, y) \geq H_0$, we consider the pixel as a corner. This will produce corner blobs. In order to select a single pixel to represent the corner, we select the local extrema (the pixel with the highest Harris score). Fig. 2.4 shows these steps of image differentiation, computing the Harris score, and identifying corners.

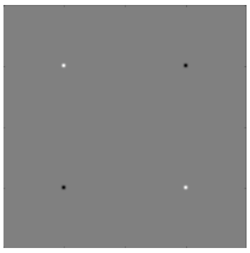
Another option to evaluate the presence of corners is to analyse the eigenvalues of M. If both eigenvalues are low, no interesting features are detected. If one is low, but the other is high, we are in the presence of an edge. Lastly, if both eigenvalues are high, the pixel is likely a corner. As such, one interpretation of the Harris detector is that corners are identified by finding the intersection of edges.

With this in mind, an alternative for the corner analysis, is to check the value of the lowest eigenvalue of $M(\lambda_{min})$, through the approximation


(a) original image


(b) second-order x derivative


(c) second-order y derivative


(d) cross derivatives

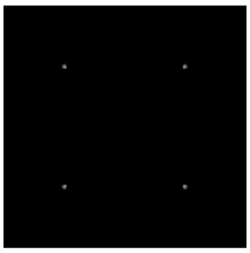

(e) detected corners

Figure 2.4: Illustration of the identification of corners through Harris corner detector. (a) shows the original image, and (b) and (c) the horizontal and vertical derivatives I_x^2 and I_y^2, respectively. (d) cross derivative I_xI_y, and (e) shows the identified corners of the original image.

$$\lambda_{min} \approx \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2} = \frac{\text{det}(M)}{\text{trace}(M)} \quad . \tag{2.7}$$

And then this value is used as a corner criterion. This relation between corners and eigenvalues is not evident, and is related to the Gaussian curvature of a surface at a point. As stated previously, corners are points where the surface bends in both directions, with corresponding high eigenvalues. This is why a valuable approach to corner detection is the analysis of the eigenvalues, as we want to identify corners that generate a great curvature, and, consequently, have higher eigenvalues.

SIFT and SURF features and descriptors SIFT (Scale Invariant Feature Transform, [Lowe, 2004]) and SURF (Speeded-Up Robust Features, [Bay et al., 2006]) are two well-known feature detectors used for conventional cameras. These detectors have been documented and their principle of working is not relevant for this work. However, even though extracting features from the image is an important step, another crucial step is matching the features that were detected between consecutive frames. To facilitate this matching, SIFT and SURF features have an accompanying descriptor, which serves as a sort of label to identify each feature, and allow for easier and faster matching of features between frames. This concept of descriptor is very powerful and useful, as it means there is objective information that can be used to match features. Taking SIFT descriptors, for examples, each feature has a descriptor, corresponding to a grid of 4x4 blocks , each containing 4x4 sub-blocks. For each sub-block, the feature's main orientation is estimated, and all 16 sub-block orientation are condensed into a bin with 8 directions. As such, each feature has a 4x4 blocks x 8 orientation bin which creates a 128D descriptor.The gradient magnitude m_g and orientation θ, which is used for the descriptor is given by

$$\begin{aligned} m_g(u,v) &= \sqrt{(L(u,+1,v) - L(u-1,v))^2 + (L(u,v+1) - L(u,v-1))^2} \\ \theta(u,v) &= \tan^{-1}\left(\frac{L(u,v+1) - L(u,v-1)}{L(u,+1,v) - L(u-1,v)}\right) \\ L(u,v,\sigma) &= G(u,v,\sigma) * I(u,v) \end{aligned} \tag{2.8}$$

where G is a Gaussian kernel and I the image.

2.1.2.2 Feature Tracking

Having identified features, an important step is being able to match them or track them across multiple frames. As such, it is important to choose features which are unique and not easily mismatched (distinct features), as well as features that are easy to reidentify from multiple angles (robust and stable features, invariant to viewing direction and distance, and illumination variances).

Sum of Squared Distances (SSD) A simple approach is to use a global minimization method through the Sum of Squared Distances (SSD) between the features being proposed as a match, with the underlying principle that matching features are close to each other in consecutive frames. However, this is not always the case.

Also, this approach complicates feature reidentification whenever it exits and re-enters the frame, and is also slow. Furthermore, there is no guarantee that the exact same number of features has been detected, which further complicates this approach.

This technique is better suited for template tracking across frames, assuming rigid body motion, which is not always the case, with independent feature movement.

Descriptors To facilitate feature matching in classical cameras, along with the features detected (interest points), image descriptors around these points are also preserved, which are then compared for matching. This approach is used by Scale-Invariant Feature Transform (SIFT [Lowe, 1999]), Speeded-Up Robust Features (SURF [Bay et al., 2006]) and similar approaches.

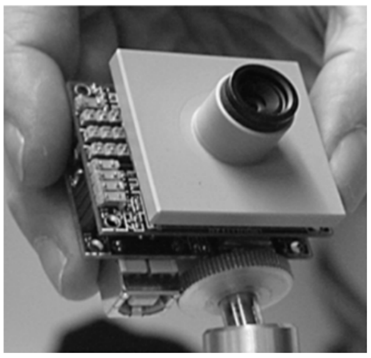
2.2 Event Cameras

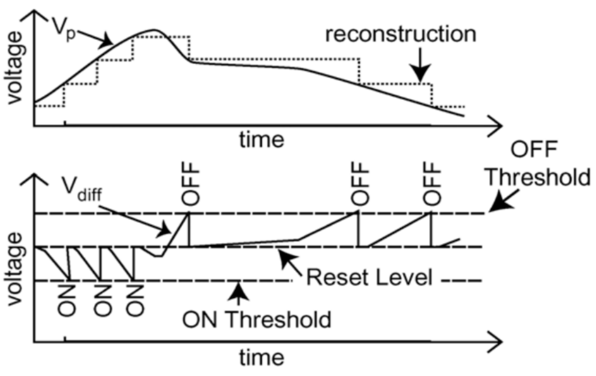
Event cameras, also called neuromorphic cameras, silicon retina or Dynamic Vision Sensor (DVS) are image sensors that respond to changes in brightness in the scene. Unlike conventional cameras, which capture full image frames at a fixed frequency (commonly 30Hz or 60Hz), producing redundant information and requiring a high bandwidth for transmission, each pixel in an event-based camera operates independently and asynchronously, reacting to changes of brightness in the scene, eliminating the transmission of redundant information, allowing for much higher temporal resolution (in the order of microseconds, as opposed to the milliseconds of conventional cameras).

Event cameras are inspired by the behaviour of the cells in the retina. Though oversimplified, retinal cells respond to changes in the environment (namely brightness), generating an electrical impulse. The transient response of each retinal cell is independent. Event cameras mimic this behaviour by asynchronously and independently responding to changes in brightness in the environment, generating ON/OFF events each time a predefined threshold in brightness is exceeded.

This architecture allows for interesting properties, such as microsecond temporal resolution, high dynamic range (above 120dB), which allows for scenes with both bright and dark zones, and does not suffer from under/overexposure, nor motion blur.

Events are triggered when the brightness in a certain pixel surpasses a certain threshold. In particular, discrete brightness steps are pre-defined, and whenever brightness detected crosses the threshold, an event is generated. Positive crossings generate ON events, and negative crossings generate OFF events. In effect, each pixel is constantly working as a comparator (with corresponding electronic to support this mode of working). Fig. 2.5 shows the first-generation DVS sensor, with an array of 128x128 pixels (from [Lichtsteiner et al., 2008]), where this idea of comparison and threshold is shown.


(a) DVS128 camera



(b) principle of working

(c) image sensor

(d) DVS240 event camera

Figure 2.5: First generation event camera (a) and corresponding principle of operation (b), showing the thresholds and the corresponding asynchronous generation of ON/OFF events, from [Clady et al., 2015]. A more recent model, the DVS240 camera, (c) shows the image sensor, and (d) shows the provided lens attached to the camera.

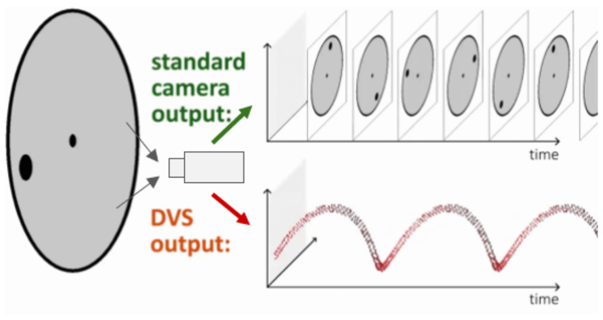
Events are then defined as a four-component vector:

$$\mathbf{e} = \left((x, y)^T, t, pol\right)^T = (\mathbf{p}, t, pol)^T \tag{2.9}$$

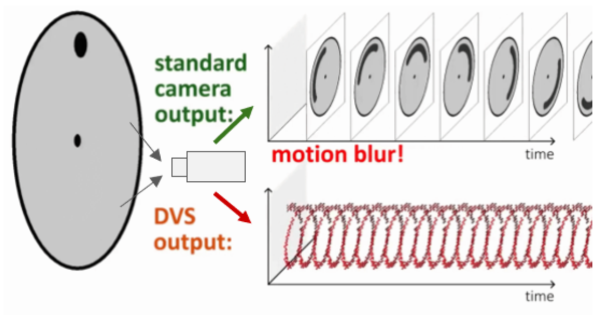
The component $\mathbf{p} = (x, y)^T$ refers to the spatial position of the event in the camera. The component t refers to the timestamp of the event, and is of extreme importance due to the microsecond temporal resolution of the camera. Lastly, the parameter *pol* refers to the polarity of the event (ON/OFF events).

With this event structure, it is common to represent events in a three-dimensional (space-time), representation, as shown in Fig. 2.6.(d), which shows the space-time evolution of events generated from a rotating black bar on a white background, over a period of 1000 ms.

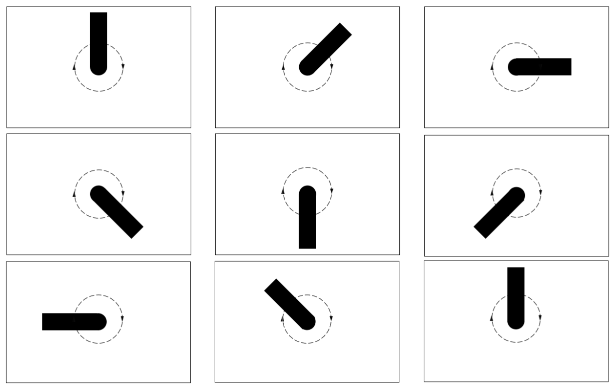
Conventional cameras and event cameras have fundamentally different modes of operation and output. As such, a comparison of the behaviour in the same scene, and an analysis of the output, is interesting. Fig. 2.6 shows the response of both a conventional camera and an event camera when presented with a disk with a black dot rotating at a high speed. The fixed capture rate of the conventional camera is unable to keep up with the speed of the dot, and the images suffer from motion blur and some discontinuity between frames. The event camera, however, due to its asynchronous event generation and temporal resolution, is able to continuously produce events relating to the movement of the dot.



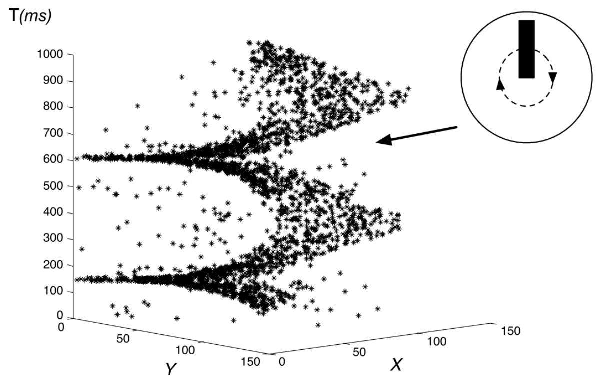
(a) comparison of outputs, no motion blur



(b) comparison of outputs, with motion blur



(c) rotating bar used for (d)



(d) spatio-temporal evolution of events

Figure 2.6: Comparison of the output of a standard camera (above), and an event camera (below), when recording a rotating disk with a black dot, adapted from [Mueggler et al., 2017a]. (a) shows a lower speed rotation; (b) shows a higher speed rotation, with motion blur from the conventional camera. Also, space-time representation of events (d) generated from a rotating black bar (c), from [Clady et al., 2015].

Advances in camera manufacturing have allowed for cameras that have both conventional camera pixel arrays, and event camera pixel arrays. This enables hybrid algorithms, which take advantage of the benefits of event cameras, with the extensive research on conventional cameras.

2.2.1 DVS240 and DAVIS240 Event Cameras

A DVS240 event camera (Fig. 2.5) was available to us and used for part of this work ([1]). It is an event camera that is also capable of full frame greyscale recording (but not simultaneously with events, as only one can be active at each time; this mode was mainly intended for calibration purposes). The camera has a resolution of 240x180 pixels and comes with an adjustable length lens that changes the focal length (from 3.5mm to 12mm) and field of view (FOV) (from 64.6 deg horizontal and 50.6 deg vertical, to 20.9 deg horizontal and 15.7 deg vertical).

A more powerful camera, the DAVIS240, allows for the simultaneous recording of events and frames, and is the camera used in some of the datasets that were used ([2]), and is therefore worth mentioning. The rest of the specifications are similar to the DVS240.

Lastly, ESIM ([Rebecq et al., 2018b], [3]), an event camera simulator was also used in this work, replicating a DAVIS240 camera, and is explained further in Section 5.1.

2.2.2 Feature Detection and Tracking on Event Cameras

For event-based cameras, new types of features, as well as detectors, are being proposed, as classical techniques are not easily transferable in most cases, or result in a non-negligible performance decrease, due to conversion overhead from asynchronous events to frames. However, corners seem to prove as interesting features to use, as not only can they be detected using both conventional and event cameras (and can therefore be matched between the event stream and full frame images), but also algorithms considering only events are available, which leverage the potential of events, namely speed and event independence.

Due to the nature of events, gradient operators are not possible (at least directly applied to the event stream), since there is no image on which to apply them, and multiple techniques have been proposed, of which three are considered.

2.2.2.1 Feature Detection

In this section we explain the techniques proposed for feature detection using events, in particular Space-time detection, Event-based Harris Corner Detector, and SAE-based corner detector.

Space-time detection This method relies on the space-time properties of events, and creates a 3D representation, containing the spatial position of an event (x, y), as well as the time it was received. In this representation, edges moving with uniform linear speed create planes (stack of lines at different instants), and corner movement creates lines (stack of points at different instants).

As such, this technique tracks moving edges by fitting planes in this 3D representation, implicitly estimating the speed of the moving edge (optical flow). Each new event is matched to the previously estimated planes, and the estimates are updated, as shown in Fig. 2.7.

The way this technique identifies (and tracks corners) is by detecting intersection between these planes, as these intersections correspond to the corner movement through a period of time (Fig. 2.8).

[1] `https://inivation.com/wp-content/uploads/2020/04/DVS240.pdf`, accessed last 2021/07/25
[2] `https://inivation.com/wp-content/uploads/2019/08/DAVIS240.pdf`, accessed last 2021/07/25
[3] `https://github.com/uzh-rpg/rpg_esim`

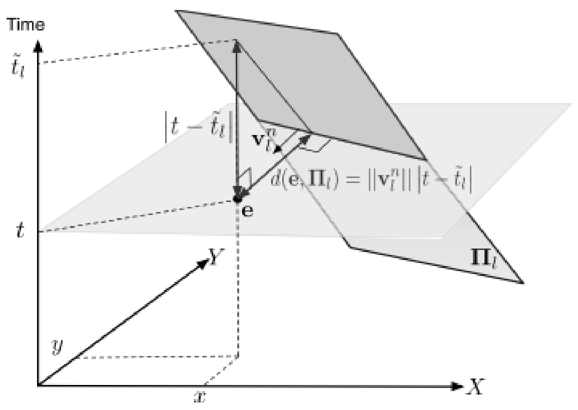


Figure 2.7: Distance calculation from event to plane, and plane fitting, from [Clady et al., 2015].

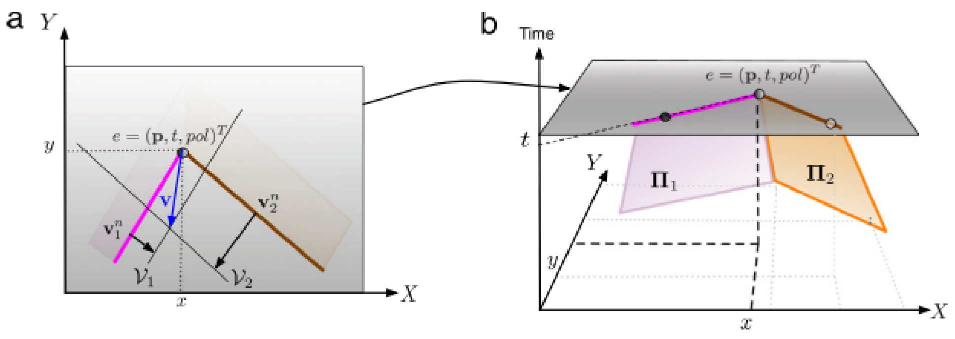


Figure 2.8: Corner are the result of the intersection of lines (edges) at a given timestamp, which themselves result from the space-time planes that are being estimated, from [Clady et al., 2015].

Event-based Harris Corner Detector This technique (from [Vasco et al., 2016]) relies on the Surface of Active Events (SAE), a representation system for events, which keeps track of the timestamp of the most recent event for any given pixel, regardless of polarity, defined by

$$SAE : (x, y) \to t \tag{2.10}$$

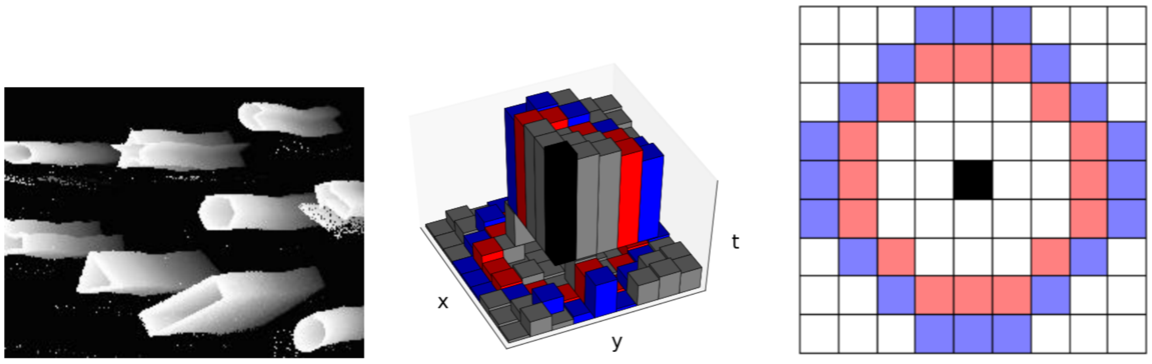
Indeed, it is a spatial representation ((x, y) coordinates, corresponding to each pixel), which can be discretized by assigning a value to each pixel based on its timestamp. Fig. 2.9 shows an example of the SAE, where the events are represented from white to grey, as they go from more recent to older.

Since this discretized representation is now a frame in the classical sense, we can apply the Harris Corner Detector directly to the SAE and identify the corners from these results. A more efficient implementation relies on considering only the neighbouring region of an event as it arrives, instead of the whole SAE. As such, only a subset of the SAE is analysed. Since each event, and consequently, each subset, is independent on the other subsets (provided the subsets do not overlap), parallel implementations are possible, and also improve speed.

SAE-based Corner Detector This technique also relies on the SAE representation of events but does not perform any computations. Rather, it performs only comparison operations on a local neighbourhood around the relevant event.

As each event is received, its timestamp is compared with the neighbouring pixels using circular segments (for isotropic response and efficiency), and checked if patterns similar to the one in Fig. 2.9 are present (contiguous pixels with decreasing timestamps), as these are typical corner patterns.

Though this method is not as effective, it is much faster, as no computations are performed, and each event can be processed independently (and concurrently in a parallel fashion).



(a) SAE representation (b) comparison filter applied (c) comparison filter

Figure 2.9: (a) Representation of SAE, on which the comparison filter from (c) is applied, and checked for patterns of decreasing intensity from centre to surrounds, as shown in (b), from [Mueggler et al., 2017a].

2.2.2.2 Descriptors

For event cameras, a typical choice of features, as previously presented, are corners. However, the choice of descriptors for matching and tracking is not yet as developed as for classic cameras, though some descriptors based on time properties of events, or their distribution and neighbourhood, have been proposed ([Cohen, 2015]).

2.2.2.3 Feature Tracking

In this section we explain the techniques proposed for feature tracking using events, in particular Sum of Squared Distances (SSD), Spatio-temporal tracking, and Event-Based Kanade-Lucas-Tomasi Tracker (EKLT).

Sum of Squared Distances (SSD) A first approach to feature matching across frames is to rely on the fast nature of events, and to create pseudo-frames[4] by combining features over a timeslice (accumulation of events for a given interval). The features of both pseudo-frames can then be matched using SSD, assuming a proximity between features in consecutive pseudo-frames.

Nevertheless, this approach is not ideal, as a critical parameter is the time of integration in the timeslice, which does not take full advantage of the nature of events and event cameras, and could be so slow as to have the same temporal resolution as conventional cameras.

[4]In this context, and throughout this work, pseudo-frames are defined as an accumulation of features detected or tracked using events, in effect generating a batch of features that can be compared to a batch of features extracted from a conventional camera.

Spatio-temporal Tracking The technique presented in Section 2.2.2.1 for corner detection is also able to track these corners across multiple timestamps, since the plane and line fitting that are performed in the spatio-temporal representation of events for edges and corners, respectively, implicitly estimates their motion and position across time.

However, since no descriptors or identifiers are being registered, some problems arise when features leave the camera space and are recaptured later, as well as situations where multiple corners overlap and end up merging together. This is particularly true for corners from organic features, as opposed to artificial structures.

Event-Based Kanade–Lucas–Tomasi Feature Tracker (EKLT) EKLT ([Gehrig et al., 2020]) is a hybrid feature tracking technique that is able to merge information from conventional cameras and events (and hence is more suitable for the new generation of DAVIS event cameras), that tracks corners across time. The method is based on the Kanade–Lucas–Tomasi Feature Tracker (KLT), hence the name EKLT (Event-Based Kanade–Lucas–Tomasi Feature Tracker).

This method tracks corners, as they are easy to recognize in both conventional cameras (Section 2.1.2.1), and correspond to areas with high event generation, that can be detected using event corner detectors as well (Section 2.2.2.1).

The idea behind this tracker is to detect features using a conventional frame, which are then tracked using events until a new frame arrives, at which point the estimation from events is compared to the corner detection in the new frame, in essence correcting this estimation. If the feature is not detected, it is still tracked in event space, as subsequent frames may re-detect missed features. This approach is particularly useful in high-speed movements, where motion blur becomes a problem for frames, but not for events.

This comparison between frames and events is crucial, and the key concept is "image variation in a frame patch". As previously discussed (Section 2.2), event cameras respond to brightness changes in the environment. Therefore, it is not farfetched to compare events to image gradients, as zones with higher gradients in the world are precisely the ones that produce the most events. In fact, integration (accumulation) of events over a period of time produce results that are very similar to the gradient of the image, as shown in Fig. 2.10.

This is the idea at the core of this approach, as the brightness change behaves as sort of descriptor for the features, and are used as patches for a Lucas-Kanade inspired patch comparison and matching, using both the patch and estimated velocity (estimated through events), using the cost function

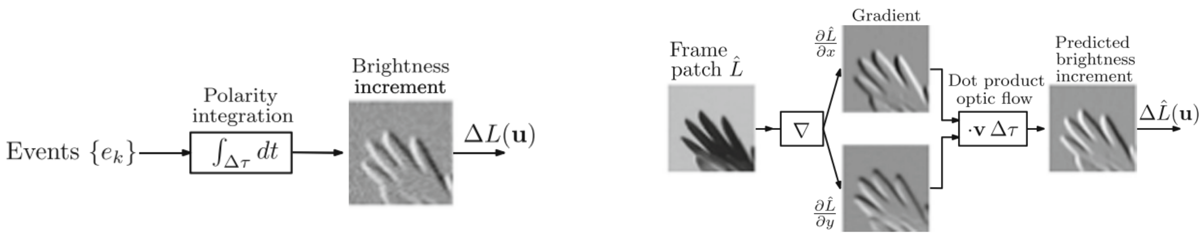
$$\min_{p,v} \left\| \Delta L(u) - \Delta \hat{L}(u,p,v)) \right\|^2 \tag{2.11}$$

where ΔL denotes changes from events, $\Delta\hat{L}$ denotes gradients from frames, and u denotes the image, p the warp parameters, and v the velocity. p and v are used as the starting values for the optimizer that minimizes the functional (2.11), and are modified during the optimization process.

While a new frame is not received, the corner is tracked in event space and the local patch is being created for comparison with a frame patch created from image gradients, as shown in Fig. 2.10.

It is worth noting, however, that the dependence on corners may present a problem for low-textured, or highly organic environments, where high quality corners are not always present.

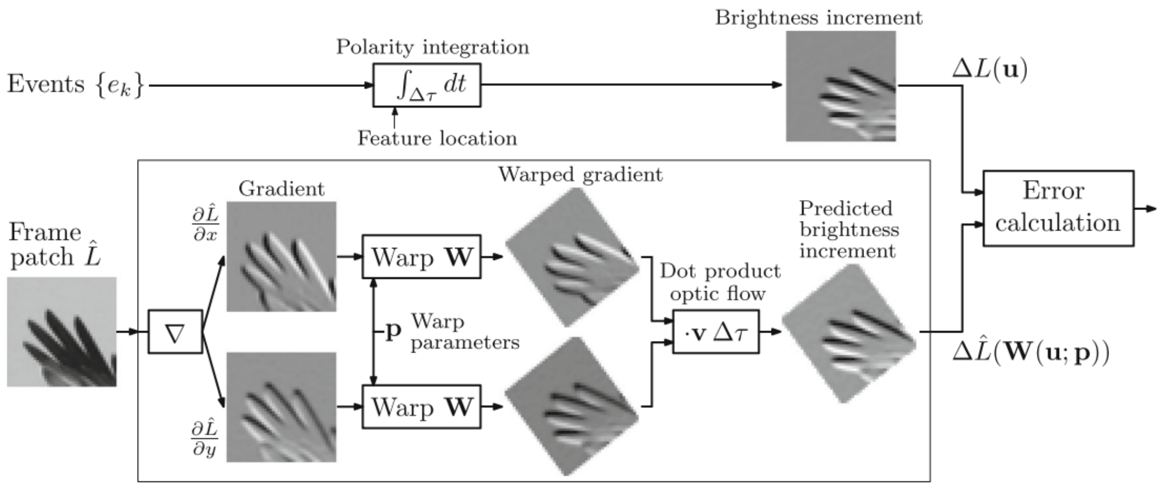
Also worth mentioning is the parameter v shown in Fig. 2.10, which accounts for the optical flow. Though inconspicuous at first glance, v is crucial for the generation of the Predicted Brightness Increment, as it estimates the flow angle (the direction objects in the image are moving), which is needed to predict the polarity of the events



Polarity integration
Brightness increment
Events $\{e_k\}$
$\int_{\Delta\tau} dt$
$\Delta L(\mathbf{u})$
Frame patch $\hat{L}$
Gradient
$\frac{\partial \hat{L}}{\partial x}$
$\frac{\partial \hat{L}}{\partial y}$
∇
Dot product optic flow
$\cdot \mathbf{v}\, \Delta\tau$
Predicted brightness increment
$\Delta \hat{L}(\mathbf{u})$

(a) template from event accumulation

(b) template from frame gradients



(c) overview of EKLT

Figure 2.10: Comparison of the brightness change from event integration (a), versus the brightness change from image gradient (b). (c) shows the block diagram of EKLT, illustrating the comparison between brightness changes from images and events, from [Gehrig et al., 2020].

and generate the template based on frames to compare against the real Brightness Increment generated from events.

This parameter is estimated by one of the following methods:

Kanade–Lucas–Tomasi Tracker (KLT) method This approach uses the classic KLT algorithm ([Lucas et al., 1981]) to estimate motion flow. The original algorithm estimates the motion flow by comparing patches between consecutive image frames (images I and T by means of the minimization of the photometric error

$$\min_{p} \left\| \left(I \circ \mathbf{W}\right)(u) - T(u) \right\|^2 \tag{2.12}$$

where $W(u;p)$ denotes a warp that maps image I to image T, and parameters p include the translation and rotation of these patches. From this warp, flow can be estimated.

We are already considering frame patches around features. This approach takes this into account and compares consecutive patches to estimate their motion, and estimate v.

Event method A second approach for optical flow estimation of the patches is based on the Brightness Constancy Formulation, with a novel approach using events.

A common approach to estimate optical flow relies on two principles: brightness constancy and small motion.

The first principle is that scene points moving through the image sequence along time remains the same, meaning

$$I\left(x(t), y(t), t\right) = C \tag{2.13}$$

that represents that a certain point x, y being tracked along time t is always constant (C).

The second principle states that the motion is small, so that we can write:

$$\begin{aligned} \text{Optical flow (velocities):} \quad & (u, v) \\ \text{Displacement:} \quad & (\delta x, \delta y) = (u\delta t, v\delta t) \end{aligned} \tag{2.14}$$

Combining both principles, we have that, for small space-time steps

$$I\left(x + u\delta t, y + v\delta t, t + \delta t\right) = I(x, y, t) \quad . \tag{2.15}$$

These assumptions yield the Brightness Constancy Equation:

$$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \quad . \tag{2.16}$$

This derivation is not obvious and is better explained in Section B. However, it is not necessary to understand the following steps.

We can then rewrite (2.16) in matrix form, in particular

$$\nabla I(u, v)^T + I_t = 0 \Rightarrow (u, v) = -\nabla I^{-1} I_t \quad . \tag{2.17}$$

This means that the optical flow (u, v) can be approximated by means of the spatial derivatives $\nabla I = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$ and the temporal derivative $I_t = \frac{\partial I}{\partial t}$.

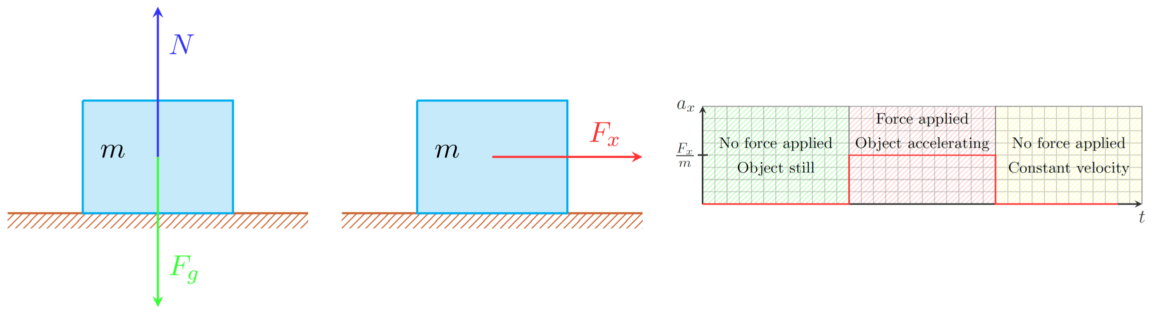
The interesting novelty in this approach is the use of events to obtain the temporal derivative, while still obtaining the spatial derivative from the frame.

2.3 Inertial Measurement Unit (IMU)

The event cameras used for this work all have an Inertial Measurement Unit (IMU), with coordinate frame matching the camera frame. As such, explaining the type of information that is produced by the IMU is fundamental.

The IMU is a sensor that reports the linear acceleration and the angular velocity of a body by means of accelerometers and gyroscopes (sometimes also the orientation by means of magnetometers, but the IMU used did not have this capacity and therefore it is not analysed).

Let us consider a square sitting on the floor (Fig. 2.11). On it acts only the force of gravity pushing down, but it is counteracted by the force of the floor on the square. Unless a force acts on this square, it remains still. Let us know consider that an horizontal force F_x acts on the square. This force will make the cube move, according to Newton's equation of force the $F = ma \rightarrow a = F/m$. Logically, the greater the force, the farther the square will move, as this acceleration is higher. An accelerometer is able to measure this acceleration that the square was subject to, and give us information on the movement of the square (that can be obtained through $x(t) = v_0\, t + 1/2\, a\, t^2$, where v_0 denotes the initial velocity (in this scenario, $v_0 = 0$)).



(a) no force applied (b) horizontal force applied (c) produced reading on accelerometer

Figure 2.11: (a) shows a square on the floor, being acted on by gravity F_g and resulting normal force N. (b) shows movement of the square from acting force F_x, and (c) the resulting reading in terms of acceleration.

The rotation can be analysed the same way, but its interpretation is not as trivial, as a rotating frame is not an inertial frame. Its analysis thus need the use of angular speed ω with moment of inertia I and angular momentum $L = I \times \omega$. Angular motion causes the gyroscope to oscillate in certain ways to maintain the angular momentum, thus giving us an estimation of the angular speed ω.

Let us now analyse the sensor readings. Starting with the gyroscope, and considering a single axis, it provides a measurement $\tilde{\omega}$ that relates to the angular velocity of that axis, and is (according to [Siciliano and Khatib, 2016]) given by

$$\begin{aligned} \tilde{\omega} &= \omega + \omega_b + \eta_\omega \\ \eta_\omega &\sim N\left(0, \sigma^2_{gyro}\right) \end{aligned} \tag{2.18}$$

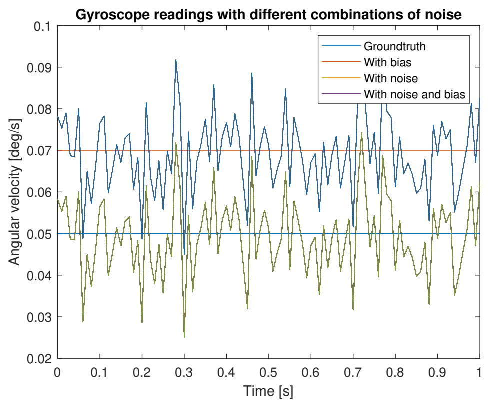
which corresponds to the true angular velocity ω corrupted by the sensor bias ω_b and the sensor noise η_ω, which is modelled as additive, zero-mean Gaussian noise.

In order to have 3DOF we use 3 gyroscopes, one for each orthogonal axis, and we assume no crosstalk between them. The bias is temperature dependent and can vary over time, but is generally modelled as a constant, and may be specified in the manufacturer's datasheet, alongside the sensor variance σ^2_{gyro}.

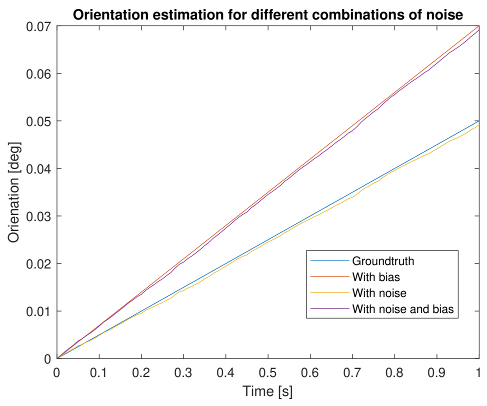
In order to obtain orientation information from angular velocity, we can integrate the angular velocity, as per the motion equations and corresponding Taylor expansion

$$\begin{aligned} \omega(t) &= \frac{\partial}{\partial t}\theta(t) \\ \theta(t + \delta t) &= \theta(t) + \frac{\partial}{\partial t}\theta(t)\delta t + \epsilon \\ \epsilon &\propto O(\delta t^2) \quad . \end{aligned} \tag{2.19}$$

A perfect measurement would produce a perfect estimation of orientation (apart from a constant offset). However, noise makes this more complicated, as shown in Fig. 2.12. Gaussian noise makes the orientation estimation fluctuate along the correct value, which may be acceptable in some situations. However, the bias poses a more complicated problem, as we are constantly integrating a wrong value, even when there is no movement.



(a) comparison of readings with different noises



(b) expected orientation for different noises

Figure 2.12: (a) Effect of the various types of noise on the reading of the sensor and its impact on the estimation of the orientation (b).

Continuing with the accelerometers, a similar reasoning can be applied. We have a measurement $\tilde{a}$ given by

$$\begin{aligned} \tilde{a} &= a + a_b + \eta_a \\ \eta_a &\sim N\left(0, \sigma_{acc}^2\right) \end{aligned} \tag{2.20}$$

dependent on the true value a, sensor bias a_b and noise η_a. The accelerometer deserves special attention, as it does not measure acceleration relative to a coordinate frame as one would expect. Rather, this acceleration is relative to a free fall state, meaning the reading is 0 when the sensor is free falling. When static, it measures the gravitational acceleration pointing upwards (since the accelerometer is accelerating upwards comparing to a free fall).

Again, a sensor for each axis is used, and no crosstalk is assumed. In order to obtain position information from angular velocity, we can integrate the acceleration twice (once for velocity and twice for position), as per the motion equations and corresponding Taylor expansion

$$\begin{aligned} v(t) &= \frac{\partial}{\partial t} x(t) \\ a(t) &= \frac{\partial}{\partial t} v(t) = \frac{\partial^2}{\partial t^2} x(t) \\ x(t+\delta t) &= x(t) + \frac{\partial}{\partial t} x(t)\delta t + \frac{1}{2}\frac{\partial^2}{\partial t^2} x(t)\delta t^2 + \epsilon \\ \epsilon &\propto O(\delta t^3) \end{aligned} \tag{2.21}$$

Similarly, a perfect measurement would produce a perfect estimation of position (apart from a constant offset), but noise and bias render this task difficult. An extra step needs to be taken into account, which is subtracting the force of gravity from the affected axis (or axes, when no single axis is pointing straight down (or up)).

IMUs are very useful in the sense that they provide high speed information regarding both position and orientation (through angular velocity and linear acceleration). However, the noisy measurements, coupled with the random walk associated with the bias, can lead to very incorrect estimations, especially when the system is standing still, and the noise overpowers the correct measurement itself. The calibration process for the IMU is explained in Appendix C.

2.4 Pose Estimation

Pose estimation refers to the problem of trying to estimate the position and orientation (collectively called the pose) of a system. A common approach is to use information provided by a camera to detect and track a set of features, which are used to estimate the motion of the system (ego-motion), as there is a relation between the 3D point in the world and its corresponding 2D projection in camera space (Section 2.1.1), which is also influenced by the movement of the system (Section 2.4.1).

2.4.1 Camera Motion and Motion Field

There is much information to be extracted from a time-varying sequence of images. When a camera moves (or when objects move in front of a camera), there are changes in the captured image, that can be used to estimate the motion of the camera.

Let us assume a generic point P_0 in the world that is projected onto a point P_i in the camera by means of the projection equations presented in Section 2.1.1. The movement of this point P_0 will create a motion on the image plane. We define the motion field corresponding to the moving point as an assignment of a velocity vector to each pixel in the image. However, the motion field cannot be directly measured, as we can only measure the motion of brightness patterns in the image which we refer to optical flow. Fig. 2.13 represents this idea of a point P_0 moving at velocity v_0 in the world, with corresponding projection P_i with velocity v_i (the motion field).

The perspective projection for this simple example is given by

$$\frac{r_i}{f} = \frac{r_0}{r_0 \cdot z} \tag{2.22}$$

which can be used to apply to the velocities

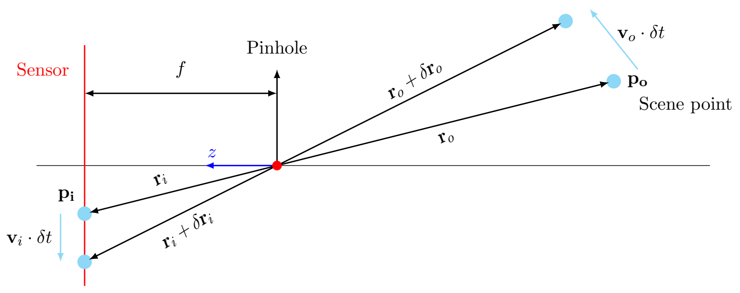

Sensor
f
Pinhole
$\mathbf{r}_o + \delta\mathbf{r}_o$
$\mathbf{v}_o \cdot \delta t$
$\mathbf{p_o}$
Scene point
$\mathbf{r}_o$
z
$\mathbf{r}_i$
$\mathbf{p_i}$
$\mathbf{r}_i + \delta\mathbf{r}_i$
$\mathbf{v}_i \cdot \delta t$

Figure 2.13: Motion field obtained when projecting a moving pointP_0 in the world to the camera plane.

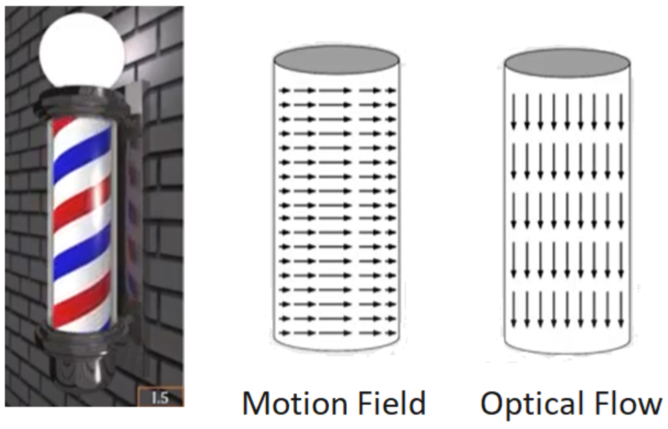


Figure 2.14: Representation of the motion field and optical flow from the barber pole, where there is a mismatch between the two.

$$\begin{aligned} &\text{Scene point velocity:} \quad v_0 = \frac{d\,r_0}{dt} \\ &\text{Image point velocity:} \quad v_i = \frac{d\,r_i}{dt} = f\frac{(r_0 \cdot z)\,v_0 - (v_0 \cdot z)\,r_0}{(r_0 \cdot z)^2} = f\frac{(r_0 \times v_0) \times z}{(r_0 \cdot z)^2} \end{aligned} \tag{2.23}$$

where $\times$ denotes cross product.

The motion field, v_i, is what we would like to have. Unfortunately, we cannot guarantee that this is what is being measured. We are only measuring the motion of brightness patterns in two consecutive frames, called the optical flow, which depends on many factors, such as the distance of the point P_0 to the camera.

The classic example where motion field and optical flow are not the same is the barber pole, Fig. 2.14, where a rotation along the vertical axis results in a mismatch between the motion field and the optical flow.

Consider now the motion field generated exclusively by the movement of the camera (ego-motion), with translational velocity T and angular velocity ω. In relation to the observer, a 3D point $P = (X, Y, Z)^T$ moves, according to [Horn et al., 1986], following

$$\dot{P} = -T - \omega \times P = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = - \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} - \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad . \tag{2.24}$$

Recovering the projection equations presented in Section 2.1.1, in particular

$$\begin{cases} x = f\dfrac{X}{Z} \\ y = f\dfrac{Y}{Z} \end{cases} \Rightarrow \begin{cases} \dot{x} = f\dfrac{Z\dot{X} - X\dot{Z}}{Z^2} \\ \dot{y} = f\dfrac{Z\dot{Y} - Y\dot{Z}}{Z^2} \end{cases} \quad . \tag{2.25}$$

Combining (2.24) and (2.25), we arrive at (check Section B for derivation)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} -T_x + \frac{x}{f}T_z \\ -T_y + \frac{y}{f}T_z \end{bmatrix} + \begin{bmatrix} \omega_x \frac{xy}{f} - \omega_y \left(f + \frac{x^2}{f}\right) + \omega_z y \\ \omega_x \left(f + \frac{y^2}{f}\right) - \omega_y \frac{xy}{f} - \omega_z x \end{bmatrix} \quad . \tag{2.26}$$

Analysing this equation is interesting, and we can see the left part pertains to translation, and the right to rotation. Furthermore, we can also see that translational component depends inversely on depth, and there is scale ambiguity (meaning T and Z can only be recovered up to a scale factor). On the rotation side of things, we see that there is no dependence of depth, and therefore depth estimation is not possible without translation.

2.4.2 Visual Odometry

Visual odometry (first coined and successfully implemented in 2004 by Nister in his landmark paper [Nistér et al., 2004], as stated in [Scaramuzza and Fraundorfer, 2011]) appears as the inverse problem to (2.26), as we want to estimate the translational and angular velocities of our camera, assuming we know how certain specific points in our camera plane (called features), are moving along time.

Odometry itself is the use of data from movement sensors to estimate the change of position and orientation of a system over time. It naturally follows that visual odometry is the use of visual information (in particular cameras) to estimate this displacement.

A typical pipeline for visual odometry is as follows: 1) Acquire image; 2) Undistort image, 3) Detect features (Match features between consecutive frames, and Obtain optical flow field), 4)Estimate camera motion from optical flow (a common approach is the use of Kalman Filter (Section A.2))

Visual odometry is particularly relevant in the context of SLAM, more specifically Visual SLAM (also called vSLAM), whose goal is to use visual information to construct a map of the system's surrounding environment, and to place the system in said map. In this case, the visual component is useful to estimate the motion of the system across time (and also to correct the predicted pose of the system when it passes through a previously mapped zone (loop closure)). Multiple SLAM approaches have been proposed, of which we highlight Fast-SLAM ([Montemerlo et al., 2002]), ORB-SLAM ([Mur-Artal and Tardós, 2017]), EKF-SLAM, and GraphSLAM ([Thrun, 2002]). These approaches rely on conventional cameras to provide features that are used to construct a map and also be used as reference for the estimation of the pose of the system.

2.5 State of the Art

In this section, we briefly present some related works to the problem of pose estimation with event cameras. We also analyse SLAM formulations using conventional cameras as a way to compare between conventional cameras' and event cameras' performance when tackling this challenge.

Many solutions have been proposed in the context of pose estimation with conventional cameras. We choose to highlight ORB-SLAM ([Mur-Artal et al., 2015]) as a reference in the area of SLAM. This is a Visual SLAM implementation, meaning only visual information (and not inertial) is used. The system also allows for stereo and depth camera implementations (with the advantage of being able to obtain the scale of the map). Being a SLAM approach, great emphasis is placed on map building, with corresponding difficulties (but also advantages). In particular, three threads are considered: 1) the tracking thread, that localizes the camera at every frame by matching the detected features; 2) the local mapping thread, that creates the map, and performs a local Bundle

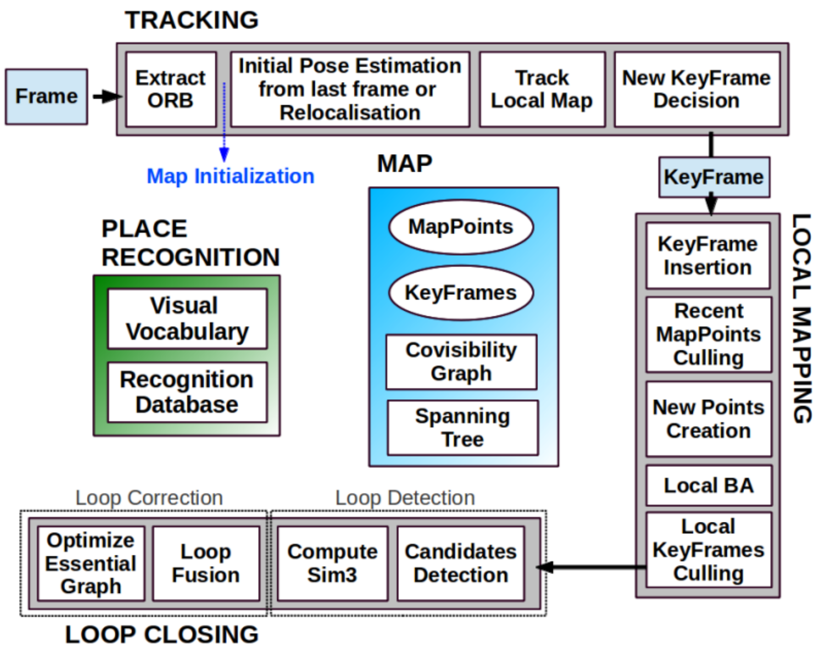


Figure 2.15: Overview of ORB-SLAM architecture, showing the tracking, local mapping, and loop closing threads, from [Mur-Artal et al., 2015].

Adjustment; and 3) the loop closing thread, that detects when the system revisits a previously before seen location, and corrects drift in the system estimation. Fig. 2.15 shows this global description of the system.

ORB features are used in the system, with an associated 256 bit descriptor, as they are faster than SIFT and SURF, which are unable to keep the temporal requirements of the authors. These features are based on FAST corners, with a custom descriptor.

The tracking thread is what is most relevant for comparison with this work. A frame by frame feature matching is performed, and the features being detected are compared against the local map to obtain the pose of the camera. A motion-only bundle adjustment is performed to minimize the error in placing the feature in the correct position (minimizing the reprojection error). If tracking is lost, the system tries to re-localize itself on the map it has built.

The map itself is created by leveraging algorithms such as ICP and local Bundle Adjustment (BA), by combining the most probable position of the camera with corresponding feature location. The initialization uses the relative poses between two scenes to obtain the fundamental matrix and place the features in the starting map.

Assuming a previous map has already been created, ORB-SLAM also allows for a “localization-only” mode, where the system is being placed in the previously generated map, but a new map is not being generated (in effect, only the tracking thread is active).

Even though ORB-SLAM uses a visual odometry approach, it is a reference in terms of performance, and a baseline against with many works compare their approach and performance.

Event cameras are much more recent (to conventional cameras), which means a definitive approach to pose estimation (and SLAM) has yet to be widely accepted, and multiple authors have proposed different approaches to this problem, ranging from conservative formulations, trying to leverage existing and proven methodologies using event cameras ([Zihao Zhu et al., 2017]), to more innovative approaches trying to make the most of what is now

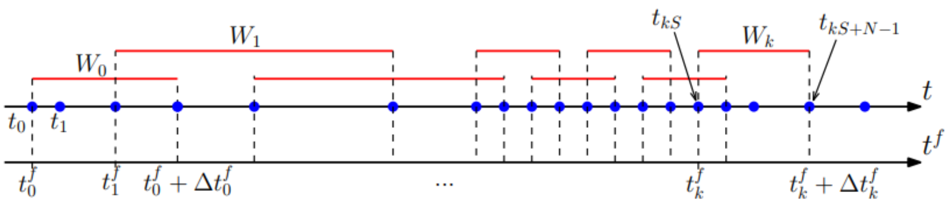


Figure 2.16: Spatiotemporal window proposed, where each event (blue) is grouped into a window W_n, that is then used to obtain features, from [Rebecq et al., 2017].

possible with event cameras ([Vidal et al., 2018]).

Starting with [Rebecq et al., 2017], a visual inertial odometry approach using event cameras is introduced that relies on events and the IMU embedded in the event camera. The estimated state is composed of a rotation matrix $\mathbf{R} \in SO(3)$, a position vector $r \in \mathbb{R}^3$, and a velocity vector $v \in \mathbb{R}^3$, forming the full state vector

$$\begin{bmatrix} \mathbf{R} \\ r \\ v \end{bmatrix} \quad . \tag{2.27}$$

The IMU includes a 3-axis accelerometer and 3-axis gyroscope, and measures linear acceleration and angular velocity, that are modelled according to

$$\begin{aligned} \tilde{\omega}(t) &= \omega(t) + b_g(t) + \eta_g(t) \\ \tilde{a}(t) &= \mathbf{R}(t)\left(a(t) - g\right) + b_a(t) + \eta_a(t) \end{aligned} \tag{2.28}$$

where $\tilde{\omega} \in \mathbb{R}^3$ corresponds to the read value on the gyroscope, $\omega \in \mathbb{R}^3$ is the "real" value, not affected by noise, $b_g \in \mathbb{R}^3$ is the gyroscope bias, and η_g is the gyroscope additive white noise. Similarly, for the accelerometer, $\tilde{a} \in \mathbb{R}^3$ is the value from the sensor, $\mathbf{R} \in SO(3)$ is the rotation matrix representing the current orientation of the camera, $a \in \mathbb{R}^3$ is the "real" acceleration value, $g \in \mathbb{R}^3$ represents the gravity vector, $b_a \in \mathbb{R}^3$ the accelerometer bias, and η_a the sensor noise.

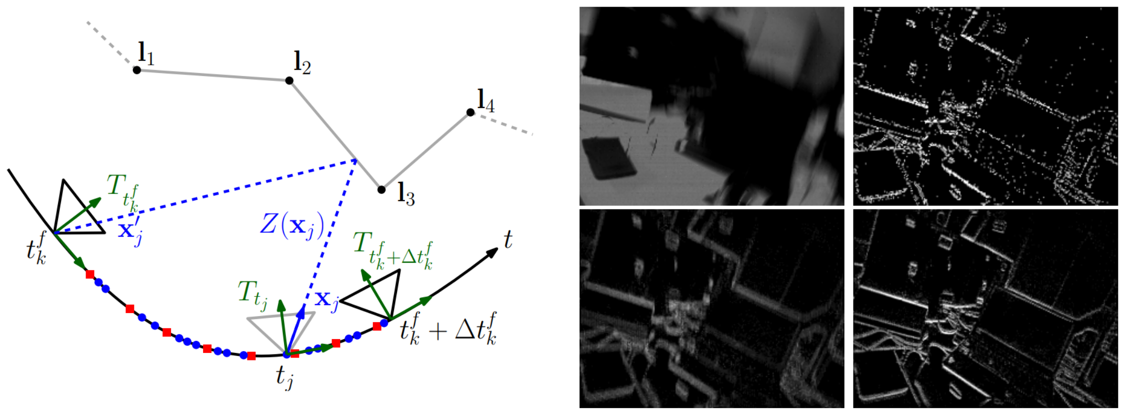
The IMU and its corresponding readings serve as input to the discretised dynamics model

$$\begin{cases} \mathbf{R}\left(t + \Delta t\right) = \mathbf{R}(t)\exp\left(\tilde{\omega} - b_g(t) - \eta_g(t)\Delta t\right) \\ v\left(t + \Delta t\right) = v(t) + g\Delta t + \mathbf{R}(t)\left(\tilde{a}(t) - b_a(t) - \eta_a(t)\right)\Delta t \\ r\left(t + \Delta t\right) = r(t) + v(t) + 1/2 g\Delta t^2 + 1/2\mathbf{R}(t)\left(\tilde{a}(t) - b_a(t) - \eta_a(t)\right)\Delta t^2 \end{cases} \tag{2.29}$$

where exp denotes the exponential map of the Lie Group.

On the vision aspect of this approach, the concept of spatio-temporal windows is introduced, which is a way this group proposes to accumulate events, so that a frame can be obtained for feature extraction (Fig. 2.16).

The novel idea in this approach is how the event frames are generated from events. Instead of simply accumulating them according to $I_k(\mathbf{x} = \sum_{e_j \in W_k} \delta\left(\mathbf{x} - \mathbf{x}_j\right)$ (Fig. 2.17.(b) top right and bottom left) which is noisy and sometimes not reliable, motion-corrected event frames are proposed, that take advantage of the IMU measurement to reproject the event into the camera frame using the linearly interpolated pose T_{t_j} (Fig. 2.17.(a)), which results in a sharper event frame (Fig. 2.17.(b) bottom right), using the expression $I_k(\mathbf{x} = \sum_{e_j \in W_k} \delta\left(\mathbf{x} - \mathbf{x}'_j\right)$, where $\mathbf{x}'_j$ denotes the corrected event position.



(a) (b)

Figure 2.17: (a) shows the motion correction, where inertial measurements (red squares) are used to compute the linearly interpolated pose T_{t_j} to reproject each event (blue dot). (b) top left shows the camera frame, top right shows 3000 events accumulated, and bottom left 30 000 events, both not motion corrected, bottom right shows 30 000 events motion corrected.

The motion corrected event frames are then fed into a FAST corner detector that extracts features. The visual-inertial localization and mapping problem is then formulated as a joint optimization of a cost function that contains weighted reprojection errors e_r and inertial error terms e_s:

$$J := \sum_{k=1}^{K} \sum_{j \in \mathcal{J}(k)} e^{j,k^T} W_r^{j,k} e^{j,k} + \sum_{h=1}^{K-1} e_s^{k^T} W_s^k e_s^k \tag{2.30}$$

where k denotes the frame index, j the landmark index, and the set $\mathcal{J}(k)$ contains the indices of landmarks visible in the $k-th$ frame. Furthermore, $W_r^{j,k}$ is the information matrix of the landmark measurement I_j, and W_s^k is the information matrix of the $k-th$ IMU error. The reprojetcion error e_r^{j,k^T} is given by

$$e_r^{j,k^T} = z^{j,k} - \pi\left(T_{CS}^k T_{SW}^k I^j\right) \tag{2.31}$$

where $z^{j,k}$ is the measured image coordinate of the $j-th$ landmark on the $k-th$ frame. The IMU measurement error is given by comparing the current predicted state based on the previous state and the actual current state.

Moving to the work of [Zihao Zhu et al., 2017], an event-based visual inertial odometry approach for estimating 6-DOF using EKF is proposed, by the name of EVIO (Event-based Visual Inertial Odometry). This approach leverages both the visual information from the event camera, as well as the inertial information from the IMU. The filter estimates a state of the form

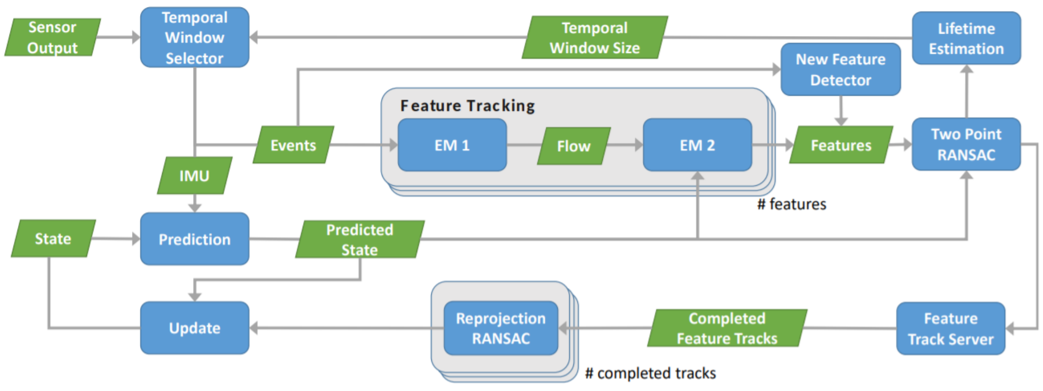


Figure 2.18: Algorithm overview. Data from the camera and the IMU are processed in temporal windows and integrated to propagate the state. Features are tracked using two expectation maximization steps that estimate the optical flow of the features and their alignment with respect to a template. Outliers are removed using RANSAC. From [Zihao Zhu et al., 2017].

$$S := \begin{bmatrix} q \\ b_g \\ v \\ b_a \\ p \end{bmatrix} \tag{2.32}$$

where q represents the orientation of the camera in the global frame by means of a unit quaternion, v represents the velocity of the system, p its position, and b_g and b_a the biases of the gyroscope and accelerometer, respectively. The global overview of the proposed approach is presented in Fig. 2.18.

Features in this approach leverage the fact that events generated from a landmark produce, and therefore, all lie in, the same curve in a spatio-temporal representation. Furthermore, this curve also relates to the optical flow of the feature created in the camera space from said landmark. The spatio-temporal windows to analyse the landmarks are dynamically assigned by measuring the activity (rate of event generation) of that feature.

In order to increase robustness of the features being tracked and fed into the filter, a RANSAC approach is used to remove outliers and correct the estimated position. The features themselves correspond to corners being detected using the FAST technique.

For the propagation itself, the dynamic equations

$$\begin{aligned}
\dot{q}(\tau_k) &= \frac{1}{2}\Omega(\omega_k - \hat{b}_g(\tau_k)q(\tau_k) \\
\dot{p}(\tau_k) &= v(\tau_k) \\
\dot{v}(\tau_k) &= R(q(\tau_k))^T(a_k - \hat{b}_a(\tau_k)) + g \\
\dot{b}_a(\tau_k) &= 0 \\
\dot{b}_g(\tau_k) &= 0
\end{aligned} \tag{2.33}$$

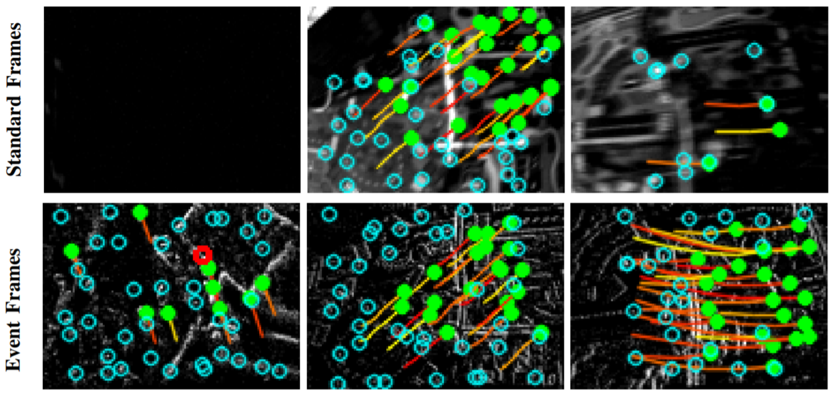


Figure 2.19: Feature detection on frames and event frames, for multiple conditions. Green features have been tracked for some time and are considered good features, whereas blue ones are candidate features, from [Vidal et al., 2018].

are used.

For the update step, the comparison between expected feature position and observed reprojection is performed, by means of

$$h(L, S) := \pi(R(q)(L - p)) \tag{2.34}$$

where L denotes the batch of features, S the estimated state, q the orientation, $R(q)$ the rotation matrix and p the expected feature position.

Lastly, [Vidal et al., 2018] proposes the current best solution for pose estimation, also leveraging frames, events, and the IMU for visual inertial odometry, but with some differences in regards to the approaches presented previously. It can be considered an improvement on the work by [Rebecq et al., 2017], but now using frames as well. It also tackles some of the limitations of [Zihao Zhu et al., 2017], in particular its slow performance (and resulting impossibility to be used real-time).

The main idea can be summarised as follows. Two types of visual information are being used in parallel: frames in the conventional sense, and event frames, created by creating spatio-temporal windows of events that are then accumulated (therefore creating a frame). Features using both types of frames are detected using the FAST corner detector, and tracked using KLT. They are also used to triangulate landmarks in the world.

The spatio-temporal window for event accumulation is the same as the one presented in Fig. 2.16, and a window of 20 000 events was used in this case. Furthermore, the idea for motion-corrected event frames, as well as the filter state, are the same as the ones in [Rebecq et al., 2017].

The optimization function is also the same as [Rebecq et al., 2017], but now using visual information that alternates between features from event frames, and from conventional frames. The use of these two types of visual information increase greatly the performance of the approach, which is also able to run in real time. Fig. 2.19 shows the detection of features using FAST with both frames, and event frames.

Chapter 3

Visual Odometry and Visual Inertial Odometry

If I had only one hour to save the world, I would spend fifty-five minutes defining the problem, and only five minutes finding the solution.

— Albert Einstein (apocryphal)

This chapter explains the proposed and developed methodology for the Visual Inertial Odometry system. It is written in a way to tell the story of how this approach was selected, in particular opposed to other methods. First the system model is explained (Section 3.1). Afterwards inertial odometry systems are presented, and their limitations explained (Section 3.2). Then visual odometry approaches (relying only on vision) and their limitations are presented (Section 3.3). Then our visual inertial odometry approach is introduced, in particular the tools used for the approach, using both vision and inertial information (Section 3.4).

3.1 Problem Formulation

Our system consists of an event camera that contains an IMU sensor embedded. It is important to understand what each sensor is reading and what reference frame each one uses to make sense of the data being fed into, and received from, the system.

Concepts like “left” and “right”, “up” and “down” only make sense when there is some sort of reference to compare it against. Reference frames appear as coordinate system that uniquely represent measurements within that frame, in an absolute manner. Cartesian coordinates are a common and intuitive choice of representation of measurements, where all axis are orthogonal and linearly independent.

By wanting to estimate the pose of our camera, what is implicit in this statement is that we have defined some sort of reference in the world (for example, the starting point of the camera trajectory), and we want to define its current position and rotation (pose) with regard to this initial reference.

Our IMU reports two types of information: angular velocity $\omega = [\omega_x \, \omega_y \, \omega_z]^T \in \mathbb{R}^3$ and linear acceleration $a = [a_x \, a_y \, a_z]^T \in \mathbb{R}^3$. However, this information also has a reference frame implied. As such, the accelerometer

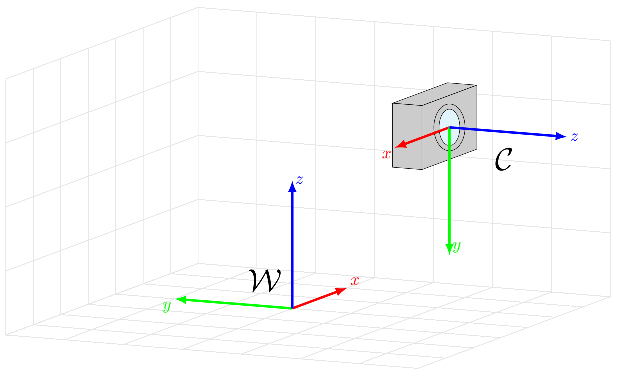


Figure 3.1: Representation of our system model, with relevant coordinate frames of reference described, in particular, world frame $\mathcal{W}$ and camera frame $\mathcal{C}$.

reads movement on the x-axis, for instance, this reading is placed in its own reference frame, and somehow needs to be related to the world frame so that in can be useful in the context of pose estimation (the same applies for gyroscope information, which also relates to the internal reference frame of the IMU).

Luckily, to relate two frames of reference, we only need to rotate their axis so they align, and translate their centres to match, by means of the rigid transformation

$$T(\mathbf{x}) = R\mathbf{x} + \mathbf{t} \tag{3.1}$$

where T denotes the transformation, R the rotation, and $\mathbf{t}$ the translation. The set of rigid body transformations constitutes the Special Euclidean Group (SE), and is better explained in Section A.1. Multiple transformations can be successively performed, so that nested frames of reference can be used (frames references that depend on other frames of references to be described, so that moving the former automatically moves the latter).

The camera also has a reference frame for the image frame. Luckily, this reference frame is aligned with the IMU reference frame. As such, the critical frames of reference are the world frame (on which we are estimating the pose of the camera), and the IMU/camera reference frame. This setup is shown in Fig. 3.1, showing the world frame $\mathcal{W}$, with regards to which we want to estimate the position and orientation of our system, and the camera frame $\mathcal{C}$, with regards to which the sensor readings are produced.

To reiterate, the odometry methods presented in this chapter intend to estimate the position of the camera, at all times, with regards to the initial (base) frame, typically the world frame $\mathcal{W}$.

3.2 Inertial Odometry

Inertial odometry refers to the use of inertial sensors to estimate the current pose of the system. As presented in Section 2.3, the IMU is an inertial sensor that reports on the movement of the system (namely angular velocity and linear acceleration). As hinted on the same section, using Newton's laws of motion, it is possible to relate the pose of the system with the sensor reading.

Being of interest to the ORIENT project, which aims to study the visual and vestibular systems, inspiration was taken from such biological systems, and multiple comparisons with biological systems are made with said systems. So, do humans have some sort of inertial odometry? The answer is yes, by means of the vestibular system.

The vestibular system is not very familiar, but it is responsible to report information on our movement, sense of balance and spatial orientation. It is comprised of two particular structures in our ears: the semi-circular channels, and the otolith organs. We have three semi-circular canals in each ear, orthogonal among themselves, one for each axis, that are capable of sensing rotational movement. We have two otolith organs in each ear, responsible for linear acceleration information. (Funny enough, our vestibular system is sensitive to angular and linear acceleration ([Luo, 2015]), whereas an IMU reports angular **velocity** and linear acceleration.)

The analogy with humans is interesting in this case, even if not explicitly analysed during daily lives. Imagine you are in a car, as a passenger, blindfolded and not moving. There is no visual information to know where you currently are, nor estimation based on steps, touch or any other systems. Nevertheless, whenever the driver makes a turn, you can feel the rotation (even if by means of centripetal force), and whenever the driver stops or accelerates, you can also feel such movements. By knowing your starting point, and continuously counting the turns of the route, time accelerating, being stopped, ..., you can keep more or less keep track of your whereabouts.

This idea is translated to an artificial system using an IMU, by continuously integrating the information from the sensor. The gyroscope registers the turns (in 3D) that the system is performing, or is being subject to, and the accelerometer reports the acceleration of the system (when it was stopped, and when it was moving). This way there is an estimate of the pose of the system at all times, using a simple approach. This inertial odometry obtained from continuous integration of the measurements is called dead reckoning.

However, there are limitations, which can also be interpreted in light of human experience. Imagine you are now in a rollercoaster, still blindfolded. You can still feel your movement in the cart (very aggressively, in fact), but over time (after a few loop-the-loops), you have lost your bearing. After a number of twists and turns, you can no longer know where you are pointing to in regards to your starting point (or at least, not with total confidence). You can still say that you are moving left or right, but you have drifted in your pose somewhere along time.

And this is the most significant limitation of inertial odometry: it is prone to drifting away over time, as there is no correction from other systems. Sensors inherently have noise, and, in particular, the IMU has biases, which means that such a naïve approach is of limited use, has it drifts over time and is, therefore, fallible.

However, inertial odometry allows for very fast updates (IMU usually report information upwards of 100 Hz), which can be needed in some cases, where a very small, very fast movement needs to be estimated.

Nevertheless, we have not opted for such an approach.

3.3 Visual Odometry

As introduced in Section 2.4.2, visual odometry refers to the approach to the pose estimation problem where information from the motion of the camera is obtained based on changes in the image, by tracking the change of certain features in the image.

Returning to the analogy with biological systems and human examples, the visual system is familiar to us, and consists mainly of our eyes. There is a great deal of information we obtain from the world using our eyes, in particular using our **two** eyes. Without any movement, we can observe the environment around us, we can estimate the motion around us, and we can infer our position. Furthermore, and by no means less important, with two eyes we can estimate depth from the mismatch of the images being captured by each eye, as well as the accommodation reflex of the pupil (we change the lens shape and pupil size based on the distance of the object in focus ([Luo, 2015]), which is obvious when a person tries to follow a pen moving in front of their eyes). Just by common sense, vision is critical (though vision can be replaced, as is evident by blind people, but we digress), which is why visual odometry has been chosen in multiple systems.

There are many methods to obtain visual odometry, i.e., obtain information from the motion of the camera based changes in the image, as mentioned in Section 2.4.2. Filtering, in particular Kalman Filtering is such an approach. Another is that of visual SLAM, again, as mentioned in Section 2.4.2.

We chose to present an approach based on the work of [Martins, 2019]. This solution was initially developed in the context of estimating the orientation of an eye, as part of the ORIENT project, and therefore only focused on rotation (as a translating eye in its socket would be a serious medical condition). As such, it is not a true Visual Odometry approach, as only rotation information is provided. For a true system, translation should also be estimated. Regardless, it is interesting to test this approach as event cameras are still recent, and many recent proposed approaches also only estimate angular motion.

In [Martins, 2019], approaches to estimate the orientation of a monocular visual system were proposed, based on the the Orthogonal Procrustes Problem, which is a problem in Linear Algebra that consists of finding the orthogonal matrix Ω that most closely maps matrix A to matrix B, as formulated in

$$R = argmin\|\Omega A - B\|_F, \text{subject to } \Omega^T\Omega = I \tag{3.2}$$

where $\|.\|_F$ represents the Frobenius norm.

This problem is equivalent to finding the nearest orthogonal matrix to a given matrix $M = BA^T$. As such, to find the orthogonal matrix R, the singular value decomposition (SVD) can be used, resulting in

$$M = U\Sigma V^T \tag{3.3}$$

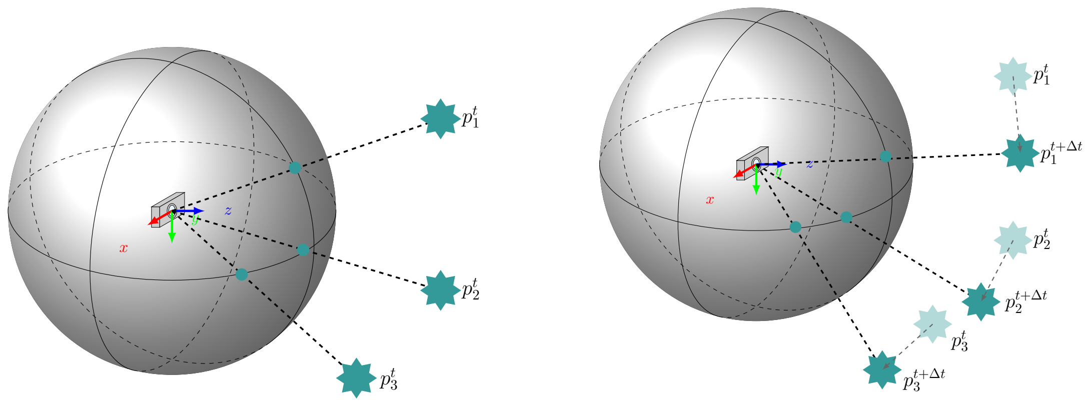
$$R = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \text{sign(det(} \ VU^T \text{))} \end{bmatrix} V^T \quad . \tag{3.4}$$

This solution though SVD was demonstrated to be optimal ([Schönemann, 1966]). In order to apply the Procrustes method to orientation estimation, a set of features are extracted from the images before and after the movement of the camera. Since depth cannot be accurately estimated using a single camera with no translation, the features are projected into a unitary sphere, thus becoming 3D landmarks (shown in Fig. 3.2).

Matrices A and B represent the pointclouds of landmarks detected, and the objective is to find the rotation matrix R (Ω in (3.2)) that maps one pointcloud to the other. The proposed method thus estimates rotation between two frames assuming no (or minimal) translation by using the Orthogonal Procrustes Problem. Assuming this setup, this solution is optimal. Nevertheless, feature detection, matching, non-ideal movement, and other factors complicate this problem.

Filtering Approach for Video Data Expanding on the work by [Martins, 2019], we add support for more than only two frames, in particular a video sequence. The idea is to apply the Orthogonal Procrustes Problem consecutively when a new frame is received, so that rotation over time can be estimated.

In order to minimize the error accumulation, rather than constantly comparing the features of the current frame to the previous frame, we instead compare it to the earliest recorded frame with a minimum number of matched features. When this number is lost, the trackers are reset and the last estimated orientation becomes the reference



(a) projections before movement (b) projections after movement

Figure 3.2: Estimating rotation using Procrustes: (a) shows instant t with corresponding projection of features, and (b) shows instant $t + \Delta t$, with movement of camera and corresponding change of projection of the tracked features.

orientation for future comparisons, which can sometimes be a source of error, as deviations are accumulated over time.

Adaptation to Events The Procrustes' strategy expects a set of 3D landmarks that are provided with each acquired frame. Events do not naturally follow this organized and expectable pattern, so changes are needed.

The proposed method involves using EKLT ([Gehrig et al., 2020]) to detect and track the features. This way we are able to asynchronously detect and track features across time, each with a fixed ID (eliminating the need for a matching step, such as RANSAC in the original implementation).

In order to address the dependence on frames, we accumulate the asynchronous features over a period of time in order to simulate frames being received. Nevertheless, in order to take advantage of the fast nature of frames (which would otherwise be lost waiting for the accumulation of events and features), we keep this integration time very short (around 5 ms, still much faster than the typical 20-35 ms of conventional cameras), or lower.

3.4 Visual Inertial Odometry

In this section we explain our proposed approach for pose estimation using event cameras, leveraging both visual and inertial information provided by event cameras. We start by explaining the work of [Brossard et al., 2017], on which we based ourselves.

3.4.1 Proposed Approach

Our filtering and sensor fusion approach is based on ([Brossard et al., 2017]), which proposes a filter that integrates IMU and visual information in the form of a UKF (Unscented Kalman Filter) (technically a SR-UKF, but this distinction is nor relevant; check Appendix A.2 for more information). This filter introduces several suggestions worth mentioning, such as 1) a Lie group structure for the state space (resulting in a matrix state space, rather than a vector state space), 2) integration of the landmark position in the Lie group, and 3) representation and computation of the uncertainty directly in the Lie group, rather than outside of it, followed by a conversion to the Lie group.

This filter estimates the body pose (position and velocity) and 3D landmark positions, as well as accelerometer's and gyroscope's biases. The first two parameters (pose and landmark positions) are integrated in the Lie group structure, and the latter two (biases) are appended to the state estimation. Though this is not a SLAM implementation, as the whole area is not being mapped (only the local, current region in the vicinity of the body is estimated), the formulation itself is very similar as if it were a SLAM problem (and could be extended if desired).

Our approach, based on the integration of the visual information (processed by EKLT) with inertial information by means of the filter presented, aims to leverage a synergy and complementarity between the two types of information. The global overview of this approach is shown in Fig. 3.3.

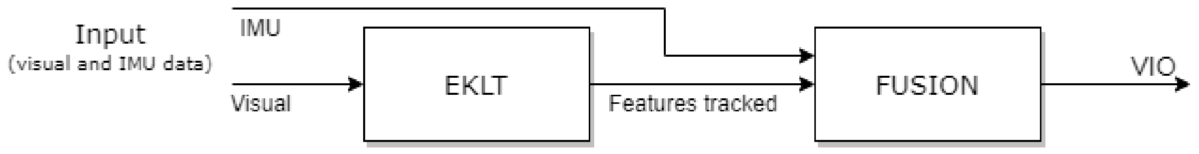


Figure 3.3: First proposed approach, where the feature tracker EKLT is fed into the pose estimator, FUSION, a Lie groups-based Unscented Kalman Filter.

Why use UKF (rather than EKF)? Simple Kalman Filter is not suitable, as the system is non-linear. Between UKF and EKF, UKF has better noise robustness, due to the unscented transform (see Section A.2), allowing for the presence of stronger noise, as it does not need to linearize the system (which incorrectly models stronger noises).

Why use Lie groups? Multiple ways can be chosen to represent the variables we want to estimate, in particular position and rotation of the system, the most common of which being a column vector to represent the state space. This gives some freedom to manipulate and update each one, eventually at the cost of some performance/consistency. The advantages of using Lie groups for state representation are twofold: first, it is a "clean" and efficient representation of these values, and the mathematics for manipulation are all defined beforehand (which involves some work, but only needs to be done once); second, there is a greater numeral consistency, from the inherent representation of the variables, resulting in greater performance. As such, in this filter, Lie groups were used.

3.4.2 Special Note on DVS Cameras

EKLT was developed with DAVIS cameras in mind, as they assume access to frames and events is always available. However, as explained in Section 2.2.1, the DVS240 camera available to us does not allow this functionality, as it can either stream frames, or events, one at a time, but not simultaneously. As such, our initial approach when testing with the DVS240 camera consisted of forcing a switch between these two modes, that was toggled when the camera was still.

The reasoning of this approach is as follows: the tracking of features is performed using events, and frames are used for feature extraction, and template matching. As such, if the camera starts in frame mode, stationary, feature extraction of the initial features is possible. Afterwards, the mode is switched to events, and these features that have been extracted are tracked using events (as is expected by EKLT) and matched against the template provided by the first frame (from which the features were extracted).

Obviously, even with perfect tracking and no loss of features, if the camera moves out of the region where the initial features were detected, there will be nothing to track. In this case, motion is stopped so that the mode can be temporarily switched to frames, to allow for the extraction of new features, that are then tracked with events.

This switching loop continues for as long as we keep the experiment alive.

3.4.3 System and Measurements Model

This section explains the relevant components of the our navigation system system, namely the state, dynamic model and measurements model. As referred, the models are based on the work [Brossard et al., 2017].

State Space The state being estimated by the filter is given by the tuple (χ, b) where χ is defined as

$$\chi = \begin{bmatrix} \mathbf{R} & \mathbf{v} & \mathbf{x} & \mathbf{p}_1 \cdots \mathbf{p}_p \\ 0_{(p+2)\times 3} & & I_{(p+2)\times(p+2)} & \end{bmatrix} \tag{3.5}$$

which incorporates the orientation $\mathbf{R} \in SO(3)$, velocity $\mathbf{v} \in \mathbb{R}^3$ and position $\mathbf{x} \in \mathbb{R}^3$, as well as the 3D positions of the landmarks $\mathbf{p}_1, \ldots, \mathbf{p}_p \in \mathbb{R}^3$. The size of χ is $(3+2+p) \times (3+2+p)$. In addition, one has the bias vector $b \in \mathbb{R}^6$, defined as

$$b = \begin{bmatrix} b_\omega^T & b_a^T \end{bmatrix} \tag{3.6}$$

containing the gyroscope and accelerometer biases b_ω and b_a, which is appended to the state, augmenting it.

Dynamics Model The system can be modelled by

$$\text{body state} \begin{cases} \dot{\mathbf{R}} = \mathbf{R}\left(\omega - b_\omega + n_\omega\right)_\times \\ \dot{\mathbf{v}} = \mathbf{R}\left(a - b_a + n_a\right) - g \\ \dot{\mathbf{x}} = \mathbf{v} \end{cases} \tag{3.7}$$

$$\text{IMU biases} \begin{cases} \dot{b_\omega} = n_{b_\omega} \\ \dot{b_a} = n_{b_a} \end{cases} \tag{3.8}$$

$$\text{landmarks } \dot{\mathbf{p_i}} = 0,\ i = 1, \ldots, p \tag{3.9}$$

where we have access to angular velocity ω and linear acceleration a through the IMU mounted on the system. n represents the various noise, defines as

$$n = \begin{bmatrix} n_\omega^T\ n_a^T\ n_{b_\omega}^T\ n_{b_a}^T \end{bmatrix}^T \sim \mathcal{N}(0, Q) \tag{3.10}$$

The notation $(\omega)_\times$ represents the skew symmetric matrix associated with the cross product with vector $\omega \in \mathbb{R}^3$

Measurement Model Visual information is also fed into the system by means of a calibrated monocular event camera, in order to correct the predicted state of the system. The camera observes and tracks p landmarks through the standard pinhole model and corresponding projection model (Section 2.1.1):

$$y_i = \begin{bmatrix} y_u^i \\ y_v^i \end{bmatrix} + n_y^i \tag{3.11}$$

where y_i is the normalized pixel location of the landmark in the camera frames, and $n_y \sim \mathcal{N}(0, N)$ represents the pixel image noise.

This location is then compared with the expected location of the feature in camera space, obtained by projecting the estimated 3D position of the landmark into camera space through

$$\lambda \begin{bmatrix} x_u^i \\ y_u^i \\ 1 \end{bmatrix} = \Pi \left[\mathbf{R}_C^T \left(\mathbf{R}^T \left(\mathbf{p}_i - x \right) - x_c \right) \right] \tag{3.12}$$

where Π denotes our camera matrix, $\mathbf{R}_C^T$ our initial rotation of the system, $\mathbf{R}^T$ the current estimated rotation, $\mathbf{p}_i$ the $i-th$ landmark 3D estimated position, x the estimated position of the system, and x_c the initial position of the system.

After testing different types of ways to obtain and feed visual data from event cameras into the system with the previous approach (Section 3.3), we found out some empirical ways that produced interesting results. The proposed method relies on EKLT ([Gehrig et al., 2020]) to detect and track the features. This way we are able to asynchronously detect and track features across time, each with a fixed ID.

Since events do not naturally follow this organized and expectable pattern of producing visual information at a constant interval (which can be both advantageous and disadvantageous depending on the situation), changes are needed to provide a batch of features to the measurement model.

The proposed approach is that of accumulation of the asynchronous features over a period of time, in order to simulate frames being received. We call these accumulation of event features over time *pseudo-frames*, and are usually of 20 ms or less, to take advantage of the speed of events.

Three strategies are proposed for the creation of the *pseudo-frames*:

- **Fixed interval integration** Features are accumulated over a fixed period of time before being fed into the system. This period of time is defined at the beginning as is configurable by the user. Typical values were under 5 ms.
- **Fixed number of features update** Features are accumulated until a batch with a predetermined number of features is achieved.
- **Hybrid approach** Features are accumulated until a fixed period of time has passed, or until a predetermined batch of features is achieved (whichever comes first), and is a combination of both previous suggestions.

Though the hybrid approach should perform better, we found that a fixed interval integration was much more easily manageable (as it translates quite naturally to a conventional camera producing features, albeit at a much faster rate), we used the fixed interval integration when validating the approach.

Lie Group The system dynamics form a Lie group, as proposed by Brossard et al in [Brossard et al., 2017] and previous publications,

$$SE_{2+p}(3) = \{\chi\} \tag{3.13}$$

where we are denoting the group just by its set, instead of writing the two tuple $(\{\chi\}, *)$ including the operation $*$, in order to simplify the writing.

The group operation $*$ can be detailed as: Let the system be at state χ_1 then a small change χ_2 is applied and the system changes to the state $\chi_3 = \chi_2 * \chi_1$. Note this $*$ respects a differential equation, the system dynamics (3.7), and therefore happens a reduction of options for χ_2 in an infinitesimal time increment. However, as required by a Lie group structure, if χ_2 exists, then there exists also its inverse, meaning an inverse motion. Note also that while it was described $\chi_2 * \chi_1$, there is also the option to describe $\chi_1 * \chi_2$, which means choosing different reference frames and interpretation of the effect of χ_2.

Observations about the Lie Group The Lie group $SE_{2+p}(3)$ can be seen as an extension to $SE(3)$, which includes the velocity of the state, as well as the landmarks' 3D position. The Lie group is relevant in the context of the state propagation, where sensor readings are mapped from the algebra (as they are contained in the tangent space) to the group, and the $*$ operator is applied to obtain the new estimation, as well as uncertainty propagation, where a similar reasoning can be used to propagate sigma points, but using the covariance values rather than sensor readings. The operator $*$ also plays its role in the context of the state update, when we integrate readings from the observation model to correct the current estimate, stored in χ (refer to Appendix A for simple explanation of Lie groups and UKF).

The choice of a matrix χ to represent the state is not common in Kalman Filters, but there are some advantages to this representation. One such reason is that the use of a Lie group that encompasses almost all the variables being estimated allows for a "cleaner" and more robust mathematical formulation, without the need to convert between different types of representation (of which we highlight rotations, that may be represented by Euler angles, rotation matrices, and quaternions, among others), thus increasing numerical consistency (a problem that may arise from the usage of SR-UKF (Appendix A.2)), and increasing accuracy in estimation by avoiding such conversions between representation ([Asl et al., 2019], [Brossard et al., 2017]).

Another reason is legibility, as such a structure, though containing redundant information (of which we highlight rotations, that are represented as a $SO(3)$ Lie group and contained within the $SE_{2+p}(3)$ Lie group, and its skew symmetric representation that includes multiple entries for each axis of rotation), is easier to read and interpret than others, for instance a vector-column containing quaternions, without losing coherence, as may happen with an Euler angles representation and its inherent gimbal-lock problem.

3.4.4 State Observation based on an UKF

Considering the sensors in Figure 3.3, the system state can be observed by designing a filter based on the model. In this section is detailed a state observation filter expanding on the works of [Barrau and Bonnabel, 2015], which introduced a Lie group based EKF, following the suggestions of [Bonnabel, 2012], that showed the similarities between the SLAM problem and the group $SE_{1+p}(3)$. [Loianno et al., 2016] proposed a UKF implementation on $SE(3)$. Lastly, [Brossard et al., 2017] proposed the inclusion of the landmarks into the group itself, creating a group $SE_{2+p}(3)$, coupled with a UKF implementation. Furthermore, two techniques of propagating the uncertainty were suggested, of which we kept the right uncertainty from their proposed Right-UKF-LG.

As previously referred, the system forms a Lie group (3.13). The objective here is to take advantage of the Lie algebra, associated to the Lie group, to create a system observer as a filter structure. This involves linearization, as provided by the Lie algebra, and discretization, as indicated in (3.14) that represents the continuous system as a discrete system. Our dynamical system evolves over time according to

$$\chi_{n+1} = f\left(\chi_n, u_n, w_n\right) \tag{3.14}$$

where χ_n corresponds to the proposed Lie group (3.5) at time instant n, $u_n = \left[\omega_n^T \, a_n^T\right]^T$ is a known input variable with gyroscope and accelerometer readings, and $w_n \sim \mathcal{N}(0, \mathbf{Q}_n)$ is white Gaussian noise. f corresponds to the discretized dynamics model implemented on Lie groups.

Furthermore, the system has associated a discrete measurement of the form

$$y_n = h\left(\chi_n, v_n\right) \tag{3.15}$$

where $v_n \sim \mathcal{N}(0, \mathbf{R}_n)$ is white Gaussian noise.

Uncertainty on Lie Groups The usage of Lie groups to represent part of the state improves accuracy and numeral consistency, but comes at the cost of a more complex representation of noise. Since our state is not a vector space, the usual approach of additive noise is not possible. Following [Barfoot and Furgale, 2014], the probability distribution $\chi \sim \mathcal{N}_{\mathcal{R}}\left(\bar{\chi}, \mathbf{P}\right)$ is defined by mapping the uncertainty ξ to our state by means of the exponential map

$$\chi = \text{exp}(\xi)\bar{\chi},\ \chi \sim \mathcal{N}(0, \mathbf{P}) \quad . \tag{3.16}$$

The uncertainty $\xi = \left[\xi_R^T \, \xi_v^T \, \xi_x^T \, \xi_{p_1}^T \cdots \xi_{p_p}^T\right]^T$ is mapped to the Lie algebra through the transformation $\xi \mapsto \xi^{\wedge}$ defined as

$$\xi^{\wedge} = \begin{bmatrix} (\xi_R)_\times \, \xi_v \, \xi_x \, \xi_{p_1} \cdots \xi_{p_p} \\ 0_{2+p \times 5+p} \end{bmatrix} \quad . \tag{3.17}$$

Time Discretization In order to implement Eqs. (3.7) to (3.9), a simple discretization using the Euler method is used, with the exception of rotation. Considering a small time step Δt, we have

$$\begin{aligned}
\mathbf{R}\left(t + \Delta t\right) &= \mathbf{R}(t)\text{exp}\left[\left(\omega(t) - b_\omega(t)\right)\Delta t + \text{Cov}(n_\omega)^{1/2} g\sqrt{\Delta t}\right]_\times \\
\mathbf{v}\left(t + \Delta t\right) &= \mathbf{v}(t) + \left(\mathbf{R}(t)\left(a(t) - b_a(t)\right) - g\right)\Delta t \\
\mathbf{x}\left(t + \Delta t\right) &= \mathbf{x}(t) + \mathbf{v}(t)\Delta t \\
b_\omega\left(t + \Delta t\right) &= b_\omega(t) \\
b_a\left(t + \Delta t\right) &= b_a(t) \\
\mathbf{p}_i\left(t + \Delta t\right) &= \mathbf{p}_i\left(t\right)
\end{aligned} \tag{3.18}$$

Predict and Update Implementation The various components of the system are described by

$$\text{state} \begin{cases} \chi_n = \text{exp}(\xi)\bar{\chi}_n \\ b_n = \bar{b}_n + \tilde{b} \end{cases}, \begin{bmatrix} \xi \\ \tilde{b} \end{bmatrix} \sim \mathcal{N}(0, \mathbf{P}_n) \tag{3.19}$$

$$\text{dynamics } \left\{\chi_n, b_n = f\left(\chi_{n-1}, u_n - b_{n-1}, n_n\right)\right. \tag{3.20}$$

$$\text{observations} \begin{cases} \mathbf{Y}_n = \left[y_1^T \cdots y_p^T\right]^T := \mathbf{Y}\left(\chi_n, w_n\right) \\ y_i \quad \text{given by (3.12)}, i = 1, \cdots, p \end{cases} \tag{3.21}$$

where $\left(\bar{\chi}, \bar{b}_n\right)$ represents the mean estimate of the state at time n, $\mathbf{P}_n \in \mathbb{R}^{(15+3p)\times(15+3p)}$ is the covariance matrix that defines uncertainties $\left(\xi, \tilde{b}\right)$, and the vector $\mathbf{Y}_n$ contains the observations of the p landmarks with associated Gaussian noise $w_n \sim \mathcal{N}(0, W)$.

These components are implemented into an SR-UKF-like filter, with the usual steps of propagation (based on the motion model with input from the accelerometer and the gyroscope) and update (based on the observation model and the visual information).

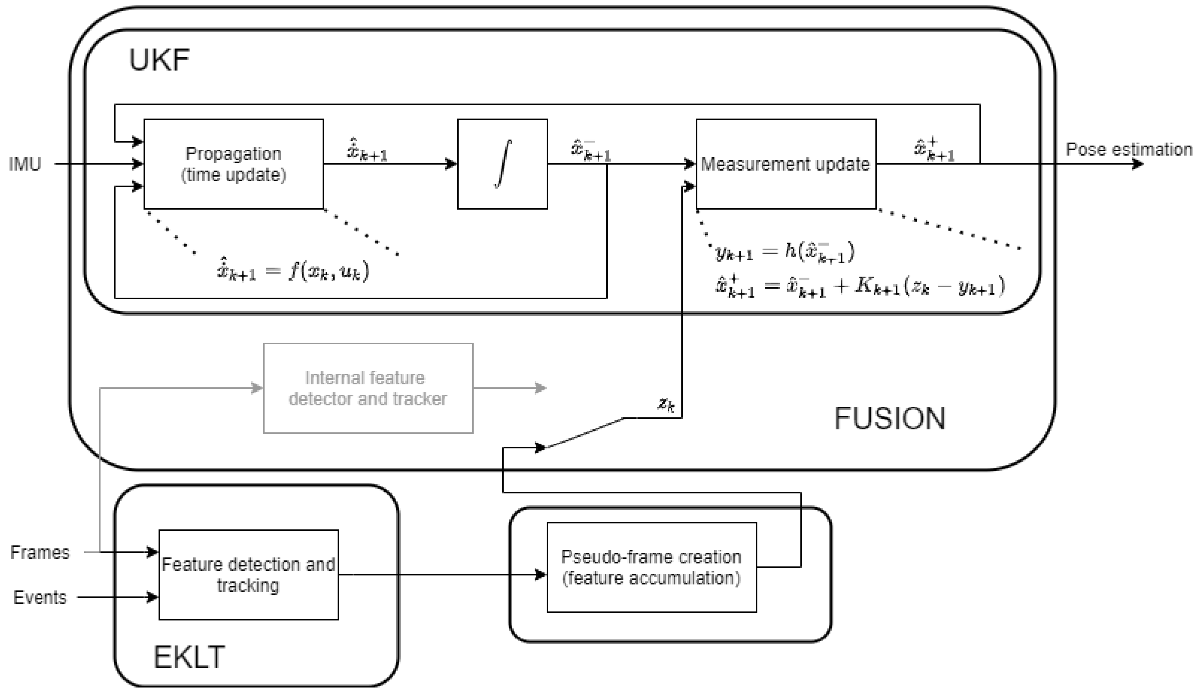

UKF
IMU
Propagation (time update)
$\hat{x}_{k+1}$
$\int$
$\hat{x}^-_{k+1}$
Measurement update
$\hat{x}^+_{k+1}$
Pose estimation
$\hat{x}_{k+1} = f(x_k, u_k)$
$y_{k+1} = h(\hat{x}^-_{k+1})$
$\hat{x}^+_{k+1} = \hat{x}^-_{k+1} + K_{k+1}(z_k - y_{k+1})$
Internal feature detector and tracker
z_k
FUSION
Frames
Events
Feature detection and tracking
EKLT
Pseudo-frame creation (feature accumulation)

Figure 3.4: Proposed integration of visual data, frames and events, into the UKF Visual Inertial filter. We replace the internal feature extractor and tracker with our approach based on EKLT and an accumulation of features, that are then fed into the UKF.

Use Imaging Events A more detailed block diagram of the implementation is shown in Fig. 3.4. In this sketch, $\mathbf{x}$ represents the state, composed of the presented tuple (χ, b). $(\hat{x})$ represents the estimate, $(\hat{x})^-$ is the estimate before measurement update, and $(\hat{x})^+$ represents the same estimate after update. z represents the observations (features) fed into the measurement update.

Filter Initialisation The initialisation of the filter should be carefully considered. The "ideal" initialisation would be to provide the correct values of all variables in the state, i.e., set the correct values of pose and the location of the landmarks, in particular. However, not only is such an approach not always possible or realistic (testing on a new, uncontrolled environment for the first time), it also defeats the purpose of a system that is placed in an unknown environment and must be robust enough to eventually converge to the correct values.

This problem is not exclusive to our approach, and some suggestions have been made elsewhere. Taking [Qin et al., 2018], for example, a suggestion where a dataset is captured, and a first pass on the first moments of the dataset is performed, so that the first few values for the landmark location can be estimated (as multiple points of view allow for triangulation of features into 3D landmarks). After these values are obtained, the system starts again, initialised with them.

However, this solution only works offline, i.e., the system first captures a trajectory, and estimation is performed afterwards. Though perfectly legitimate, we preferred to strive for an approach where the system is able to run online, meaning it can estimate its location at the same time it is exploring the world, without the need for the first initialisation pass.

For the initialisation of the position, orientation, and velocity of the system, no prior information is given, and the filter starts with all values at 0. From our perspective, not only is this approach fair in the sense that the filter must be robust enough to be able to survive the first instants and quickly obtain these values, it also allows for quick testing in different environments, as no prior estimation is needed. Furthermore, this implementation assumes the starting pose to be the *base frame*, meaning all future pose estimations are given in relation to this frame, which we consider to be aligned with the world frame. If there are other components in the system, and we know this not to be true, a simple rigid motion transformation for a coordinate change can be performed. Also, if some information of the initial state of the filter is available, it can also be used for the initialisation, but it is not needed.

The landmarks, however, are a different story, and two distinctions for the initialisation are made: initialisation for the start of the system, and for the replacement of features. For the former, we place landmarks in the world by following the projection line of the corresponding features, which allows for estimation of the X and Y coordinates, but depth is trickier, as a single point of view is not enough for depth estimation (and for that matter, neither is rotation). With this regard, we assign a distance value d that reflects the average distance of the landmarks in the world, to have a notion of scale. As such, every landmark in the world follows $|X, Y, Z| = d$ (which in practice means they are all in the same sphere of radius d). Another approach that was used for planar scenes was to consider the same Z for all features (they are all on the same plane at distance Z from the camera). Regardless, since the depth value is not trustworthy, a high variance is assigned to the start, which decreases as the system evolves.

For the latter (feature replacement), since information of past states of the filter are available (in particular, we keep the previous sightings of every feature), we can introduce a new landmark by triangulating the past sightings into the world, thus creating a much more reliable 3D position of the landmark that is introduced in the filter state.

Chapter 4

Closed Loop Integration of Sensor and Pose Filter on Event Cameras

"Though this be madness, yet there is method in't." (There is method to his madness.)

— William Shakespeare, Hamlet

This chapter explains our second proposed approach to the pose estimation problem using event cameras, and (as designed) is only applicable to event cameras (though the idea can be converted to conventional cameras). It consists of creating a sort of "closed loop"[1] between the filter, estimating the pose, and the tracker, tracking and providing visual features.

Two problems identified when testing the previous approach were as follows: the number of features is limited; and sometimes features are lost, only to be found a few moments later, but with a different ID (which is not necessarily bad, but then the filter treats this feature as a new one, and all previous sightings are discarded). The first problem is of difficult resolution without major changes in the approach, as it is based on corner detection, which are common in images, but still limited. The second problem, however, implies improving the tracking of features so that they are kept alive for longer. As such, we set out to improve EKLT tracking performance.

4.1 Motivation for Improved Approach

Our proposed approach is to develop a system that can better track the features in EKLT, as these are corners, which are distinct, but not particularly abundant in a scene, and therefore EKLT extracts less features than approaches such as SIFT or SURF features. Therefore, keeping as many features alive for as long as possible becomes paramount in our system.

Revisiting EKLT (Section 2.2.2.3), at first glance it may seem like a simple implementation of KLT, where the matching patches are obtained from frames and events (as opposed to the normal strategy of both patches coming from frames), and, to a certain extent, this is true. But there is more to be said about the generation of these

[1]Not to be confused with loop closure, as is common in SLAM formulations, but more like a control theory approach of open vs closed loop control.

templates.

On the event side of things, other methods rather than simple accumulation can be used (such as motion corrected frames proposed in [Rebecq et al., 2017]), or some sort of filtering can be used to reduce outliers or noisy events. Nevertheless, from our experience, the patch region are so small (usually 20 pixels wide), and the integration time so short (usually under 5 ms, and generally under 10 ms), that we thought our efforts were better spent elsewhere.

The frame side of things is a different story, however. Looking back at Fig. 2.10, we can see that the x-wise and y-wise image gradients are generated, and (after being subject to a warp) are merged by means of a dot product with the flow angle. This parameter of the flow angle v may look innocent, but it is critical in the generation of the template, and has proven to be one of the main reasons for tracking loss.

Its function is not obvious or intuitive, but it can be interpreted as a weight in the linear combination of the x-wise and y-wise gradients of the image. If the camera is moving horizontally, then the accumulation of events is mainly on the horizontal direction, and v reflects this by placing more importance on the x-wise derivative. Movements in other directions have respective flow angle values that reflect this movement.

Furthermore, the flow angle v also allows to infer the direction of movement, as moving left to right produces a different polarity of events to a movement from right to left. The template obtained from frames needs to take this into account to be able to be compared against the event template. As such, it is now clearer that this parameter is of paramount importance in the tracker.

Lastly, another parameter to be optimized is the initial location of the feature, which corresponds to the expected position of the feature, and is fed into the optimizer as a starting value. This parameter is also important, but since features are updated very frequently (sometimes around 1 ms, in very fast paced scenes) it is not as critical as the flow angle (though it also plays an important role, in particular when a feature is lost (Section 4.4)).

With this in mind, we propose an approach where the current estimated pose is fed back into EKLT to help with the tracking of features. This approach creates a sort of "closed loop", where position from FUSION is fed into EKLT, which then provides information for FUSION, as shown in Fig. 4.1.

To the best of our knowledge, though not novel, this is an uncommon and innovative approach where there is some sort of additional processing at the sensor level, with information external to said sensor.

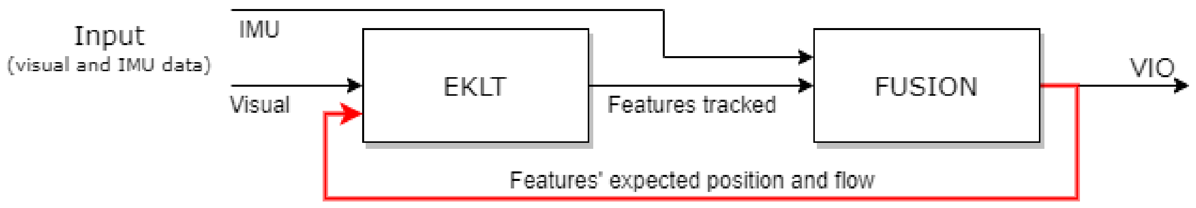


Figure 4.1: Global view of our proposed closed loop integration suggestion, where the pose estimation is fed back into EKLT to help with feature detection.

This approach also tries to turn EKLT into a more robust alternative to DVS cameras, that can only capture either frames or events at any given time, and take some time when switching between these two modes (see Appendix C, where an experiment is performed and this delay is analysed).

Therefore, when features are inevitably lost, there is a period where no new information is provided, which reduces the contribution of the visual component. Therefore, by improving the tracking of features, this approach can also be beneficial to DVS cameras, as features are lost less, and their need to be replaced (and, therefore, of full frames), is reduced.

Ideally, with perfect matching, only the initial frame would be needed. However, obviously, this is never the case, so a switch to frames (and then to events) is necessary to ensure tracking for DVS cameras.

4.2 Ego Motion and Optical Flow

In Section 2.4.1, it is already derived how the movement of the features being captured by the camera are influenced by its motion, in particular (according to [Heeger and Jepson, 1992]), following

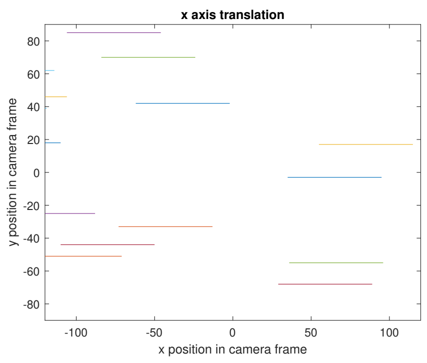
$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} -T_x + \frac{x}{f} T_z \\ -T_y + \frac{y}{f} T_z \end{bmatrix} + \begin{bmatrix} \omega_x \frac{xy}{f} - \omega_y \left(f + \frac{x^2}{f} \right) + \omega_z y \\ \omega_x \left(f + \frac{y^2}{f} \right) - \omega_y \frac{xy}{f} - \omega_z x \end{bmatrix} \tag{4.1}$$

where $\dot{x}$ and $\dot{y}$ represent the flow in x and y axes, respectively, x and y represent the feature in the image frame, f the focal length of the camera, $T_{(.)}$ the translations of the camera, and $\omega_{(.)}$ the rotation of the camera.

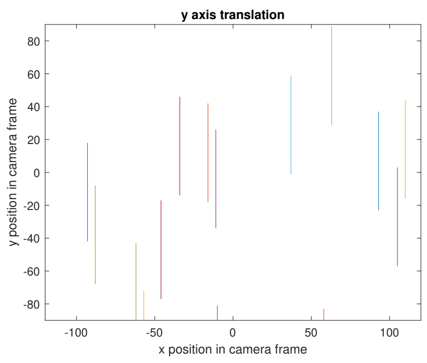
Let us now explore (4.1) in more detail. In particular, let us analyse the influence of each motion in the evolution of the movement of the features. The equation has been grouped in a translation component on the left, and a rotation component on the right. In total, there are 6 basic types of movement possible, corresponding to an isolated rotation or translation in the x, y, and z axes.

Exciting a single axis at a time produces the patterns presented in Fig. 4.2, which show the position of each feature over time, accumulated on a single frame. Notice the similarities in pattern between rotation in the y axis and translation in the x axis, both of which produce mostly horizontal patterns, as well as rotation in the x axis and translation in the y axis, both of which produce mostly vertical patterns. These similarities also speak to the limitations of a purely visual odometry approach, as inertial measurement can help disambiguate between such motions. Only z axis rotation and z axis translation produce more distinctive patterns, the former producing concentric ellipses, and the latter producing lines moving into or away from the centre of the image (point of expansion), similar to the “warp speed” in space movies.

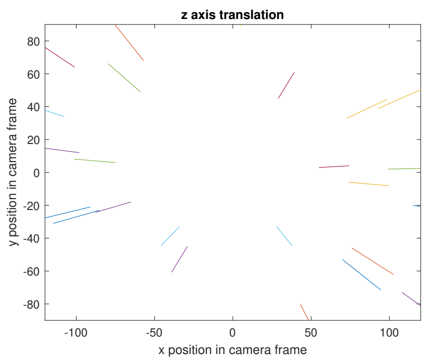
Since we can predict the evolution of the position of the feature over time, it is possible to estimate the flow angle. However, it is very important to take into account that these predictions work for the immediate proximity of the feature, i.e., this assumption only works for small timestep. If the timestep is bigger, the predicted movement will not match the real position of the feature (see Fig. 4.2, where features initiating at the same position eventually drift away and up up at different positions).



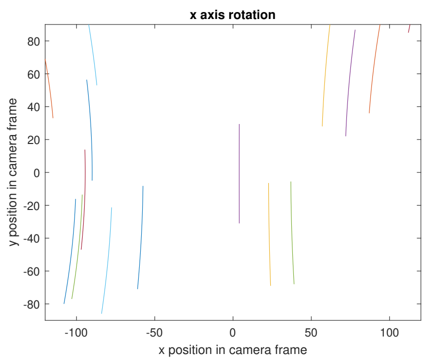
(a) x translation



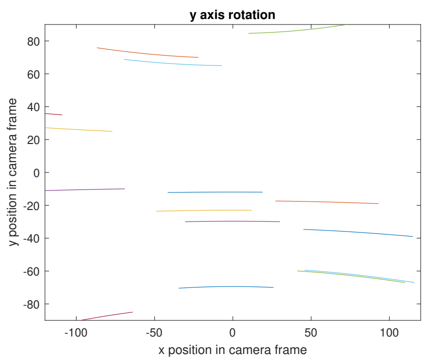
(b) y translation



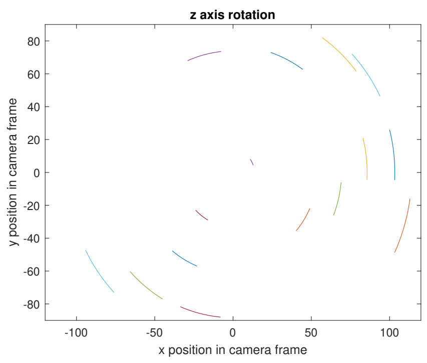
(c) z translation



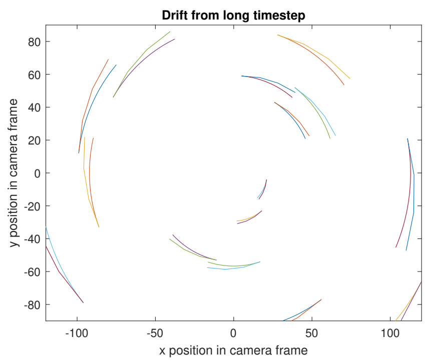
(d) x rotation



(e) y rotation



(f) z rotation



(g) y rotation

Figure 4.2: Evolution of features' position over time due to ego-motion. (a), (b) and (c) show translations in x, y, and z axis, respectively, and (d), (e), and (f) show rotations in x, y, and z axis, respectively. Notice the similarities between (a) and (e), and (b) and (d). (g) shows drift in expected feature position from bigger timestep.

4.3 Features Tracking complemented by the Pose Filter State

Our objective is to improve the feature tracking from EKLT by feeding it information from the current pose estimation, which, in turn, will benefit from a greater number of features.

As already explained, the most critical (or, at least, the component that contributes most from loss of features) is the generation of the template to match from frames, which depends on two main components: the initial position location and the flow angle. Their importance (and relative importance) have already been mentioned (Section 4.1). We hypothesise the pose estimator can (either directly or indirectly) help the tracker with regards to these two components.

Starting with the flow angle, we propose the use of (4.1) to determine it by means of

$$v = \text{atan2}\left(\frac{\dot{y}}{\dot{x}}\right) \tag{4.2}$$

where $\dot{x}$ and $\dot{y}$ are given by (4.1). We reiterate the importance of a small timestep, and synchronisation between the current estimate and EKLT, as disparities become more detrimental than beneficial. To tackle this problem, all timesteps are kept to a minimum, and are usually of about 1 ms. This ensures the assumptions for (4.1) are valid (according to our testing).

In terms of the estimations being used for motion, the angular velocity comes directly from the last measurement of the gyroscope (or some sort of mean or median of the last measurements, if readings are too contaminated by noise). The linear velocity, on the other hand, comes from the state, that estimates the filter velocity (along with system rotation and position, landmark position, and sensor bias). However, it is important to remember the relevant frames of reference (see Fig. 3.1). The velocity being estimated is in the world frame $\mathcal{W}$, meaning it needs to be first converted to the correct frame of reference (the camera's) before being applied to (4.1). Luckily, since the rotation of the camera $\mathbf{R}$ is one of the variables being estimated in the state, the conversion of velocity to the camera frame is given by

$$v_{\mathcal{C}} = \mathbf{R}^T v_{\mathcal{W}} \tag{4.3}$$

where $v_{\mathcal{C}}$ and $v_{\mathcal{W}}$ denote the velocity v being referenced in the world frame $\mathcal{W}$, and the camera frame $\mathcal{C}$.

Moving on to the initial feature position, our proposed filter structure keeps the estimated 3D position of landmarks in the state, which can be projected into camera space to obtain the estimated position of the features by means of the projection equation

$$\lambda \begin{bmatrix} x_u^i \\ y_u^i \\ 1 \end{bmatrix} = \Pi\left[\mathbf{R}_C^T\left(\mathbf{R}^T\left(\mathbf{p}_i - x\right) - x_c\right)\right] \tag{4.4}$$

where Π denotes our camera matrix, $\mathbf{R}_C^T$ our initial rotation of the system, $\mathbf{R}^T$ the current estimated rotation, $\mathbf{p}_i$ the $i-th$ landmark 3D estimated position, x the estimated position of the system, and x_c the initial position of the system.

This way, the tracker benefits from having an additional information of the features being tracked by adding the depth factor.

In effect, by "helping" the tracker with the starting values, what is being done is placing the initial estimate inside the region of convergence, and closer to the global minimum, as the rest of the matching is still performed by the optimizer, that tries to match both templates, and estimate the current position of the feature (and its flow) in the process.

The analysis of this region would be interesting, i.e., to say that convergence is guaranteed whenever the initial flow angle is less than 5 degrees, or when the initial position is not further than 3 pixels. In fact, it would be perfect, as there would be a component of predictability in the tracker. However, such an analysis is not trivial, as there are many factors that come into play. To name a few, the speed and overall motion of the scene are critical and produce different scenarios.

A slowly moving, poorly distinctive feature produces less events (as the changes in brightness are fewer), and therefore take longer to create a patch, which means a greater displacement is produced, and without outside help of the closed loop the initial position is farther away from the minimum, perhaps outside the region of convergence. Not only this, but the patch that is created itself is usually not as distinctive, which result in suboptimal solutions to the flow estimation that result in suboptimal patches for comparison, and a poorer tracking of position overall.

From our experiments, the importance of the flow angle is much greater than the initial feature location. This is because the neighbourhood of the feature is really small (the displacement between initial location and final location is typically around 2-3 pixels diagonally), whereas flow angle could have deviated significantly from the last optimization (imagine a rotation in the z axis of the camera, where features on the borders of the camera move faster than those on the inside), and influences the next matching negatively.

4.4 Set of Backup Features

The representation and management of features and landmarks is nothing new, and is a point where much effort is placed. Referring back to ORB-SLAM ([Mur-Artal et al., 2015]), for instance, the local map being generated is constantly being updated and corrected by means of local bundle adjustment, so that when new features are detected, they can be compared against the local map, which consists of a 3D point cloud (with additional parameters that simplify matching but are not particularly relevant in this case).

Based on this idea of using the map created over time to help with localization, we try to take advantage of the location of features over time to help with localization.

The implementation of the proposed closed loop approach also has other benefits. Tracking features in EKLT is costly (computation-wise). As such, when features are lost (either because their tracking quality decreasing under a certain threshold, or because they move outside the FOV of the camera), they are dropped.

In our case, the pose estimator, on the other hand, is capable of storing features and landmarks over time (in effect, creating a sort of map, as per a SLAM formulation) and keeping these lost and discarded features in a sort of *zombie* or *dormant* state.

Since we can project their predicted location onto camera space by means of (4.4), it is possible to awaken these features when they enter the FOV again, for example. This means that these features (that would eventually be detected again, but would be given a new ID, which would not benefit from using past sightings of these features), are able to be reidentified as used in the filter with the same ID.

This method also allows for bigger jumps in feature tracking, as the initial feature position can be set to a place that is far away from the previous estimated position (imagine a situation where this landmark becomes occluded, and therefore disappears, but is kept in memory by the pose estimator; when the landmark is no longer occluded, since the pose estimation kept running, the expected position based on current pose can be used).

Internally, this structure corresponds to a table that keeps track of all the sightings of a specific feature across time (based on its ID), in particular its x and y position in camera space, as well as the estimated camera pose (position and rotation) at that instant. Through the multiple points of view, associated with the feature position and camera pose, it is possible to triangulate the position of the landmark in 3D space, in the world frame (using epipolar geometry constraint).

Since the 3D position is being estimated, it is possible to project it into the camera frame at current time, and feed it into EKLT when it re-enters the camera FOV. The algorithm of this proposed idea is shown in Algorithm 1.

Algorithm 1: Storing of backup features

```
Result: Internal representation of landmarks (that can be projected into camera space and fed to EKLT)
while features are received by FUSION do
    store feature ID, position, and camera estimated position;
    triangulate 3D landmark position;
    if landmark projection (feature) is in camera space then
        inform EKLT of feature expected location;
    else
        continue;
    end
end
```

An interesting side effect of such an approach is that it is possible to rank features based on how distinct and/or observable they are, as features that are detected more, have more entries in the table, and therefore are probably the ones we want to use, as they are more robust.

4.5 Implementation

The more detailed block diagram of the proposed approach is shown in Fig. 4.3. It is an extension of the block diagram presented in the first approach, with the inclusion of the new features being proposed. In particular, the motion estimation that is being performed in the filter is being feeds back into EKLT, and helps better keep track of the features by leveraging the predictable ego-motion effect on features, as well as the creation of backup features, that also help keep track of the features, as well as reidentify them later.

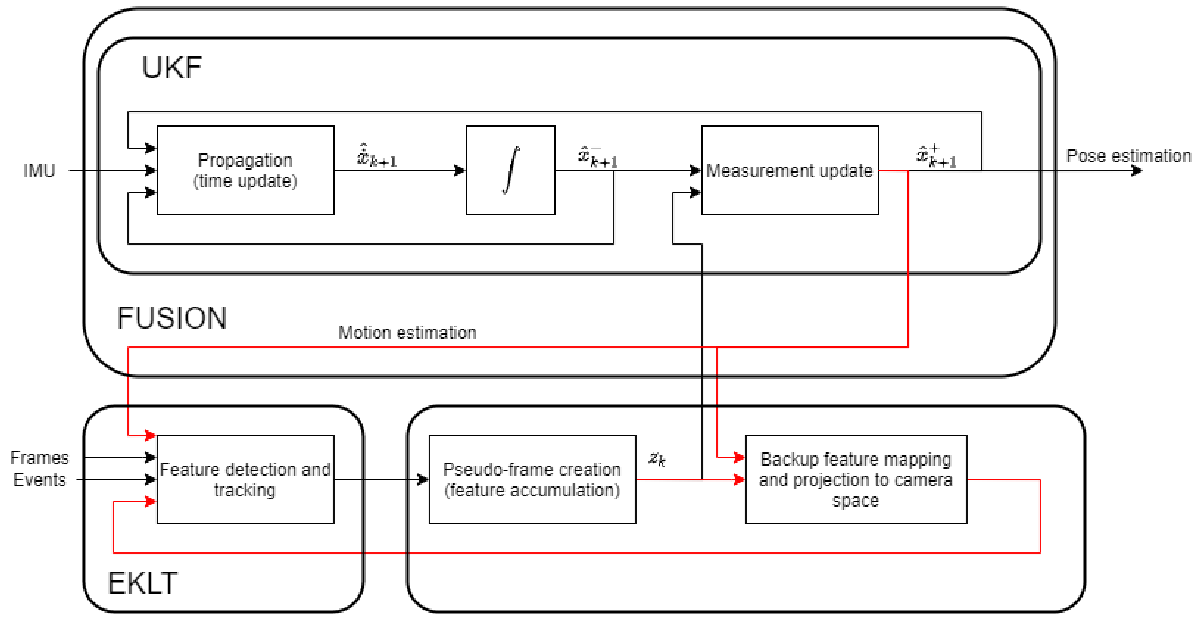

UKF
IMU
Propagation (time update)
$\hat{\dot{x}}_{k+1}$
$\int$
$\hat{x}^-_{k+1}$
Measurement update
$\hat{x}^+_{k+1}$
Pose estimation
FUSION
Motion estimation
Frames
Events
Feature detection and tracking
Pseudo-frame creation (feature accumulation)
z_k
Backup feature mapping and projection to camera space
EKLT

Figure 4.3: Overview of the proposed approach.

Chapter 5

Experiments and Results

The person who can combine frames of reference and draw connections between ostensibly unrelated points of view is likely to be the one who makes the creative breakthrough.

— Denise Shekerjian

In this section we validate our proposed methods and present their results. In particular, Section 5.1 introduces the validation environments and datasets used to test the proposed methods. Section 5.2 characterises the experiments performed and their objectives. The following sections present the results obtained for each experiment, and lastly, in Section 5.6, a critical analysis of the results obtained is performed.

5.1 Datasets and Experimental Setups

In this section we explain the methods which were used to validate our proposed approaches. In particular, we introduce the simulator used for some of the experiments, the datasets available online that use real event cameras, and lastly the recorded datasets using the Kinova robot arm to generate trajectories.

Event Camera Simulator Research on event cameras and event-based computer vision is still in its early stages. One of these reasons is tied to the hardware itself, since event cameras are expensive, not widely available, and are mostly prototypes, with low resolution (no more than 346x260 in the case of a high-end DAVIS346) and therefore not ready for commercial applications. Another such reason is the need for data and datasets, which are also scarce for the time being.

To tackle these problems, a number of simulators have been developed, of which ESIM ([Rebecq et al., 2018b]) is an example, developed by Depts. Informatics and Neuroinformatics at ETH Zurich, in a similar spirit to the conventional cameras simulator available. ESIM is an open-source ROS package.

Most event camera simulators continuously render images, and consecutive frames are compared. Events are then generated from big enough differences between frames.

A common problem with this technique is the choice of frame rate, as a naïve approach would have the engine render at 1 000 000 frames per second, in order to match the microsecond temporal resolution of event cameras.

However, this is not ideal, as many frames are redundant, and therefore performance is affected. An ideal approach would only generate frames that are guaranteed to generate events, but at the same time should have a temporal resolution comparable to event cameras. ESIM uses a tightly coupled system between the simulator and the rendering engine, allowing for a better simulation, by using an adaptive sampling rate based on the dynamics of the scene. Fig. 5.1 shows the ESIM architecture.

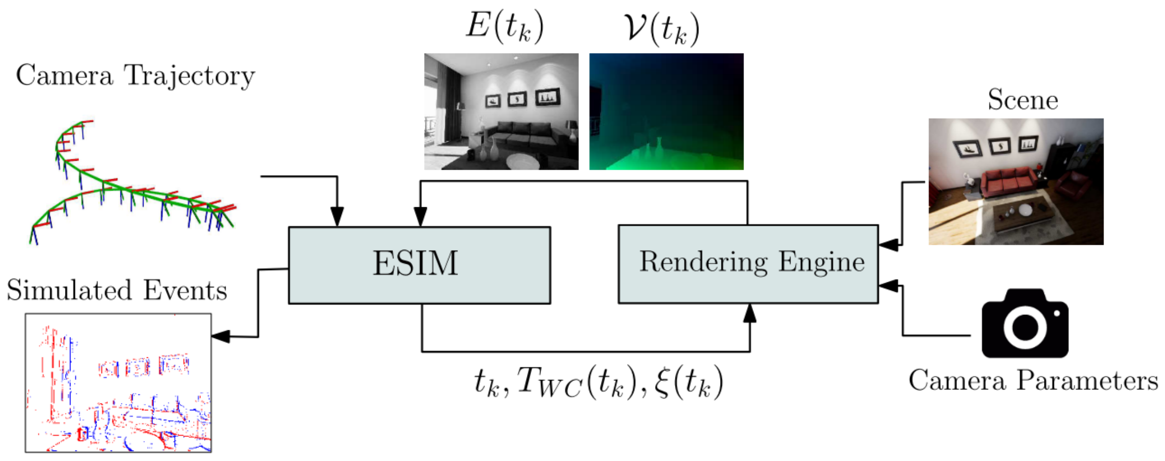


Figure 5.1: Architecture of ESIM, showing the tight coupling between the simulator and rendering engine, sharing the information of time t_k, camera pose $T_{WC}(t_k)$ and camera twist $\xi(t_k)$, generating irradiance map $E(t_k)$ and motion field map $\nu(t_k)$, from [Rebecq et al., 2018b].

Fig. 5.2 shows the advantage of an adaptive sampling approach as opposed to a uniform sampling approach. In the latter, rapid changes in the environment are sometimes overlooked.

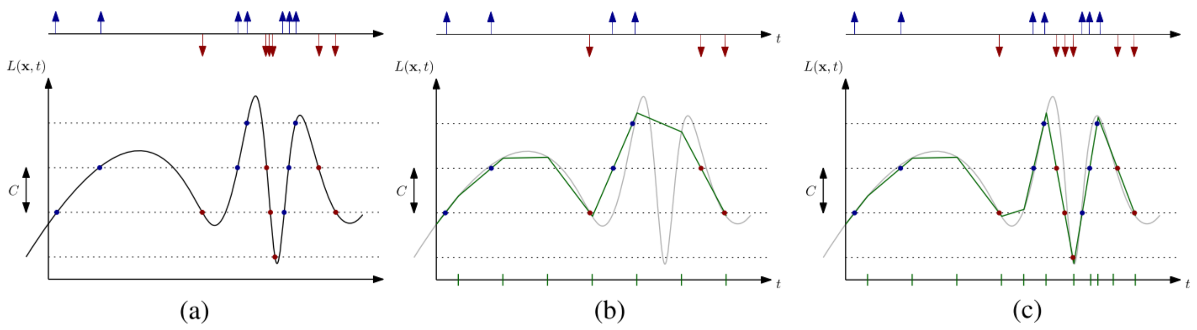


Figure 5.2: Comparison of a uniform sampling approach (b) and an adaptive sampling approach (c) when recreating the reference response shown in (a), from [Rebecq et al., 2018b]

This adaptive sampling is possible due to a communication between the simulator itself, and the rendering engine being used. It relies on the knowledge of the trajectory of the virtual camera to estimate the motion on the scene, which is used to guess changes due to brightness, pixel displacement, noise and non-idealities, to adapt the sampling rate and the event generation.

ESIM allows the use of multiple rendering engines and modes, such as as an OpenGL integration, as well as a photorealistic rendering using Unreal Engine. The simulation of stereo cameras, and panoramic cameras is also possible. Furthermore, it is possible to “convert” videos to events, though the results are not as good, due to the fixed sampling rate of the videos, and not using feedback from the rendering engine.

The generation of events is as previously described. This simulator also allows for the generation of images, in effect replicating the more advanced DAVIS event cameras. For the generation of frames, a simple capture of the render from the viewpoint of the camera is performed, at specific intervals, corresponding to the framerate of the camera being generated. It is also possible to simulate exposure time, producing images that are subject to motion blur (useful for a "fairer" comparison in high-speed motions, though it falls short of a real system in this regard).

The trajectory performed by the virtual camera is crucial for generating the sensor readings, in particular the IMU measurements (which include both accelerometer and gyroscope readings), as well as groundtruth trajectory values (for posterior validation of methods). An explanation of the type of information provided by IMUs is included in Section 2.3.

The simulator can generate a random trajectory, by defining a random set of points, and can also receive a trajectory predefined in a file. In both cases, the points being supplied are only indicative, and internally the simulator fits a spline to the trajectory. This concept is shown in Fig. 5.3, and is crucial to generate a smooth trajectory that can be continuously differentiable (notice the jagged trajectory supplied and the smooth trajectory received, shown in Fig. 5.3).

The importance of the smooth trajectory is that it can be directly related to the sensor readings. Taking the accelerometer, for example, it measures the linear acceleration of the camera in space. Well, the acceleration of a body can be given as the second derivative of movement (the first derivative being velocity). Likewise, velocity and position can be obtained (apart from a constant) by integrating the acceleration.

As a result of the smooth trajectory from the spline, which is continuously differentiable, we can obtain the accelerometer reading for that axis by differentiating the trajectory twice, as shown in Fig. 5.3, which compares the accelerometer reading obtained by differentiating the trajectory twice, and the reading that was in fact generated by the simulator. It is possible to observe they coincide, apart from a slight noise and bias that the simulator generates in order to generate more realistic data.

Likewise, for the other sensor readings (namely gyroscope), a similar approach is used, but instead taking into account the rotation along the trajectory.

Using ESIM, we have created many test scenarios of different complexities, ranging from single axis movement to all axis, from rotation-only or translation-only, to combined movements. Fig. 5.4 shows a sample scene generated with ESIM, and corresponding generated events. Even though simulators cannot perfectly mimic the real system, much work was put into getting ESIM close to matching a real event camera, and many authors use it to test their approaches.

Public Datasets Datasets, like simulators, provide an interesting way to test algorithms without the need for a camera, compare results with other algorithms, and have a ground truth with which to benchmark. Due to the novelty of event cameras, there are no “standard” datasets to which to compare against. In fact, not many datasets have been publicly made available.

In [Mueggler et al., 2017b], a handful of datasets were introduced and made publicly available. The Depts. of Informatics and Neuroinformatics at ETH Zurich provides a set of event camera datasets [1], recorded with a

[1]`http://rpg.ifi.uzh.ch/davis_data.html`

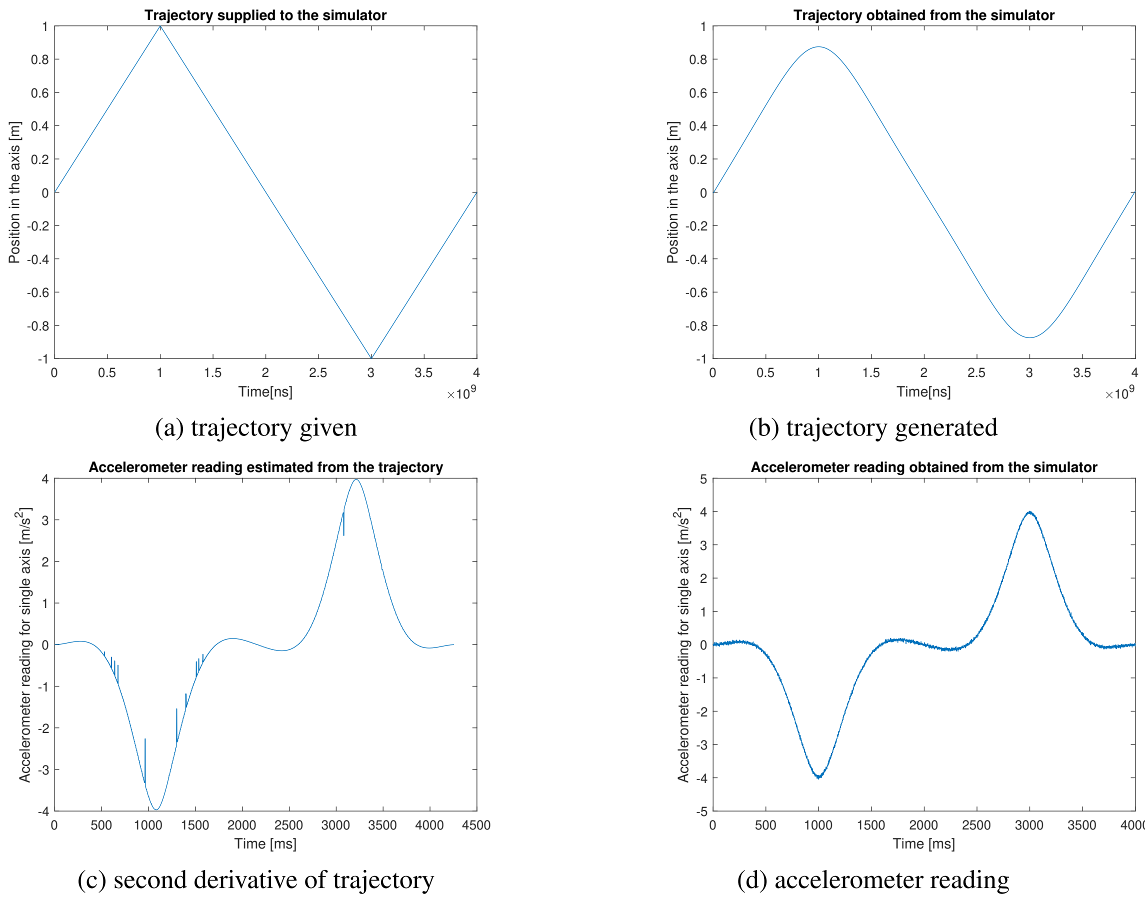


Figure 5.3: Comparison of the trajectory that was supplied to the simulator (a) vs. the trajectory that was performed by the simulator (b), and provided in the trajectory groundtruth. The trajectory corresponds to a movement in a single axis, back and forth. Comparison of the accelerometer reading obtained by differentiating the trajectory twice (c) vs. accelerometer reading that was obtained from the simulator (d). Notice the simulator adds some noise to simulate a real sensor. The spikes in (c) result from the discrete approximation of the derivative, but are not relevant for the point being made and the comparison.

DAVIS240C (an event camera with both events and full frames), along with IMU measurements and ground truth (position and orientation) from a motion-capture system. Some of these datasets were used to test the algorithms developed, as other authors have chosen to use them as well.

These datasets were recorded by moving a DAVIS240C event camera by hand, as such the trajectories produced are random and not necessarily "precise". Different scenes were recorded, with more textured environments and less textured environments being available (Fig. 5.5).

These datasets share a common pattern: they get progressively difficult by increasing speed. Furthermore, there are datasets that focus on rotation, others translation, and others all 6 DOF. However, since the movement was done by hand, even the isolated rotation or translation dataset have some undesired motion.

Kinova Datasets In order to have a more rigorous dataset in terms of movement, as opposed to the datasets with movement by hand (in particular, to generate a dataset that more closely mimics the way an eye works, taking into

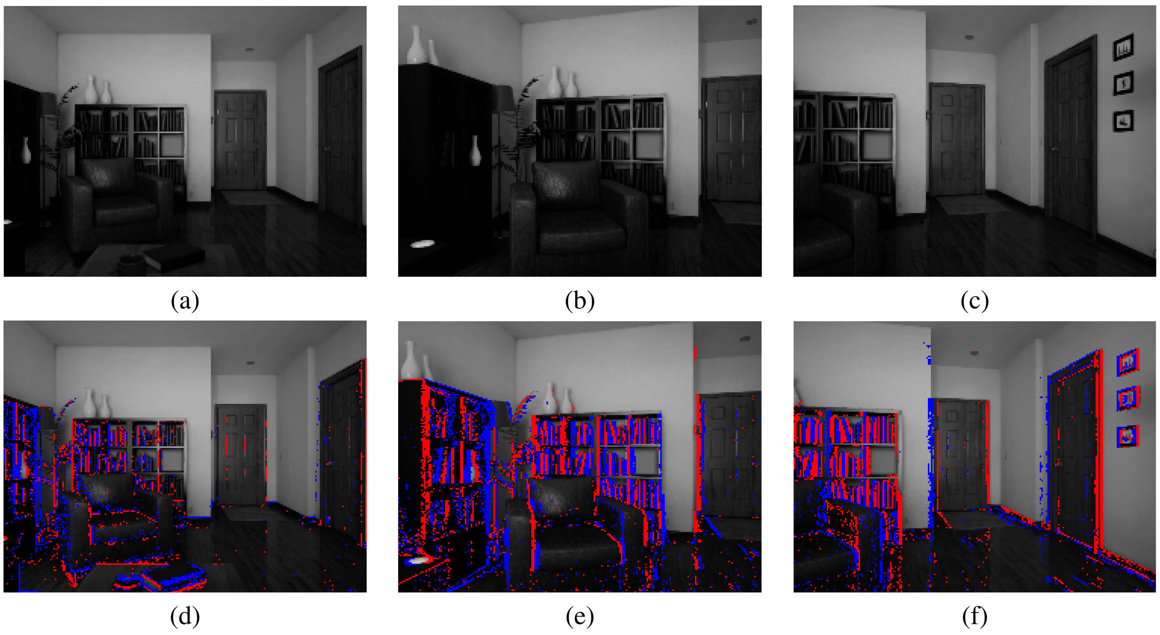


Figure 5.4: Example of scene generated using ESIM. (a) through (c) show frames produced at different instants, and (d) through (f) show events produced in those instants superimposed on the corresponding frames.

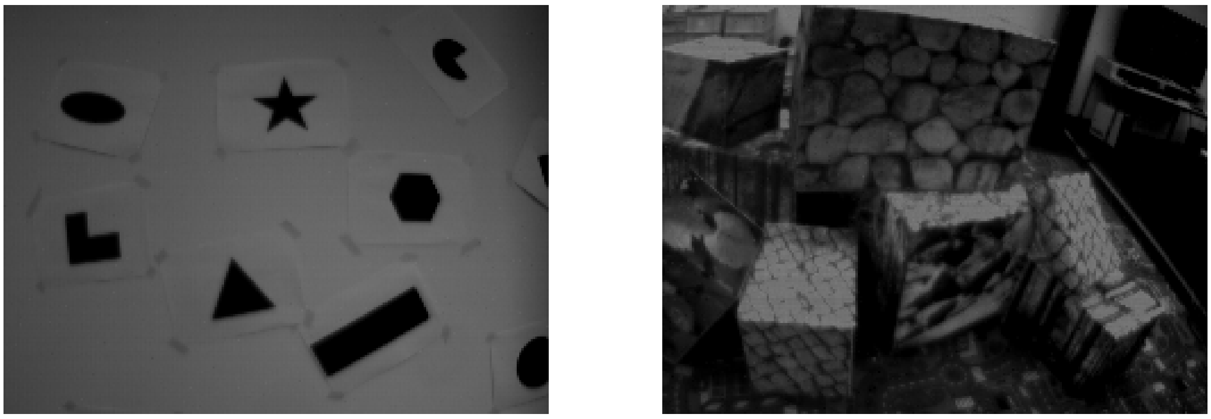
account the context of the ORIENT project this work is part of), we recorded a dataset using the Kinova Gen3 robot arm (Fig. 5.6.(a)), with the hope that the robot arm can perform a more controlled and precise movement, mimicking the eye saccade.

The setup of this experiment relies on a Kinova arm that grips the camera with its end effector (for this purpose, a special end effector was designed and 3D printed to fit the DVS240). A set of poses is pre-generated, and the inverse kinematics are computed offline for the Kinova arm specifically. The trajectories generated follow the saccadic velocity profile. Furthermore, a motion capture system was used to record the pose of the camera at all instants.

In order to create a dataset relevant for the context of an eye, we tried to replicate the eye saccades. Unfortunately, the robot arm has some velocity and acceleration limitations, which means such speed were not attainable, and were quite slow when compared with the eye. Nevertheless, the acceleration and velocity profiles follow the correct shape (the acceleration approximately follows the pattern of an arctangent function, and the velocity its derivative $(1/(1+x^2))$), albeit much slower. This limitation does not impede the validation of the proposed approaches *per se*, but somehow lessens the advantage that event cameras may have had in high movement scenes.

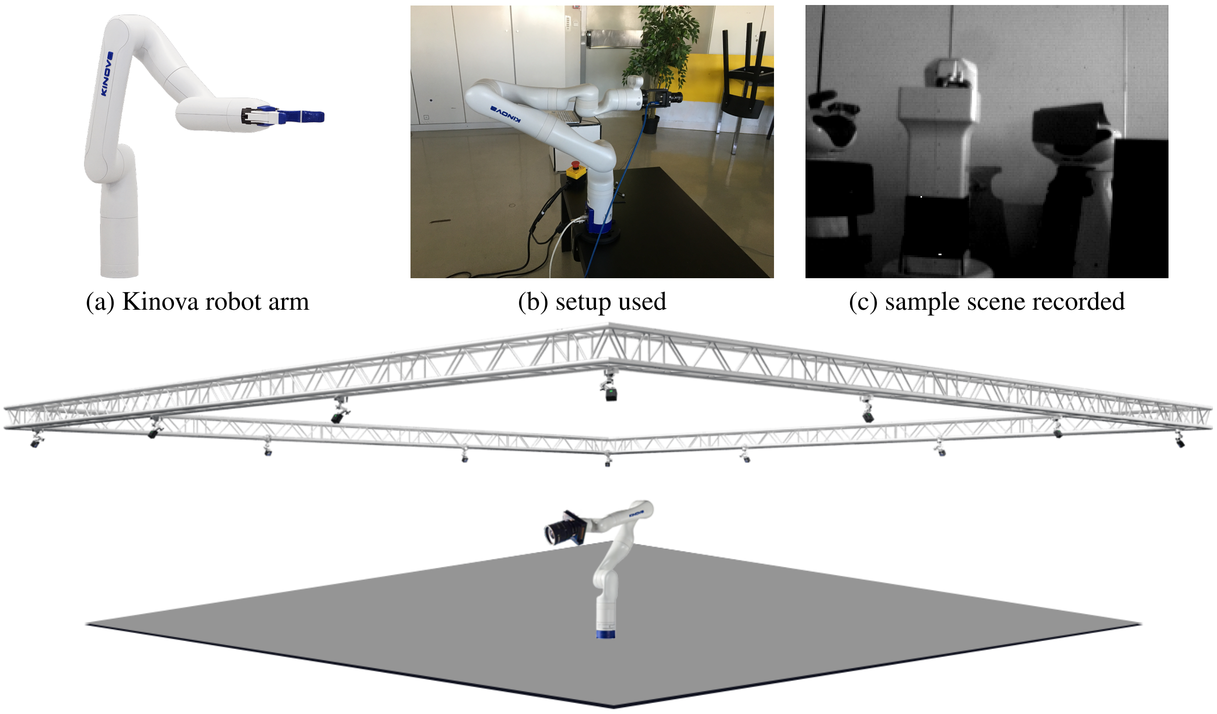
With this setup, we recorded visual and inertial information from the camera, as well as groundtruth information from the motion capture system, and directly from the feedback of the arm (each servo's angle and velocity were being recorded at each instant). Fig. 5.6.(b) Fig. 5.6.(d) and shows the setup used for dataset generation.

However, since we were using a DVS240 camera, which cannot record frames and events simultaneously, three recordings were captured: 1) recording frames 2) recording events; and 3) recording frames when still and events when moving. Fig. 5.6.(c) shows the scene captured using this technique.



(a) *shapes* scene (b) *boxes* scene

Figure 5.5: Example scenes available on the public datasets provided by ETH Zurich. (a) shows a less textured scene but with clearer edges and corners. (b) shows a more textured scene.



(a) Kinova robot arm (b) setup used (c) sample scene recorded

(d) sketch of setup used for recording

Figure 5.6: Setup for dataset recording showing the Kinova robot arm for trajectory generation and groundtruth recording, and surrounding motion capture system for groundtruth recording. The DVS240 camera is coupled at the end of the arm.

5.2 Experiment Plan

In this section we characterise the experiments that were performed in order to test the proposed approaches, which consisted of data using simulations and real datasets. Given the group's interest in the visual and vestibular system, the focus of the work (and therefore, of the results presented), was directed towards rotation movement (though the trajectories themselves may (and do sometimes) have translational components).

Experiment 1, DAVIS240 Public Dataset using Events, Frames, and IMU For this experiment, we tested the proposed approaches in the publicly available datasets from ETH Zurich. Such tests allows us not only to test the approaches in a real environment, with all noises and uncontrollable factors associated with real systems, but also to compare them with other groups and approaches. Since a DAVIS240 is used, the setup is similar to the simulation (which was created to mimic this exact scenario).

Experiment 2, Simulation using Events, Frames, and IMU In this experiment we test our system with simulated information, as it provides a controlled setting on which to evaluate the performance of the approach. This setting simulates a DAVIS240 camera, and frames, events and IMU information is present at all times. Multiple trajectories were generated, from isolated translations and/or rotations in a single axis, to more complex movements. We present the results of some of these tests. We also perform an ablation experiment, where we tried to isolate the relevance of the IMU in the system (thus testing a visual odometry system as opposed to the proposed visual inertial odometry system). Instead of the Kalman filter, we tested this experiment using the Procrustes approach. Our reasoning for this choice is as follows: 1) there is no need for sensor fusion in this approach, as only visual information is used, and 2) Procrustes finds the optimal solution (whereas the UKF is not guaranteed to). Finally 3) we want to compare our results against the ones previously obtained in [Martins, 2019].

Experiment 3, Kinova with DVS240 using Events and/or Frames, and IMU The objective of this setup is to evaluate the viability of the approach in the context of saccadic eye movement estimation, which is of the interest of the group in which this work is integrated. As such, we used a Kinova robot arm and developed software to generate controlled trajectories following the profile of the saccades. Motion capture cameras were also used to record groundtruth. It is important to note the arm is not able to generate the values of velocity and acceleration associated with a saccadic movement, but allows running controlled experiments, useful for development of the motion estimation methodology. The camera used is a DVS240 which is lower cost as compared with the DAVIS240, used in Experiment 1 and simulated in Experiment 2, and for which the proposed approaches were designed and the base software tools were developed. Using the DVS240 is a challenge whose success can provide major savings in the total cost of the setup.

5.3 Experiment 1, Integrated Experiment with a DAVIS Camera Dataset

To test the proposed approaches, we tested the performance on the datasets available online, which consist of a series of movements of the camera, by hand, on multiple scenes. In particular, we tested the approach on the *shapes* and *boxes* datasets, as the former has clear contrast between background and shapes, and the latter has a much more textured environment.

We focused mainly on the rotation aspect of the movement, as this is the one that is most relevant in the context of the project. Since the images produced are from a real system, they have associated distortion parameters, noise, motion blur and all these undesired properties that should be overcome.

Open Loop Approach

We start with the first proposed approach. The inputs to the system are shown in Fig. 5.7, where we show 3 illustrative frames on the movement, and superimpose the events on frames, making evident their relation with the edges and corners of objects in the image.

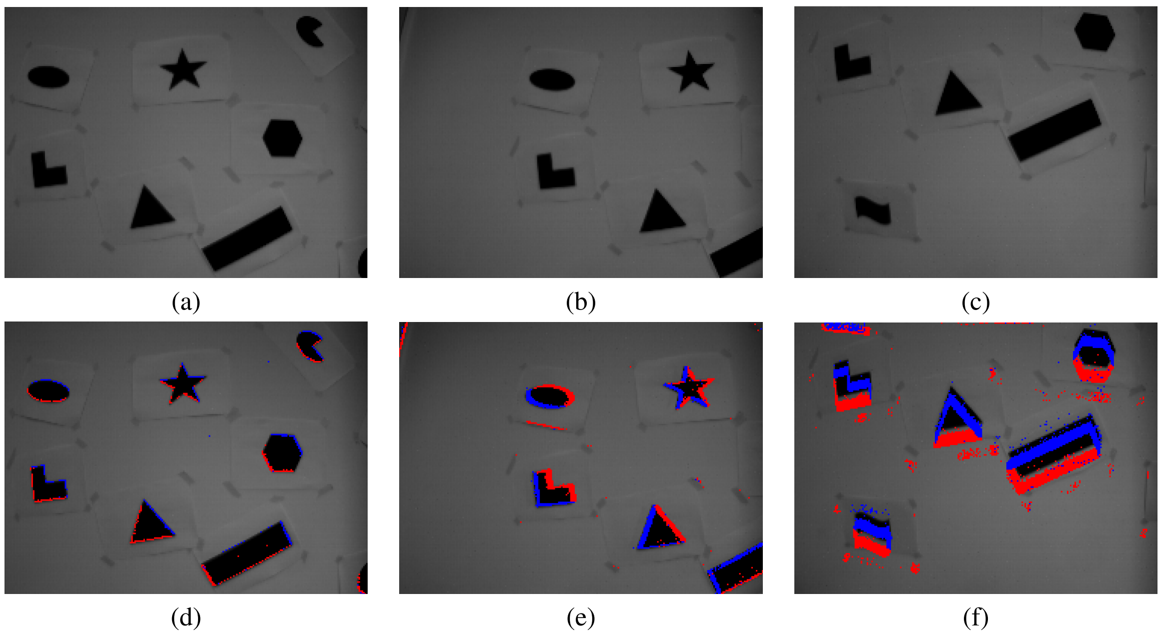


Figure 5.7: Input used for the experiment. (a), (b) and (c) show sample frames along time. (d), (e) and (f) show the same instants, but with the generated events superimposed.

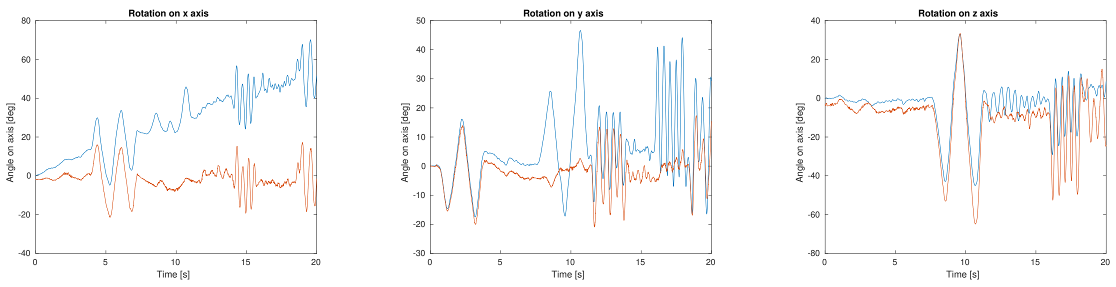
The results running this approach are presented in Fig. 5.8 and Table 5.1. We have decided to isolate each axis estimation for the sake of a less cluttered analysis, as well as to interpret the evolution of each axis independently.

Movement	Mean error [deg]	Max error [deg]	Std dev [deg]	RMSE [deg]
Rotation x axis	-30.49	-58.04	16.10	34.48
Rotation y axis	-8.14	-45.70	10.45	17.35
Rotation z axis	-7.03	-32.46	6.63	9.66

Table 5.1: Results obtained running the open loop approach in the *shapes* scene.

It is possible to observe that the obtained results are not exactly satisfactory. First, there is an obvious drift in the x axis that was not able to be compensated, caused by an uncompensated bias in the gyroscope, as this axis more easily loses features by moving out of the FOV, which means that the local map may itself drift over time, and not correct sensor bias.

Then, for the remaining axes, there is some oscillation that can reach significant error values. Curious enough, these occur when there is movement in another axis. For example, around the 10 s mark, there is oscillation in the y axis, caused by the movement in the z axis. Nevertheless, there is some positive aspects, in the sense that the general shape of the rotation is indeed estimated, albeit not very well.



(a) x axis (b) y axis (c) z axis

Figure 5.8: Results obtained running the first approach on the *shapes* scene. The orange represents the groundtruth value, and the blue the estimated value.

Though the dataset takes about 60 s, we only present around 20 s of estimation, as the filter panicked and crashed a little bit after this value. This is likely because the drift in the x axis, which leads to incorrectly placed landmarks, and some inconsistencies between the observed features and expected position.

Furthermore, our conclusion is that the features being fed into the filter are responsible for these poor results. Analysing the number of features over time (Fig. 5.9, it is possible to observe that not only is the number of features low (usually less than 25 features at a time), they also oscillate quite a bit.

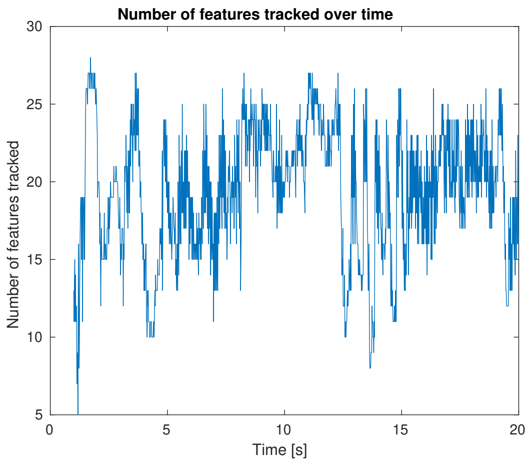


Figure 5.9: Evolution of features over time using the *shapes* dataset.

After, we tested the proposed approaches on the *boxes* scene. The inputs to the system are shown in Fig. 5.10, where we show 3 illustrative frames on the movement, and superimpose the events on frames, making evident their relation with the edges and corners of objects in the image.

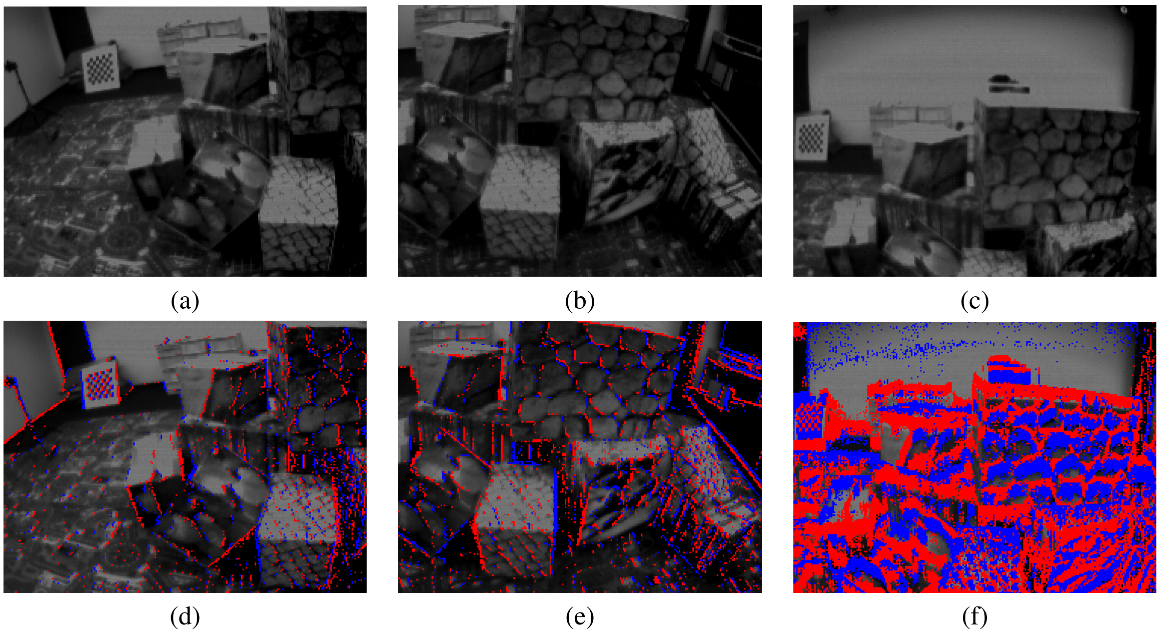


Figure 5.10: Input used for the experiment. (a), (b) and (c) show sample frames along time. (d), (e) and (f) show the same instants, but with the generated events superimposed.

The results running this approach are presented in Fig. 5.2 and Table 5.2.

Movement	Mean error [deg]	Max error [deg]	Std dev [deg]	RMSE [deg]
Rotation x axis	-20.40	-50.10	12.04	30.88
Rotation y axis	3.63	-10.25	4.94	6.13
Rotation z axis	5.96	-12.27	6.48	8.80

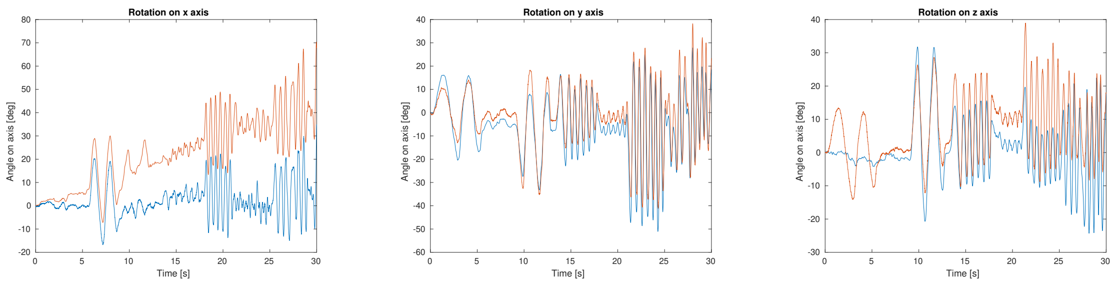
Table 5.2: Results obtained running the open loop approach in the *boxes* scene.

This experiment still produces some clear deviations on the true values. However, these results are slightly better than the ones presented previously, as not only are the errors smaller (with the exception of the x axis rotation, that shall be addressed in a moment), the overall profile of the estimation more closely follows the true values, which is positive.

This difference has to do with the environment, in particular the much more textured environment, which means that more features can be, and are, in fact, detected (Fig. 5.12), which allows for a more forgiving tracking of features fed into the filter, as the higher number means the relative importance and impact of each feature in the feature is lower.

Comparing with the *shapes* dataset,we have 2-3 times as many features bring tracked, which obviously help estimation. The result is that the local map is created more accurately (and features lost in the filter can be more easily replaced), which improves overall performance.

However, results are still far from perfect, in particular the x axis rotation still drifts (up and down movement), meaning the filter still cannot estimate the bias in this axis. Nevertheless, the drift seems to be less than in the



(a) x axis (b) y axis (c) z axis

Figure 5.11: Results obtained running the first approach on the *boxes* scene. The orange represents the groundtruth value, and the blue the estimated value.

shapes dataset, which leads us to think bias is partially being corrected. As in the previous scene, when the camera moves up, there are less textures, and, since the camera space is rectangular, it is easier for features being tracked to fall out of the FOV, though this argument has much less force in this scene.

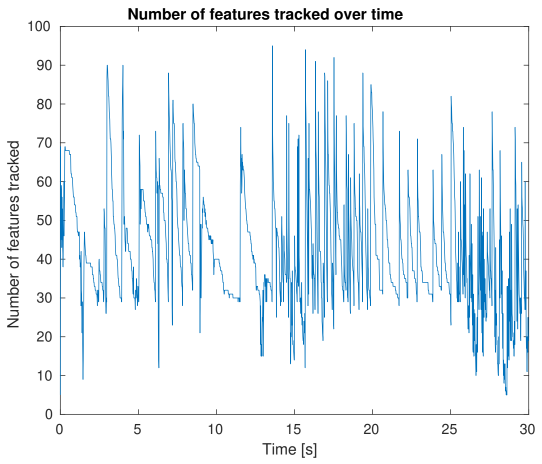


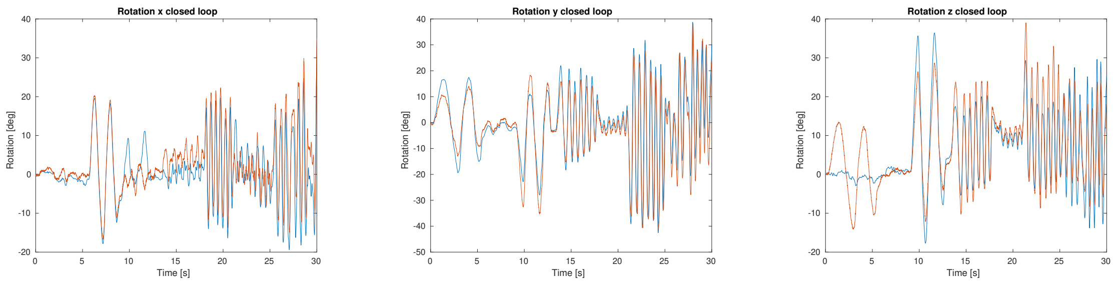
Figure 5.12: Evolution of features over time using the *boxes* dataset.

Closed Loop Approach

Lastly, we tested the closed loop approach on the *boxes* scenario. The inputs to the system are the same, shown in Fig. 5.10. The results running this approach are presented in Fig. 5.13 and Table 5.3. It is possible to see that the proposed approach does help with tracking, as the results show an improvement over the previous approach.

Movement	Mean error [deg]	Max error [deg]	Std dev [deg]	RMSE [deg]
Rotation x axis	4.35	23.51	5.04	6.66
Rotation y axis	-1.53	-10.77	4.88	5.11
Rotation z axis	1.05	14.89	5.29	5.39

Table 5.3: Results obtained running the proposed closed loop approach in the *boxes* scene.



(a) x axis (b) y axis (c) z axis

Figure 5.13: Results obtained running the closed loop approach on the *boxes* scene. The orange represents the groundtruth value, and the blue the estimated value.

For the sake of transparency, however, it should be stated that these results were obtained by initialising the filter with the bias of the sensors closer to the real one (obtained by the results of the previous approach). Though we argue such an approach is valid, as it is a parameter that can be calibrated previously, and many works also assume this value is know (and therefore initialise their filters with it), it is still very much worth mentioning, as quality of estimation improved not only because of the closed approach, but also because of this. Without this initialisation, the mismatch between inertial information and visual information is not compensated and results are very poor (as the closed loop approach is less robust). We chose not to do the same with the previous approach, as being able to converge to correct (or better) values with as little prior information of the system as possible is a positive aspect.

5.4 Experiment 2, Integrated Experiment on Simulation

To validate the implementation proposed in Section 3.4, we generated a synthetic dataset based on ESIM (Section 5.1). We generated a simple rotation along the z axis. The trajectory generated consisted of a rotation along the z-axis, with an amplitude of 18 deg for each side. Speed was 18 deg/s, so that the whole movement took about 4 s. A DAVIS240 camera was simulated, with both access to frames and events simultaneously.

The images produced have some distortion parameters, and some noise, but have no motion blur when simulating conventional frames, which would (in principle) give the upper hand to event cameras. The inputs to the system are shown in Fig. 5.14, where we show 3 illustrative frames on the rotation movement, and superimpose the events on frames, making evident their relation with the edges and corners of objects in the image.

Open Loop Approach

Starting with the first proposed approach, we tested its performance using simulated data. We compare the features extracted by the original feature extractor and tracker, and EKLT, in Fig 5.15. In particular, we show their position over time, which match their expected movement over time given the motion of the camera.

We also analyse the number of features over time that each tracker provides, and present it in Fig. 5.16.

The groundtruth is compared against an approach using conventional cameras, and using event cameras. The results are presented in Fig. 5.16, and their values presented in Table 5.4.

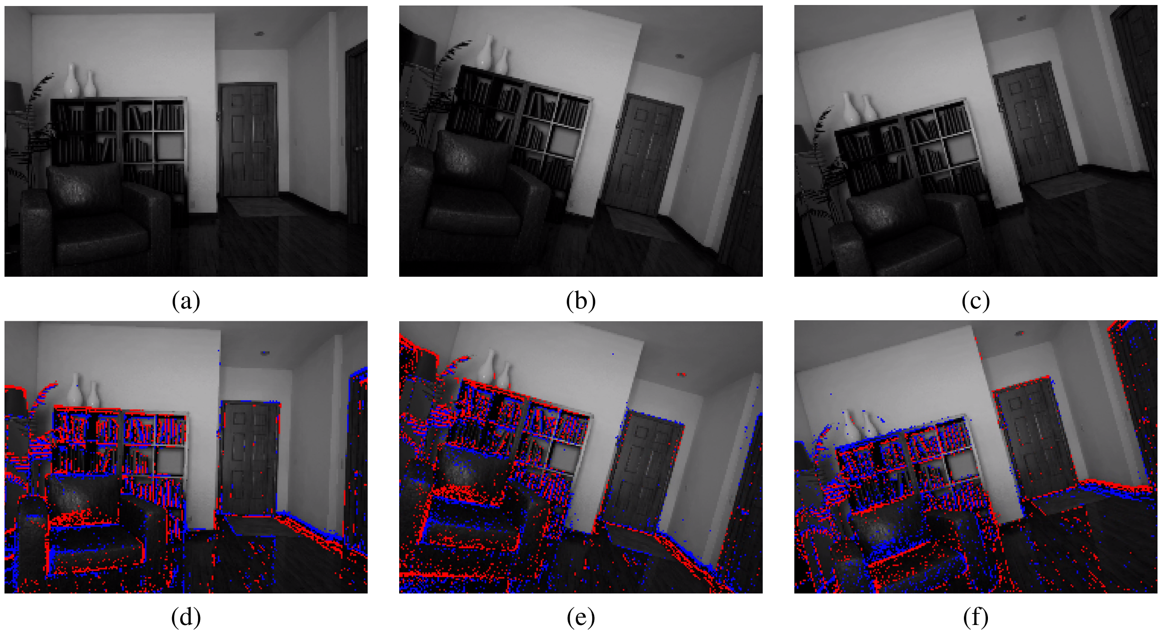


Figure 5.14: Input used for approach using only visual information (visual odometry). (a), (b) and (c) show the evolution of the rotation along time, with rotation movement towards both sides on the z-axis. (d), (e) and (f) show the same instants, but with the generated events superimposed. This input was generated using ESIM (Section 5.1).

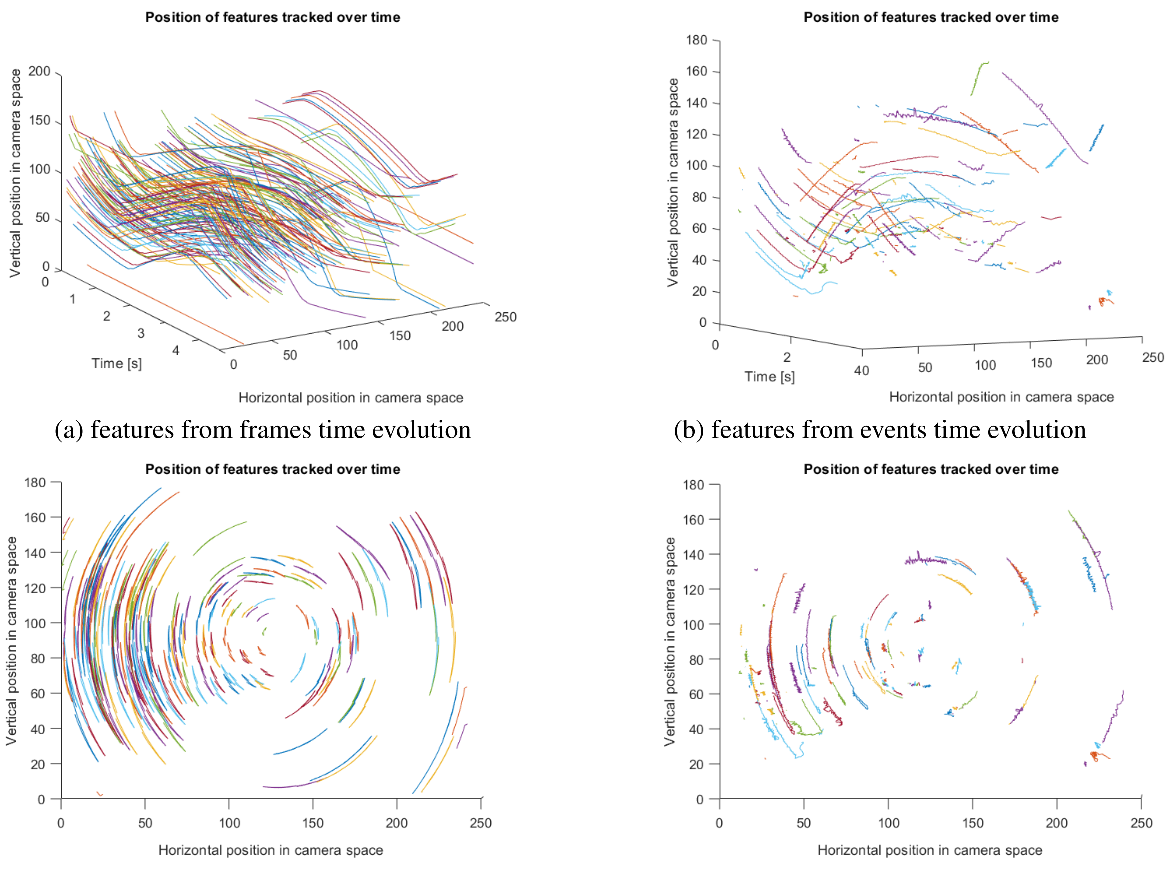
Setup	Mean error [deg]	Max error [deg]
Image + IMU	1.05	4.41
Image + Events + IMU	0.85	3.05

Table 5.4: Results obtained running the proposed visual inertial odometry approach with ESIM dataset, and comparison with a conventional setup performance.

It is possible to observe that including visual information with inertial information is beneficial in the sense that vision contributes to a better estimation, as the spikes in the estimation correspond to the update steps of the filter, which generally point towards the correct value.

Furthermore, we observe that the approach with events outperforms the approach using frames, both in terms of the mean error, and the max error. This has to do with two factors: the more frequent updates that events allow mean that the filter can be updated with visual information more frequently; the features, being tracked with events, are tracked better (which can be debatable as it has their own problems, as lack of good descriptors, and being limited to corner features).

There are some other interesting points to be made by analysing Fig. 5.16. Instants t=1 s and t=3 s are periods of inversion of direction and, therefore, zero speed. Notice that the number of features being tracked from frames (Fig. 5.16.(a)) remains largely the same, regardless of speed. The situation is completely different for events. Not only do they start decreasing as the speed decreases, they also start increasing as the movement picks up (Fig. 5.16.(b)). This is the interesting distinction between both types of cameras. As long as there is no motion blur, or the displacement too big, conventional cameras are indifferent to motion, but event cameras need it to



(a) features from frames time evolution

(b) features from events time evolution

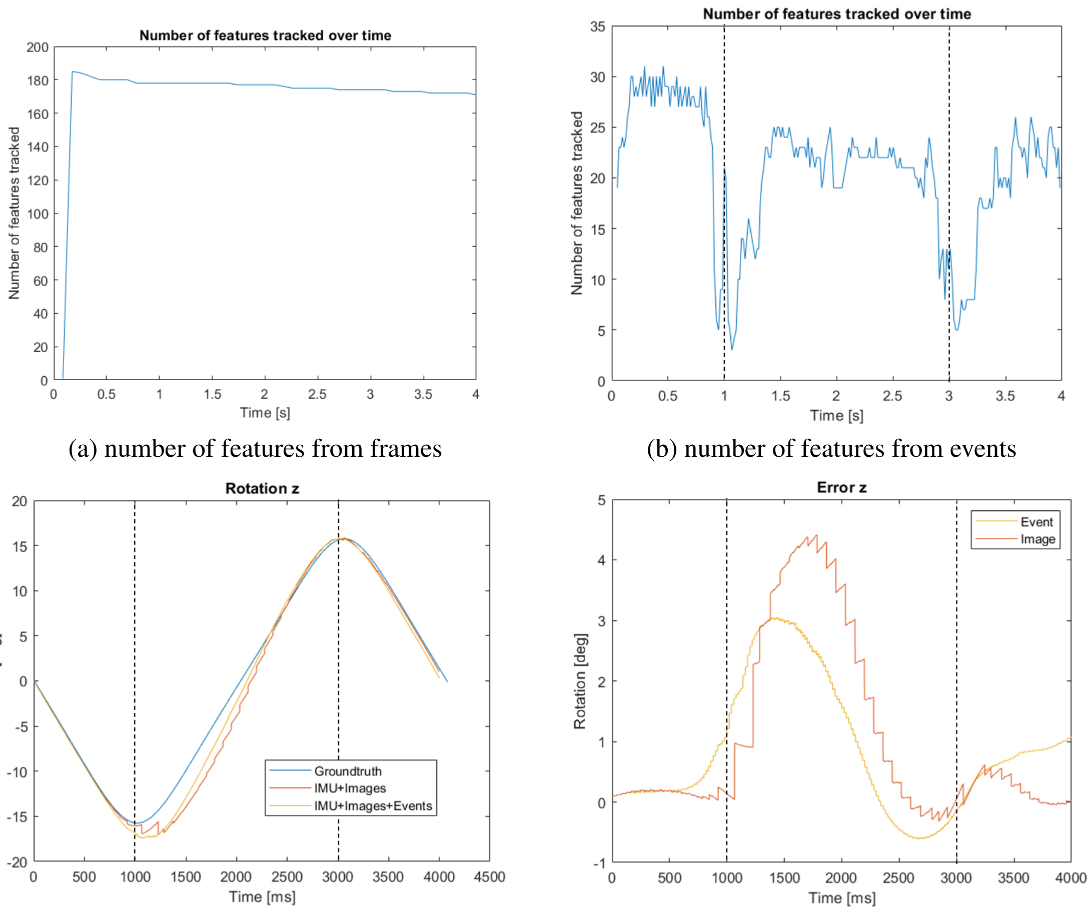
(c) features from frames all time instants

(d) features from events all time instants

Figure 5.15: Frame vs event based features for a rotation about camera Oz. (a) trajectory of features obtained from frames vs time, and (c) trajectory of features on the camera frame, all time instants superimposed. (b) and (d) show similar information, but with features obtained from event cameras. In (b), it is also interesting to observe the loss of features at the inversion of movement, and their later re-detection (albeit with a new ID). (c) and (d), showing the superposition of the features at all times in the camera plane, is compatible with the theoretical prediction of a rotation around the camera z axis.

live. Notice the spike on t=1 s, where features are detected (because they are extracted from a frame), but are immediately lost, due to the lack of events associated with a change of direction. No motion destroys the usefulness of event cameras.

Obviously, this has consequences on the estimation of the filter. Both filters are still converging to the correct values of the biases and landmarks. As such, after t=1 s, there is a period where mismatch between vision and inertial systems, which does not occur (or at least, as abruptly) for t=3 s, when there is another change in direction (Fig. 5.16.(c)). However, notice that performance of the event camera system starts to degrade even before t=1 s, as is evident by the error in Fig. 5.16.(d). The error starts increasing before the change in direction because visual information is lacking, therefore not compensating the IMU bias which is still converging to its correct value, and the estimation drifts. The error from conventional information only starts increasing after the inversion, as this is when there is a more significant mismatch between the visual information and the non-compensated IMU. However, because visual information from events is more frequent when available, the estimation actually compensates



(a) number of features from frames

(b) number of features from events

(c) rotation about camera Oz vs time

(d) error on estimated rotation

Figure 5.16: Number of features being provided over time. (a) shows features obtained from frames, and (b) using events. Also shown are the results obtained when using the UKF being proposed. Comparison of the groundtruth with the approach using frames and frames+events (c), and the error between methods with regard to the groundtruth (d).

faster, and the maximum error value is both lower and achieved faster, showing a faster convergence to the correct estimate.

Though not directly relevant for the objective of this group (and by extension, to the analysis and testing performed in this work), we also present the estimated translation over time, to illustrate that the filter is working correctly. Fig. 5.17 shows the estimated position of the camera over time, which takes some time to converge but seems to be bounded appropriately, producing a low error overall, as shown in Table 5.5.

	x axis	y axis	z axis	average
RMSE [m]	0.13	0.10	0.014	0.08

Table 5.5: Error of position estimation on ESIM with open loop approach

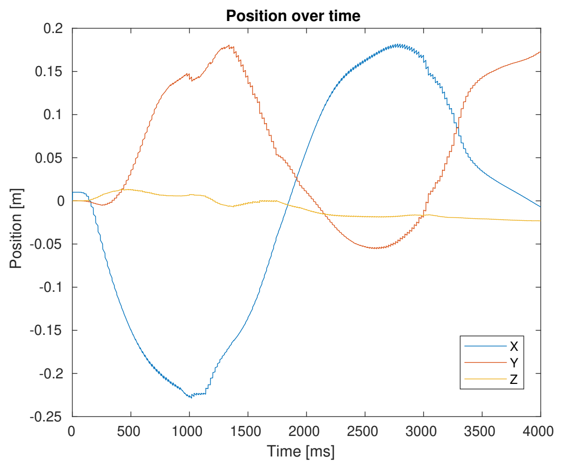


Figure 5.17: Position estimation over time in ESIM with open loop approach.

We repeated this same setup, in the same scene, but with other motions, in particular rotations in the other axes. The rotations follow the same pattern as presented for the z axis, with a 18 deg rotation to each side. The results are summarised in Table 5.6

Movement	Mean error [deg]	Max error [deg]	Std dev [deg]
Rotation x axis 18 deg	1.66	5.04	1.70
Rotation y axis 18 deg	0.65	2.43	1.11

Table 5.6: Results obtained running the proposed visual inertial odometry approach with ESIM dataset for the remaining possible axes.

The results are in line with the ones presented for the z axis rotation presented, with the maximum error being achieved on the first change of direction (when some features are lost, the filter is still converging to the correct values, and the internal estimation ramps up towards the direction it was going, and now takes some time to invert the direction), and then being able to correctly keep up with the real values.

Overall, the results are very uplifting, and validate the proposed approach, even if with a simulator, and outperform the conventional camera approach in all cases (even though sometimes a change in the settings for the filter or the tracker is needed). For a fair comparison, though not exhaustive, care was taken to make sure the results using conventional cameras were favorable. Nevertheless, we do not reject the hypothesis that there are better parameters that we did not select.

Rotation-Only Hypothesis

In order to analyse the reliability of the features tracked using events, we tested EKLT with the event visual odometry approach mentioned in Section 3.3. The results are presented in Fig. 5.18. The groundtruth is compared against an approach using conventional cameras, and using event cameras. This approach, even though simple (as only rotation is being estimated, and only one axis is being excited), is enough to illustrate the viability of using events as sources of data in the system.

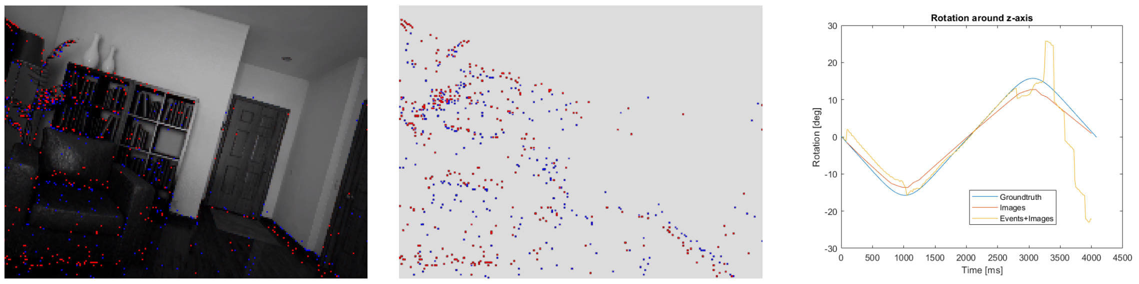
Though final results are what grabs our attention, and the approach using events loses effective estimation at around 3 s, it is nevertheless interesting to analyse what is happening behind the scenes. Furthermore, even though this approach eventually “blows up”, it is able to maintain an estimation close to the groundtruth.

The first point worth analysing is the number of features that are being supplied to the system on both cases. Using frames, we have around 170 features being tracked at each time, whereas using events, the number of features oscillate quite a bit, but always remain under 30 features at all times. This in itself is worth mentioning, as it means that an order of magnitude lower in terms of features is enough to have comparable tracking performance (before estimation is completely lost). This is because the quality of the features being tracked is better.

However, at least two questions arise:

Why the low number of features? This question has to do with the type of features that are acceptable for each system. Conventional cameras have the advantage of years of research, and multiple "types" of features can be used (as seen in Section 2.1.2, corners, blobs, lines, and more can be used as features), and still be distinctive enough to be matched later (Section 2.1.2). However, this hybrid approach of EKLT only tracks corners,

Why the oscillation on the number of features? Analysing Fig. 5.16, the number of features declines at around 1 s and 3 s, which corresponds to moments where there is a change in direction in the trajectory. As mentioned in Section 2.2.2.3, flow is being used to generate the template from frames. In these periods of change in movement, the flow estimation fails, and the matching of features fails more easily. Furthermore, in these changes of direction there is an instant where there are no events being generated (as their generation is dependent on movement, as shown in Fig. 5.18, where there are many less events when compares to Fig. 5.14).



(a) Image and Events (b) Events only (c) Evolution over time

Figure 5.18: Lower event rate being produced when there is a change of direction in the motion of the camera ((a) and (b)). (c) shows results using visual information only. We compare the real value of the rotation with the approach using a conventional camera and using an event camera.

Nevertheless, from this experiment, we emphasise some advantages of the use of events, obviously not without some disadvantages. Some advantages are: 1) Faster tracking of features, which allows for 2) More frequent updates, 3) Better for high movement scenes, and 4) Features detected are more relevant. Some disadvantages are: 1) Less features available (only corners), 2) Less features means the estimation is more erroneous and less robust, resulting in 3) Estimation that is easier to "blow up".

Closed Loop Approach

After testing the first proposed approach, we present the result for the second proposed approach (closed loop) using the same scene and setup. The results are presented in Fig. 5.19 and summarised in Table 5.7.

Movement	Mean error [deg]	Max error [deg]	Std dev [deg]
Rotation x axis 18 deg	1.73	5.89	2.37
Rotation y axis 18 deg	0.049	0.12	0.07
Rotation z axis 18 deg	0.80	4.23	1.41

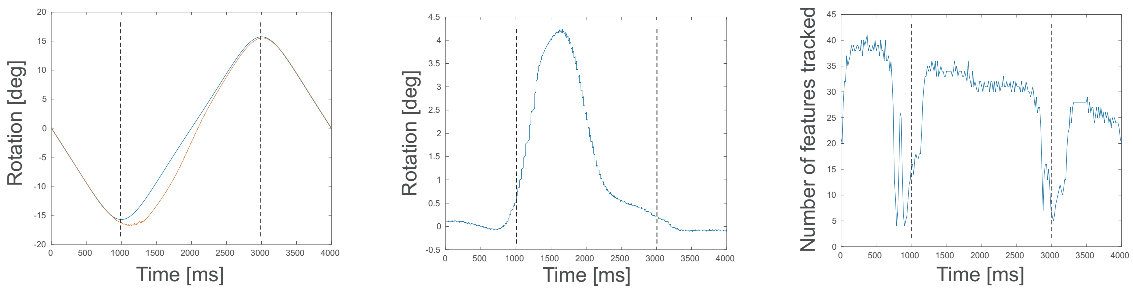
Table 5.7: Results obtained running the proposed closed loop approach with ESIM dataset.

These results seem to corroborate that the closed loop approach does, in fact, improve feature tracking, which, in turn, improves the estimation quality, as these results seem promising (even though simulated data was used as input, which does not necessarily transfer to the real system effortlessly). Notice, in particular, the mark at around 3 s, which improved greatly compared to the first approach, which seemed to drift a bit after this inversion of direction. The closed loop, on the other hand, sticks remarkably close to the correct value.

The number of features being tracked over time is shown in Fig. 5.19, and shows a number that is consistently higher than in the previous case. Nevertheless, for direction changes, the number of features being tracked still dips considerably, as not only are there fewer features being produced, which created poorer matching templates, the filter also takes a while to adjust, which means there is a period where the estimated velocity is still converging to the real one, and therefore the velocities used for motion estimation are not correct.

However, it is interesting (and honestly, reassuring) to observe that there is a period of about 1-2 s for the filter to converge, after which the orientation is correctly tracked with minimal error (less than 0.5 deg). For this experiment, only flow angle is being used to improve tracking.

Nevertheless, some limitations of this approach also presented themselves, the most relevant of which is the lower robustness of the filter. The closed loop presents itself as a double edged-sword, in the sense that good pose estimations lead to better tracking, which leads to better estimation, but a bad starting estimation is very detrimental to the tracker, which loses features very quickly. This can be shown for the case of x and y axis rotations. The former had some biases that hurt the performance of the filter, resulting in a rotation that was worse than the first proposed approach. However, for the former, some parameters must have been optimal, as the error is very low.



(a) rotation about Oz vs time (b) rotation error vs time (c) number of features vs time

Figure 5.19: Results obtained when using the closed loop approach. Comparison of the groundtruth (in blue) with the approach using frames+events (in orange) (a), and the error with regards to the groundtruth over time (b). The features tracked over time are also represented (c).

5.5 Experiment 3, DVS Camera Mounted on the Kinova Arm

In this experiment, multiple rotation-focused movements were performed by means of a Kinova robot arm, in order to mimic the eye saccadic movement, and being able to track it along time. This mimicking was mostly in terms of the velocity and acceleration profiles, not what is humanly possible, as we consider torsional movements, which do not occur in the eyes, for example.

This experiment is meant to close one of the questions that motivated this work, whether event cameras could be used in the context of eye saccadic movement estimation. Though this movement follows a certain pattern that could be used to help with the estimation, namely it is purely rotational, for instance, with an arctangent profile in orientation, no changes to the proposed approaches were made. In other words, the capability of estimating translation is kept.

Since the DVS camera is the camera considered for this experiment, either frames or events may be recorded at each time (exclusive or). The data recording encompasses two parts (i) image frames when the camera is still, and (ii) events when the camera is moving. The commutation from frames to events is not automatic, is placed in the script of the data acquisition. IMU is always recording.

The reasoning for the commutation approach is: when still, frames are not contaminated by motion blur, and features can be extracted from them. Afterwards, when movement starts, by switching to events, the previously identified features can be tracked. The limitation of this idea is that lost features can only be replaced when a new frame is available, meaning the tracking has to endure for as long as possible.

Sample frames that were recorded during this session are shown in Fig. 5.20, which correspond to the lab with some objects placed for texturing. The movement was performed by the Kinova, and groundtruth recorded with Kinova and a motion capture system. We started by performing rotations along the z axis.

After feeding this recording into EKLT it was verified that frames based features are effectively tracked, however the event based features are lost between tens to hundreds of milliseconds after detection. Fig. 5.21.(a) shows the evolution of features tracked over time, and Fig. 5.21.(c) shows their number over time. There are multiple periods of time where all features are lost, and no new features can be detected because no new frames are available. When a frame is finally available, some features can be extracted (see the spikes in the graph).

The first results obtained from this experiment are presented in Fig. 5.21.(d). The plot shows an uncompensated drift due to bias due to lack of visual information correcting it. It is interesting to observe that when features are

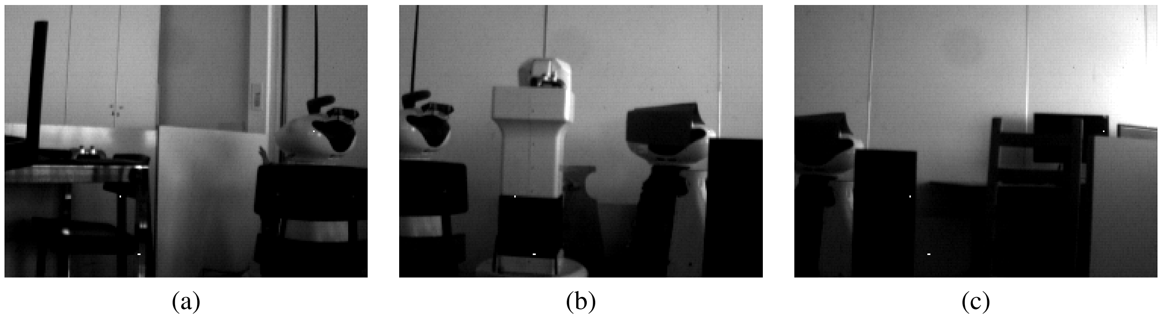


Figure 5.20: Example of frames captured using the Kinova arm.

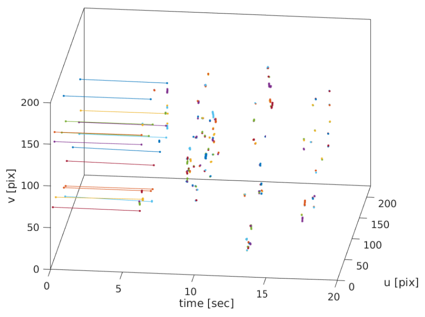
detected, they do generally follow the expected pattern of rotation, as seen in Fig. 5.21.(b).

In a second experiment, we have calibrated the IMU and initialized the pose estimation method with estimated biases of the IMU (see details of IMU calibration in Appendix C). Under these conditions, though not perfect, the estimated rotation much closely follows the real value, as shown in Fig 5.21.(e). One can still see that insufficient visual information still allows some drift.

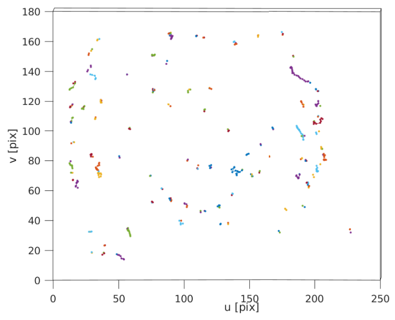
In a third experiment, we took an hybrid approach leveraging the start of the recording, where the camera stays static until around 10 s, and therefore IMU output is mostly noise (and gravity). As such, we start by running the filter considering frames (as if we were using a conventional setup), in order to estimate bias. Assuming this bias is corrected by the end of this interval, we switch to the closed loop approach.

The results obtained using this third approach are presented Fig. 5.21(f), corresponding to a RMSE of 0.3635 deg on the z axis, which is actually quite interesting, though results mostly from a good estimation of the biases from the initial estimation from frames.

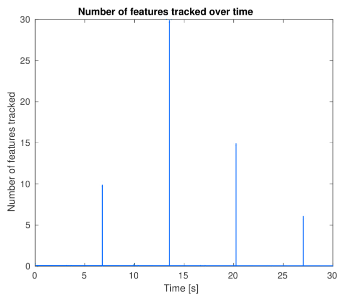
Comparing the results in this section, obtained with a DVS camera, with results in previous sections based on a DAVIS camera, one finds much harder doing events based feature tracking and therefore running a full visual inertial odometry methodology. Our reasoning is that this has to do with the DVS camera itself, which may have been further complicated by the illumination conditions of the lab, as incandescent lights cause unwanted events. The DVS camera is not designed to allow fast switching between frames and events modes. In reality, the frames mode capability is meant in essence just for calibration. As such, there are two problems identified: first is the latency of commutation between modes, which is non-negligible; second, after commutation there is noise in the measurements, in particular, erroneous events are produced. As such, this contaminates the templates for matching and imposes events based tracking difficulties. One possible approach would be to reject first readings, but they are not consistent, as sometimes they are not present, and other times they contaminate the whole frame. This latency and reading corruption is explored in Appendix C.



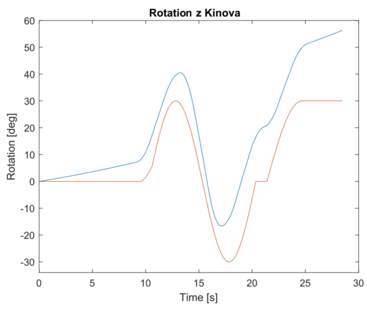
(a) features' position over time



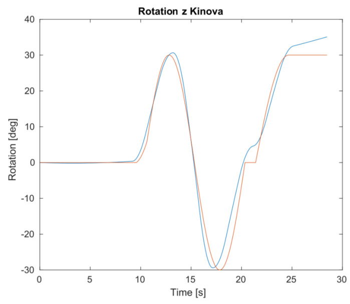
(b) projected features on the camera plane



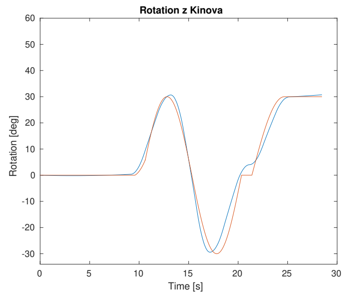
(c) number of event based features over time



(d) estimated rotation and groundtruth (GT)



(e) estimated rotation and GT using pre-estimated IMU bias



(f) estimated rotation and GT using initial image frames

Figure 5.21: Obtained results for experiment 3, showing the features' position over time in (a), their projection to the camera plane in (b), and their overall number over time in (c). The estimated rotation without any correction is presented in (d), as blue, and compared to the groundtruth in orange, showing a clear drift. In (e), initialisation is done using the sensor bias from a prior calibration, and shows an estimated trajectory (blue) following the groundtruth (orange) more closely. In (f), initialization is done by running teh filter using frames, and then switching to the closed loop proposed approach, and shows an estimated trajectory (blue) following the groundtruth (orange) even more closely.

5.6 Results Analysis and Experiments Comparison

Being novel contributions (in particular with novel technology), it is interesting to analyse the strengths and the shortcomings of the proposed methods. In this section we critically analyse the results produced.

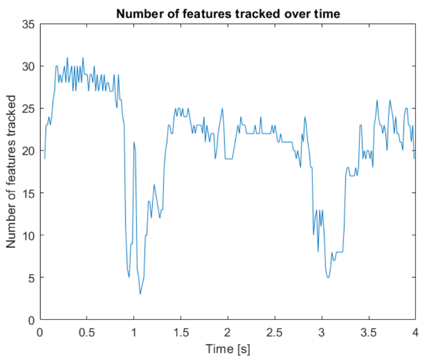
Starting with the comparison between event cameras and conventional cameras, in Table 5.4 we show that event cameras can, in fact, outperform conventional camera approaches, as the error obtained is lower using conventional cameras. Both approaches are "fair" in the sense that the filter being used is equivalent (the same proposed UKF filter). This is because of the more frequent corrections performed by the visual component, which are limited by the frame rate of conventional cameras.

The validation setup was a simulated environment (ESIM). Nevertheless, this simulator is a common tool with event cameras (as public datasets are scarce), which speaks to its validity. One possible argument is that the simulator is biased towards event cameras (which is true), and frames being produced are of low resolution, which would not happen with a conventional camera. Even though this is true, we do not think it would affect the results much, as the number of features being tracked is already a order of magnitude greater than the features being tracked using events. Therefore, we consider that event cameras can, in fact, be used in the context of visual inertial odometry with the proposed approach of UKF and EKLT.

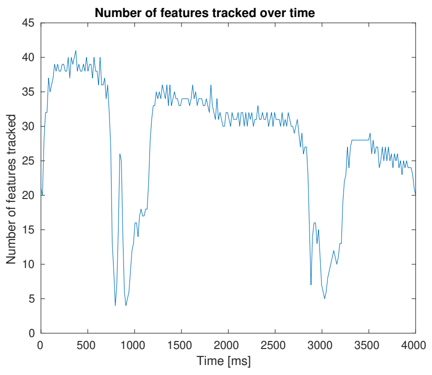
The next comparison that is interesting to be made is between both approaches proposed. The first method is a natural choice considering conventional pose estimation approaches using filtering. A feature extractor feeds visual data into a filter that also uses inertial information. However, in our second method, we propose a feedback in the system so that the tracker can benefit from the current pose estimation. This feedback idea is not exactly novel or innovative, but is nonetheless uncommon. However, its practical usefulness is debatable.

Analysing the results obtained with each method, we can see that generally the closed loop approach performs better (exception on the x axis, by a small margin). Our understanding is that this is because features can be tracked better, and, as a result, more features are available at each instant. Furthermore, their estimated position is more accurate. Fig. 5.22 shows the features being tracked over time by each approach, and the closed loop not only tracks more features, but also tracks them better.

The reason for the tracking of more features is not obvious, and benefits from watching the output of the tracker in real time. What effectively happens is that features are constantly being lost and replaced. The closed loop reduces the rate at which features are lost, which allows for more features being tracked. More features, in turn, help with the estimation.



(a) features using open loop



(b) features using closed loop

Figure 5.22: Comparison of the number of features over time between both proposed approaches.

As such, the closed loop approach, even though initially based on intuition, and the knowledge of the influence of ego motion on the movement of features in camera space, is a valid approach that performs very well under the right conditions.

However, what is not obvious from the results presented is that although the closed loop is generally more precise, it is much less robust. The flow angle estimation is critical for the tracker, and the initial value plays an important role in the success of the tracking. As such, when the state estimation is accurate, the solution performs better (as shown by the higher number of features and generally better results). However, when estimation degrades, it becomes much easier for tracking quality to degrade, and to cause tracking loss.

Another problem with the closed approach is execution time. Though time analysis of either approaches was not considered in this work, it is also worth mentioning the impact of the closed loop (as is implemented). Time performance is very poor. The simulated datasets took around 10 min to process, and the *boxes* dataset was very slow, taking around 30 min for 2 s of data (depending on the rate of events generated at each time). However, this time also comes from the way the approach was implemented, which needed synchronization semaphores and constant messaging between the tracker (implemented in ROS), and the filter (implemented in Matlab). A “proper”, eventually commercial approach could benefit from a single program, as well as a parallel processing of events (which, as is, are sequential).

The next experiment relied only on vision. Though the reason for such an experiment may not seem obvious, the objective was to evaluate the quality of features tracked using events, and their reliability when isolated (without inertial information). *A priori*, based on the previous results, we knew changes in movement corresponded to instants were many features are lost, and that this might pose a difficulty.

Such suspicions were verified, in particular that changes in movement can be unforgiving, as the estimation eventually loses so many features that it drifts away. Visual odometry approaches using conventional cameras are common, but all works with pose estimation and event cameras (that we could find) use inertial information in combination with events. This experiment confirmed the reason. Even with perfect tracking, stops in movement are a major limitation in this approach (though some reconstruction approaches could be pursued).

We then tried real datasets to verify the performance of the approaches on real systems. We tested two environments: high contrast but low texture, and low contrast but high texture. From these experiments we conclude that textured environments help the approach, even though features may be less evident. This conclusion comes from the fact that the *boxes* scene provided slightly better results than the *shapes* scene. Nevertheless, initial results were a bit demotivating, as the first proposed approach was not effective in estimating the correct bias in the x axis, resulting in a drift over time. The closed loop approach, however, produced better results, which are acceptable given the speed of the movement.

Nevertheless, the contrast between the simulations and real datasets is notorious, and begs the question of why such a difference. It is true that the simulation produces a controlled environment to test on, but it is a very realistic scene, not only visually (in terms of frames generated), but also events and inertial information.

The reason has to do with noise and other non-idealities. First, the simulator does not replicate motion blur very well, which means features can always be easily extracted. On the real dataset, when the movement speeds up, this is not the case. Then there is the temporal noise associated with the real system. This type of noise is unfamiliar, and corresponds to incorrectly timestamped events, meaning that they arrive later or earlier, and are processed at an incorrect time. Lastly, there is the matter of salt and pepper noise in the real sensor, which contaminates the template for the tracker and complicates the matching step in EKLT, which is also not accurately present in the simulator. Nevertheless, we consider the experiment to be successful, as estimation followed the correct trend overall, even though it fell short of expectations based on the performance on the simulator.

Lastly, we tested the proposed approaches on a DVS240 camera coupled to a Kinova robot arm. However, ultimately, this experiment proved unsuccessful due to hardware limitations which we were unable to overcome (that might be interesting to analyse in future work) when using both approaches "out of the box", without care. However, when a more careful initialisation is performed, the results become quite interesting. Some additional exploration may be worthwhile, and all software needed for future studies, in terms of dataset acquisition, has been prepared.

Finally, though we consider the general obtained results satisfactory, we would be remiss not to mention other approaches, such as [Zihao Zhu et al., 2017] and [Vidal et al., 2018], which present very positive results. However, they were mostly focused on translations and 6DOF movements, while our experiments focused mainly on rotations, which makes direct comparisons difficult (which is why none were presented in the Experiments sections).

Chapter 6

Conclusion and Future Work

Don't only practice your art, but force your way into its secrets, for it and knowledge can raise men to the divine.

— Ludwig van Beethoven

This chapter concludes the thesis by presenting an overview of the work developed, the methods proposed, and details possible interesting paths for a future exploration. It is divided into a Conclusion section (Section 6.1), and a Future Work section (Section 6.2). We hope the work presented has been interesting for the reader, and that it can be of use for others working on the same subjects.

6.1 Conclusion

In this work we set out to develop a system for pose estimation that was based around event cameras, a novel type of visual sensor that is yet to be fully explored. The motivations for the work were centered around the exploration of event cameras, on the one hand, which were new to the group, and whose potential was still unclear, and the analysis of their viability in the context of the research group's area of interest, on the other hand. In particular, their viability for the estimation of the orientation of the eye, as the movement speed on eye saccades is on the order of hundreds of degrees per second.

This work was the first time using event cameras by members of the group, which entailed some work related to familiarisation of such visual systems. Actually, not many groups work with event cameras, due to their novelty and price[1]. However, exploring such a new technology was quite interesting, as I was able to start from the beginning, almost, and learn the evolution of what had been done. It was particularly amusing seeing the shift in mentality from initial works, focused on image reconstruction and applying established methods in conventional cameras into this new technology, from a new mentality that embraces events as useful in themselves.

In the end, two approaches were developed to tackle this problem. A first, which combined an Unscented Kalman Filter developed around a Lie group structure, with a feature detector and tracker based around events, which performed well under simulated environments, but ultimately under-performed in the real system.

[1]As an anecdote, I often joked with my friends that my event camera had the serial number 206, which meant googling for people with the same problems often proved unfruitful.

The second approach seems even more promising based on simulations results, but its usefulness in real environments is debatable, as the initialisation of the filter is much more critical in this method , and good feature tracking requires good pose estimation. Nevertheless, when said initialisation was done more carefully, the filter performed well.

To validate these approaches, multiple experiments were performed, both with simulated data and real data, and overall they were able to produce decent results. Our methods, focused on the research topics of the group, and at times inspired by biological systems, had multiple innovative concepts in the context of event cameras (the usage of UKF with event cameras, the usage of EKLT for pose estimation, the feedback in the closed loop), as well as some interesting ideas from other fields (such as the UKF with a Lie group state).

Though far from perfect (in fact, both fell short of the current state of the art), both introduced new concepts that can be further improved, and not only produced acceptable results, but served as a basis to understand the current status of event cameras, their limitations and advantages. In a way, it served as a learning experience not only for myself, but for the group as well.

6.2 Future Work

Personally, I believe approaches leveraging only events (or events and IMU) should be pursued, with the objective of letting go of conventional frames, and fully explore the advantages made possible by the use of events, perhaps inspired by the human visual and vestibular pathway. As a small introduction to such an area of exploration, it has been proven that visual processing happens at various levels, and is not exclusively concentrated in the visual region of the brain, located in the occipital cortex (the back of the brain) ([Luo, 2015]). In fact, there is already some processing of information present in the eye, at the level of the retina. Furthermore, before reaching the occipital cortex, neurons synapse at the level of the lateral geniculate nucleus, located in the thalamus, which already extracts information such as blobs, edges, corners and moving objects (and is, in fact, the inspiration for the initial operations performed in Convolutional Neural Networks (CNN)). Such concepts should be interesting to adapt to event cameras.

Image reconstruction is also a possible interesting path to take, as, if perfected to a quality similar to conventional frames, could be comparable to high-speed cameras, which would allow for more frequent updates of the filter (that we showed benefited estimation). It would be interesting seeing said reconstructed frames being used in the current state of the art pose estimators based on conventional cameras. Nevertheless, such strategies are not trivial, and have been pursued since the beginning of event cameras, with varying degrees of results.

Lastly, as machine learning and neural network approaches seem to be ubiquitous, considering such an approach might be interesting. However, depending on the type of data being used (for example, if direct events are to be used), work on Spiking Neural Networks (SNN) may be needed, as the asynchronous nature of events is best captured by the asynchronous nature of SNN.

Appendix A

Lie Groups and the Unscented Kalman Filter

A.1 Lie groups and Lie Algebra

This section explains the concept of Algebraic groups, in particular Lie groups, which are fundamental to understand the structure of the filter in 3.4. This section was written based on the works by [Wang and Chirikjian, 2008], [Müller, 2017], and [Deray and Solà, 2020].

Algebraic groups An algebraic group is an algebraic structure with no singularities (holes or points), with its corresponding operation set. Considering group $(A, *)$, where A is a set and $*$ is the group operator, the following properties must hold for a structure to be a group: Closure, $\forall a_1, a_2 \in A, a_1 * a_2 \in A$; Associativity, $\forall a_1, a_2, a_3 \in A, (a_1 * a_2) * a_3 = a_1 * (a_2 * a_3)$; Identity, $\exists a_0 \in A, s.t. \forall a \in A, a_0 * a = a * a_0 = a$; Inverse, $\forall a \in A, \exists a^{-1} \in A, s.t., a * a^{-1} = a_0$. A Lie group is an algebraic group that is a smooth differentiable manifold. Furthermore, the operator and the inversion are smooth functions.

Lie algebra Each Lie group has an associated Lie algebra, corresponding to the tangent vectorspace around the identity element of the group, which means that the Lie algebra is a vector space obtained by differentiating the group at the identity transformation. As such, the Lie group representation is desirable when we want to represent differential quantities pertaining to the group, such as velocity and covariance, which are well-represented in the tangent space around the transformation, in particular because we can convert any element of the tangent space exactly into a transformation in the group through the exponential map, and the adjoint transforms tangent vectors from one tangent space to another.

Exponential and logarithmic maps The exponential and logarithmic map allows the transfer of elements between the Lie group and the corresponding Lie algebra. The exponential map locally maps an element of the tangent space to the group, and the logarithmic map of a Lie group provides the "inverse operation", transferring elements from the Lie group to its tangent space.

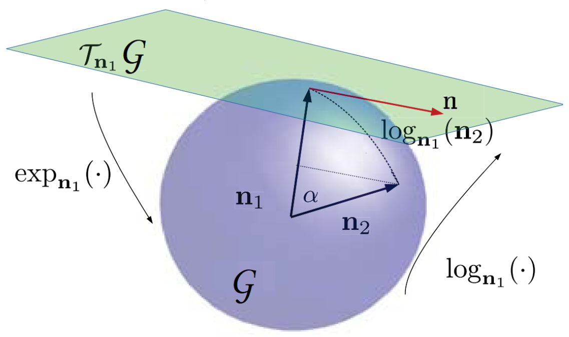

$\mathcal{T}_{\mathbf{n}_1}\mathcal{G}$
$\mathbf{n}$
$\log_{\mathbf{n}_1}(\mathbf{n}_2)$
$\exp_{\mathbf{n}_1}(\cdot)$
$\mathbf{n}_1$
α
$\mathbf{n}_2$
$\log_{\mathbf{n}_1}(\cdot)$
$\mathcal{G}$

Figure A.1: Representation of the Lie group $\mathcal{G}$ and its corresponding Lie algebra (tangent space) around n_1 denoted $\mathcal{T}_{n_1}\mathcal{G}$. The exp operation takes elements from the tangent space and maps them to the group, and the log operation takes elements from the group and maps them to the tangent space. Adapted from [Li et al., 2018].

Consider Fig. A.1. Here we represent group $\mathcal{G}$ with elements of the group n_1 and n_2. The tangent space considered at n_1, denoted $\mathcal{T}_{n_1}\mathcal{G}$ is also represented. The relevance of the exp and log operations is evident in the figure, as we can see that n_2 can be mapped onto the tangent space by

$$n = \log_{n_1}(n_2) = \log\left(n_1^{-1}n_2\right) \tag{A.1}$$

and symmetrically, the element n can be mapped onto the group through

$$n_2 = \exp_{n_1}(n) = n_1 \exp(n) \tag{A.2}$$

Particular Lie groups The Lie group theory introduced so far is very abstract. However, some groups are of particular interest to the area of Robotics, such as the Special Orthogonal Group (SO) (Section A.1), useful to represent rotations, and the Special Euclidean Group (SE) (Section A.1), relevant when representing rigid body transformations. We will briefly talk about these two groups.

The Special Orthogonal Group $SO(3)$ The rotation group $SO(3)$, with its corresponding Lie algebra $\mathfrak{so}(3)$, and vector space $\mathbb{R}^3$. This group is formed by the set of orthogonal rotation matrices $\mathbf{R} \in \mathbb{R}^{3\times3}$ that fulfil

$$SO(3) := \left\{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}^T\mathbf{R} = \mathbf{I}_3, \det(\mathbf{R}) = 1\right\}. \tag{A.3}$$

From the orthogonality condition $\mathbf{R}^T\mathbf{R} = \mathbf{I}_3$, the tangent space may be found by derivating with respect to time, in particular $\mathbf{R}^T\dot{\mathbf{R}} + \dot{\mathbf{R}}^T\mathbf{R} = 0$, that can be rearranged to form

$$\mathbf{R}^T\dot{\mathbf{R}} = -\left(\mathbf{R}^T\dot{\mathbf{R}}\right)^T, \tag{A.4}$$

which reveals that $\mathbf{R}^T\dot{\mathbf{R}}$ is, in fact, a skew-symmetric matrix, which are usually represented by the hat operator $(.)^\wedge$

$$\hat{\omega} = [\omega]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad \in \quad \mathfrak{so}(3) \tag{A.5}$$

where $\omega = [\omega_x \omega_y \omega_z]^T \in \mathbb{R}^3$. Since $[\omega]_\times \in \mathfrak{so}(3)$ has 3 degrees of freedom, the Lie algebra can be decomposed into the components

$$[\omega]_\times = \omega_x \mathbf{E}_x + \omega_y \mathbf{E}_y + \omega_z \mathbf{E}_z \tag{A.6}$$

$$\mathbf{E}_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{E}_y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \mathbf{E}_z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{A.7}$$

This relation allows us to map transformations between $\mathfrak{so}(3)$ and $\mathbb{R}^3$, by use of the operators hat and vee:

$$\text{Hat:} \quad \mathbb{R}^3 \longrightarrow \mathfrak{so}(3); \quad \omega \mapsto \hat{\omega} = [\omega]_\times \tag{A.8}$$

$$\text{Vee:} \quad \mathfrak{so}(3) \longrightarrow \mathbb{R}^3; \quad [\omega]_\times \mapsto [\omega]_\times^v = \omega \tag{A.9}$$

The exponential mapping for $SO(3)$ can be obtained by means of the Rodrigues' formula ([Mebius, 2007]):

$$\mathbf{R} = \exp\left([\omega]_\times\right) = \mathbf{I}_3 + \frac{\sin\left(\|\omega\|\right)}{\|\omega\|} + \frac{\left(1 - \cos\left(\|\omega\|\right)\right)}{\|\omega\|^2} \left(\hat{\omega}\right)^2 \tag{A.10}$$

where $\|\omega\| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$. The logarithmic mapping can also be obtained by inverting (A.10):

$$\theta = \begin{cases} \cos^{-1}\left(\dfrac{\text{tr}(\mathbf{R} - 1)}{2}\right) & \text{if} \quad \mathbf{R} = \mathbf{I}_3 \\ 2\pi k & \text{if} \quad \mathbf{R} \neq \mathbf{I}_3 \end{cases} \tag{A.11}$$

$$\hat{\omega} = \begin{cases} \dfrac{\theta}{2\sin(\theta)} \left(\mathbf{R} - \mathbf{R}^T\right) & \text{if} \quad \mathbf{R} = \mathbf{I}_3 \\ 0 & \text{if} \quad \mathbf{R} \neq \mathbf{I}_3 \end{cases} \tag{A.12}$$

where ω gives the direction of rotation, and θ gives the angle of rotation.

The Special Euclidean Group $SE(3)$ This group is defined as

$$SE(3) := \left\{ \mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \| \mathbf{R} \in SO(3), \mathbf{p} \in \mathbf{R}^3 \right\} \tag{A.13}$$

This group can be considered to represent rigid body transformations, and $\mathbf{H}$ is referred to a homogeneous transformation matrix, where $\mathbf{R}$ provides the rotation information (refer to Section A.1), and $\mathbf{p}$ refers to the translation information, hence the rigid body transformation.

The associated Lie algebra is represented as $\mathfrak{se}(3)$, consists of the 4×4 matrices of the form

$$\begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \quad \in \quad \mathfrak{se}(3) \tag{A.14}$$

where $\hat{\omega} \in \mathfrak{so}(3)$ and $v \in \mathbb{R}^3$. Considering a robotic system, ω usually refers to the angular velocity of the system, and v its linear velocity. The two components can be grouped together in a vector $\mathbf{v} = \begin{bmatrix} v \\ \omega \end{bmatrix}$ in what is called a twist.

The hat operator is applied as

$$\hat{\mathbf{v}} = \begin{bmatrix} v \\ \omega \end{bmatrix}^{\wedge} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \quad \in \mathfrak{se}(3) \tag{A.15}$$

Furthermore, the inverse of an element $\mathbf{H}$ is given by

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{p} \\ 0 & 1 \end{bmatrix} \tag{A.16}$$

The Lie group in this work In this work, part of the state being estimated by the filter is given by the Lie group $\chi \in SE_{2+p}(3)$ that incorporates the orientation $\mathbf{R} \in SO(3)$, velocity $\mathbf{v} \in \mathbb{R}^3$ and position $\mathbf{x} \in \mathbb{R}^3$, as well as the 3D positions of the landmarks $\mathbf{p}_1, \ldots, \mathbf{p}_p \in \mathbb{R}^3$:

$$\chi = \begin{bmatrix} \mathbf{R} & \mathbf{v} & \mathbf{x} & \mathbf{p}_1 \cdots \mathbf{p}_p \\ 0_{(p+2)\times 3} & & I_{(p+2)\times(p+2)} & \end{bmatrix}_{(5+p)\times(5+p)} . \tag{A.17}$$

This group can be seen as an extension to $SE(3)$, where we include the velocity of the state, as well as the landmarks' 3D position.

A.2 Sensor Fusion and Filtering

It is common to have multiple sensors in a system that produce complementary or redundant readings of the world and/or the state of the system. This may be for several reasons, such as sensor noise, sampling rate, and robustness, among others. For example, one sensor may report information on velocity, whereas another reports on position. These two quantities are related through movement equations, and can be considered redundant, but by combining different types of readings, global uncertainties on the system can be reduced, and limitations of the single sensor may be overcome.

However, how to properly use and fuse the readings from multiple sensors is itself a critical part. In this work, the event camera was used in conjunction with an IMU, and so fusing the visual information with inertial information was needed. The solution used was an Unscented Kalman Filter (UKF) (explained in Section A.2), which is described in Section 3.4.3. The UKF is an extension to the Kalman Filter (explained in Section A.2) applicable to nonlinear systems.

Kalman Filter The Kalman Filter [Kalman, 1960] is an algorithm that can be applied when we have a linear system that follows the model equations (the system equation)

$$x(k+1) = Fx(k) + Bu(k) + Gv(k) \tag{A.18}$$

and (the measurement equation)

$$z(k) = Hx(k) + w(k) \tag{A.19}$$

namely a linear system that can be represented with a state s that evolves according to Eq. (A.18) and that can be measured by an observer function (A.19) that provides z, that can be somehow related linearly to the state. $v(k)$ is the state noise process, $u(k)$ is the input to the system, and $w(k)$ is the measurement noise.

Both noises are assumed to be zero-mean Gaussian white noise, meaning

$$E(w(k)) = E(v(k)) = 0 \tag{A.20}$$

$$\left[\left(\begin{smallmatrix} w(k) \\ v(k) \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} w(m)^T & v(m)^T \end{smallmatrix}\right)\right] = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix} \delta(k-m) \quad . \tag{A.21}$$

Furthermore, $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^m$, $w(k) \in \mathbb{R}^n$, $v(k) \in \mathbb{R}^q$, and $y(k) \in \mathbb{R}^q$.

With this formulation, the optimal estimator is given by a set of equations that relate to the prediction of the state based on the input to the system, in particular

$$\hat{x}(k+1|k) = F\hat{x}(k|k) + Bu(k) \tag{A.22}$$

$$P(k+1|k) = FP(k|k)F^T + GR_1G^T \tag{A.23}$$

and another set related to the update and correction of the estimate, namely

$$K(k) = P(k|k-1)H^T \left[HP(k|k-1)H^T + R_2\right]^{-1} \tag{A.24}$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k)\left[z(k) - H\hat{x}(k|k-1)\right] \tag{A.25}$$

$$P(k|k) = \left[I - K(k)H\right]P(k|k-1) \tag{A.26}$$

provided by the measurement.

The Kalman Filter, though optimal, is only applicable to linear systems. As such, alternatives are needed when nonlinear systems are used (which arise particularly in the estimation of rotation in the case of this work). Two extensions to the Kalman Filter are usually considered: the Extended Kalman Filter (EKF) (presented in Section A.2) and the Unscented Kalman Filter (UKF) (presented in Section A.2).

Extended Kalman Filter (EKF) The Extended Kalman Filter (EKF) is an extension to the Kalman Filter when the system or the measurements from the system are nonlinear, such as

$$x(k+1) = f(x(k), u(k), v(k)) \tag{A.27}$$

$$z(k) = h(x(k), u(k)) + w(k) \tag{A.28}$$

where functions f and/or h are nonlinear functions. In this case, the system cannot be written in matrix notation (such as (A.18) and (A.19)).

As such, there are no F or H matrices that can be used for uncertainty propagation ((A.23) and (A.26)), since the system is nonlinear, and therefore functions f and/or h cannot be written as a linear combination of the input variables. Furthermore, due to the nonlinear nature of the system, the covariance P would not preserve the Gaussian nature of the noise.

EKF solves this problem by linearizing the system dynamics around the predicted and filtered estimates of the state, at each cycle:

$$F_{k+1} = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k+1|k}, u_k} \tag{A.29}$$

$$H_{k+1} = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k+1|k}} \tag{A.30}$$

so that a matrix F_{k+1} and H_{k+1} are generated at each cycle, suitable for use in (A.23) and (A.26), as if the system was linear.

Clearly, this filter is sub-optimal, as it does not contemplate the full system model, as well as the effect of the noise on the system. Moreover, the need for the linearization of the model can be cumbersome for large systems.

Nevertheless, the EKF is a simple extension to the Kalman Filter that allows its usage with nonlinear systems, and generally obtains positive results.

Unscented Kalman Filter (UKF) The inherent flaws of the EKF stem from its linearization approach for calculating the mean and covariance of a random variable. The Unscented Kalman Filter (UKF) ([Julier and Uhlmann, 1997]) addresses these flaws. It is another extension to the Kalman Filter when the system or the measurements from the system are nonlinear:

$$x(k+1) = f(x(k), u(k), v(k)) \tag{A.31}$$

$$z(k) = h(x(k), u(k)) + w(k) \quad . \tag{A.32}$$

This situation is similar to the one in the EKF (Section A.2).

Unlike the Extended Kalman Filter (EKF, Section A.2), which can also be used with nonlinear functions, the UKF does not linearize these functions when propagating the uncertainty, and instead tries to approximate distribution of the output to a Gaussian distribution by using an unscented transform. This procedure works by choosing a set of sigma points ($2L+1$ points, where L is the state dimension, corresponding to 2 points around the current estimate for each dimension in the state vector, plus the mean) which are subject to the true nonlinear transformation and used to approximate the output distribution, as shown in Fig. A.2. The comparison of approached between the EKF and the UKF is shown in Fig. A.3, in particular the propagation of the state, in which EKF linearizes the system, whereas the UKF takes (in this case, 5) sigma points to obtain the propagated mean and covariance of the state. Notice this approach usually produces a "truer" result, accurate to the third order of the Taylor series expansion for Gaussian inputs, and at least second order for non-Gaussian inputs ([Van Der Merwe and Wan, 2001]).

These sigma points are given by

$$\chi_0 = \bar{x}, W_0 = \kappa/(L+\kappa) \tag{A.33}$$

$$\chi_i = \bar{x} + \left(\sqrt{(L+\kappa)P_{xx}}\right), W_i = 1/2(L+\kappa) \tag{A.34}$$

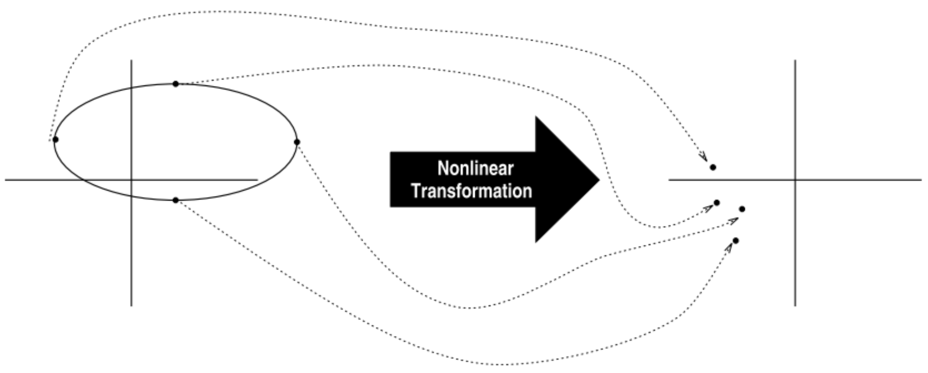


Figure A.2: The principle of the Unscented Transform, where sigma points obtained by varying each dimension is the state space is propagated using the nonlinear model to approximate the distribution of the covariance, from [Julier and Uhlmann, 1997]

$$\chi_{i+L} = \bar{x} - \left(\sqrt{(L+\kappa)P_{xx}}\right), W_{i+L} = 1/2(L+\kappa) \tag{A.35}$$

with corresponding weights W_i associated to each point, where $L \in \mathbb{R}$ denotes the dimension of the state and $\kappa \in \mathbb{R}$ is a scaling parameter to control the spread of the sigma points. Furthermore, $\bar{x}$ represents the mean of state x, P_{xx} its covariance, and $i \in [1, 2n]$.

The propagation of the sigma points is then given

$$y_i = f(\chi_i) \tag{A.36}$$

$$\bar{y} = \sum_{i=0}^{2L} W_i y_i \tag{A.37}$$

$$P_{yy} = \sum_{i=0}^{2L} (y_i - \bar{y})(y_i - \bar{y})^T \quad . \tag{A.38}$$

This transformation allows us to have the mean and covariance associated with the nonlinear process, without the need to linearize it beforehand, producing a better approximation as well.

The full algorithm is given by Eqs. (A.39) through (A.50).

By using the UKF, it is not needed to linearize the system equations, which results in a better approximation of the uncertainty of the estimate, since the nonlinearities of the system are somewhat taken into account. Furthermore, the use of the unscented transform also makes the filter more robust to noise ([Wan and Van Der Merwe, 2000]).

Square-Root Unscented Kalman Filter (SR-UKF) The most computationally expensive operation in the UKF is the generation of the new set of sigma points at each time update, that requires taking the matrix square root of the covariance matrix $P \in \mathbb{R}^{L\times L}$, which can be given by $P = SS^T$. Efficient implementations are usually $\mathcal{O}\left(L^3/6\right)$. However, by propagating S directly, there is no no need to refactorize at each step. This is the idea at the heart of the SR-UKF. Though the complexity remains similar, but the numerical properties improve.

SR-UKF relies on three techniques:

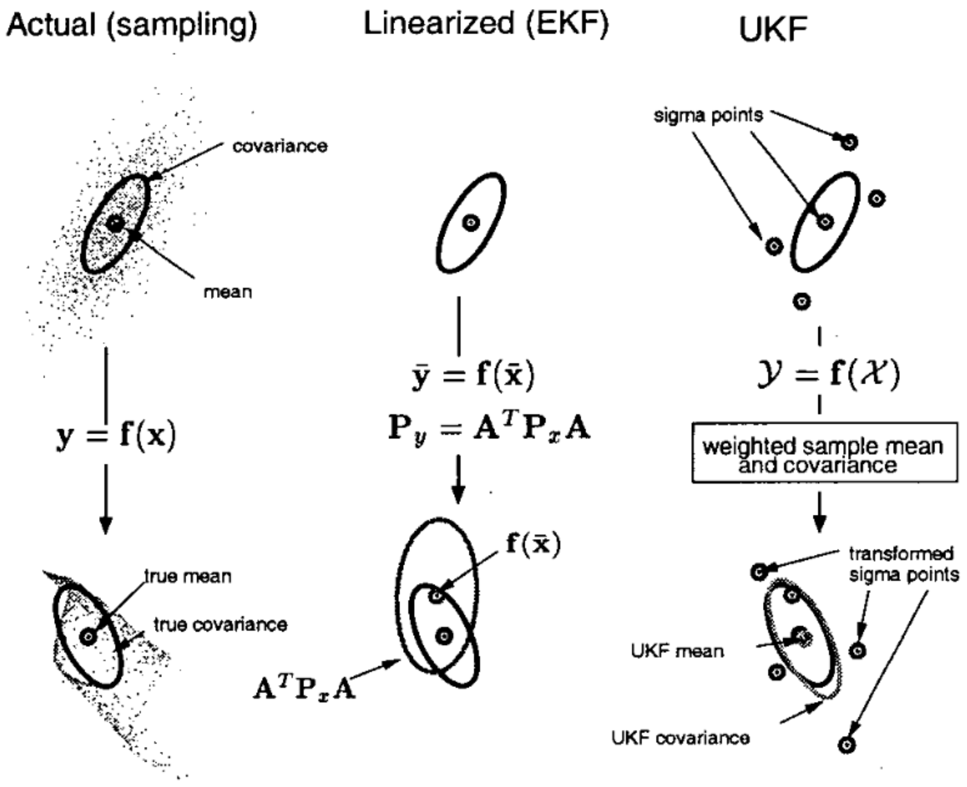

Actual (sampling)
Linearized (EKF)
UKF
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UKF mean
UKF covariance

Figure A.3: Comparison of propagation between EKF and UKF. Left shows the true mean and covariance; middle represents the EKF approach, linearizing the system; right shows the UKF approach, generating sigma points (in this case, 5) to propagate the state and obtain estimation on the mean and covariance, from [Van Der Merwe and Wan, 2001]

- **QR decomposition** The QR decomposition is factorization of a matrix $A \in \mathbb{R}^{L\times N}$ such that $A^T = QR$, where $Q \in \mathbb{R}^{N\times N}$ and $R \in \mathbb{R}^{N\times L}$ is upper triangular, and $N \geq L$. The upper triangular part of R, denoted $\bar{R}$, is the transpose of the Cholesky factor, so we have that $\bar{R} = S^T$ and $P = AA^T = \bar{R}^T\bar{R}$. qr$\{.\}$ denotes the QR decomposition that returns R. The computational complexity is $\mathcal{O}\left(NL^2\right)$, as opposed to the Cholesky decomposition that is $\mathcal{O}\left(L^3/6\right)$, plus $\mathcal{O}\left(L^2\right)$ to form AA^T.

- **Cholesky factor updating** Assuming S is the Cholesky factor of $P = AA^T$, the Cholesky update corresponds do $P \pm \sqrt{\nu}uu^T$, and is denoted as $S = \text{cholupdate}\{S, u, \pm\nu\}$. The complexity is $\mathcal{O}\left(L^2\right)$ per update.

- **Efficient least squares** The solution to $(AA^T)x = A^Tb$ is also the solution to $Ax = b$, which can be solved efficiently using QR decomposition with pivoting.

The full algorithm is given by Eqs. (A.51) through (A.65).

UKF algorithm

Initialize with:

$$\bar{x}_0 = \mathbb{E}\left[x_0\right] \quad P_0 = \mathbb{E}\left[\left(x_0 - \bar{x}_0\right)\left(x_0 - \bar{x}_0\right)^T\right] \tag{A.39}$$

For $k \in \{1, \cdots, +\infty\}$
Calculate sigma points:

$$\chi_{k-1} = \left[\hat{x}_{k-1} \quad \hat{x}_{k-1} + \eta\sqrt{P_{k-1}} \quad \hat{x}_{k-1}\hat{x}_{k-1} - \eta\sqrt{P_{k-1}}\right] \tag{A.40}$$

Time update:

$$\chi_{k|k-1} = F\left[\chi_{k-1}, u_{k-1}\right] \tag{A.41}$$

$$\hat{x}_k^- = \sum_{i=0}^{2L} W_i \chi_{i,k|k-1} \tag{A.42}$$

$$P_k^- = \sum_{i=0}^{2L} W_i \left[\chi_{i,k|k-1} - \hat{x}_k^-\right]\left[\chi_{i,k|k-1} - \hat{x}_k^-\right]^T + R^v \tag{A.43}$$

$$y_{k|k-1} = H\left[\chi_{k|k-1}\right] \tag{A.44}$$

$$\hat{y}_k^- = \sum_{i=0}^{2L} W_i y_{i,k|k-1} \tag{A.45}$$

Measurement update equations:

$$P_{y_k y_k} = \sum_{i=0}^{2L} W_i \left[y_{i,k|k-1} - \hat{y}_k^-\right]\left[y_{i,k|k-1} - \hat{y}_k^-\right]^T + R^n \tag{A.46}$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i \left[\chi_{i,k|k-1} - \hat{x}_k^-\right]\left[y_{i,k|k-1} - \hat{y}_k^-\right]^T \tag{A.47}$$

$$\mathcal{K}_k = P_{x_k y_k} P_{x_k y_k}^{-1} \tag{A.48}$$

$$\hat{x}_k = \hat{x}_k^- + \mathcal{K}_k\left(y_k - \hat{y}_k^-\right) \tag{A.49}$$

$$P_k = P_k^- - \mathcal{K}_k P_{y_k y_k} \mathcal{K}_K^T \tag{A.50}$$

where R^v is process noise covariance and R^n is measurement noise covariance

SR-UKF algorithm

Initialize with:

$$\bar{x}_0 = \mathbb{E}\left[x_0\right] \quad P_0 = \text{chol}\left\{\mathbb{E}\left[\left(x_0 - \bar{x}_0\right)\left(x_0 - \bar{x}_0\right)^T\right]\right\} \tag{A.51}$$

For $k \in \{1, \cdots, +\infty\}$

Calculate sigma points and time update:

$$\chi_{k-1} = \left[\hat{x}_{k-1} \quad \hat{x}_{k-1} + \eta S_k \quad \hat{x}_{k-1}\hat{x}_{k-1} - \eta S_k\right] \tag{A.52}$$

$$\chi_{k|k-1} = F\left[\chi_{k-1}, u_{k-1}\right] \tag{A.53}$$

$$\hat{x}_k^- = \sum_{i=0}^{2L} W_i \chi_{i,k|k-1} \tag{A.54}$$

$$S_k^- = \text{qr}\left\{\left[\sqrt{W_1}\left(\chi_{1:2L,k|k_1} - \hat{x}_k^-\right) \quad \sqrt{R^v}\right]\right\} \tag{A.55}$$

$$S_k^- = \text{cholupdate}\left\{S_k^-, \chi_{0,k} - \hat{x}_k^-, W_0\right\} \tag{A.56}$$

$$y_{k|k-1} = H\left[\chi_{k|k-1}\right] \tag{A.57}$$

$$\hat{y}_k^- = \sum_{i=0}^{2L} W_i y_{i,k|k-1} \tag{A.58}$$

Measurement update equations:

$$S_{y_k} = \text{qr}\left\{\left[\sqrt{W_1}\left[y_{1:2L,k} - \hat{y}_k\right]\sqrt{R_n^n}\right]\right\} \tag{A.59}$$

$$S_{y_k} = \text{cholupdate}\left\{S_{y_k}, y_{0,k} - \hat{y}_k, W_0\right\} \tag{A.60}$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i\left[\chi_{i,k|k-1} - \hat{x}_k^-\right]\left[y_{i,k|k-1} - \hat{y}_k^-\right]^T \tag{A.61}$$

$$\mathcal{K}_k = P_{x_k y_k} P_{x_k y_k}^{-1} \tag{A.62}$$

$$\hat{x}_k = \hat{x}_k^- + \mathcal{K}_k\left(y_k - \hat{y}_k^-\right) \tag{A.63}$$

$$U = \mathcal{K}_k S_{y_k} \tag{A.64}$$

$$S_k = \text{cholupdate}\left\{S_k^-, U, -1\right\} \tag{A.65}$$

where R^v is process noise covariance and R^n is measurement noise covariance

Appendix B

Computing Optical Flow given the Ego-motion

The aim of this appendix is to explain the assumptions and derive the equations between optical flow and the effect of ego-motion (the movement of the camera) on optical flow.

Derivation of the Brightness Constancy Equation How do we go from $I\left(x + u\delta t, y + v\delta t, t + \delta t\right) = I(x, y, t)$ to

$$\frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \quad ? \tag{B.1}$$

The key step is considering that if the time step is small, we can linearize the intensity function using the Multivariable Taylor Series Expansion into

$$I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = I(x, y, t) \tag{B.2}$$

which yields

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \quad . \tag{B.3}$$

Dividing both sides by δt, and taking the limit $\delta t \to 0$, we obtain

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{B.4}$$

which corresponds to the Brightness Constancy Equation.

This constraint direct application provides only *normal optical flow*, not the complete optical flow (usually called just *optical flow*). Various ways exist to approximate optical flow, as using higher order derivatives or tracking feature points (using KLT, for example, as detailed in Chapter 2.2.2.3). In the following is considered just the optical flow, $(\dot{x}, \dot{y})$.

Effect of Ego-motion on Optical Flow We begin with the equation that relates the movement of a 3D point P in the scene as seen from the camera:

$$\dot{P} = -T - \omega \times P = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = -\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} - \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{B.5}$$

and the projection equation (and its derivative)

$$\begin{cases} x = f\dfrac{X}{Z} \\ y = f\dfrac{Y}{Z} \end{cases} \Rightarrow \begin{cases} \dot{x} = f\dfrac{Z\dot{X} - X\dot{Z}}{Z^2} \\ \dot{y} = f\dfrac{Z\dot{Y} - Y\dot{Z}}{Z^2} \end{cases} . \tag{B.6}$$

Starting with (B.6), we can rewrite it as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{f}{Z}\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} - \frac{f}{Z}\begin{bmatrix} X \\ Y \end{bmatrix}\frac{\dot{Z}}{Z} . \tag{B.7}$$

Now, expanding (B.5):

$$\begin{cases} \dot{X} = -\left(T_x + \omega_y Z - \omega_z Y\right) \\ \dot{Y} = -\left(T_y + \omega_z X - \omega_x Z\right) \\ \dot{Z} = -\left(T_z + \omega_x Y - \omega_y X\right) \end{cases} \tag{B.8}$$

we can isolate and manipulate into

$$\frac{f}{Z} = -\frac{f}{Z}\begin{bmatrix} T_x + \omega_y Z - \omega_z Y \\ T_y + \omega_z X - \omega_x Z \end{bmatrix} = \frac{f}{Z}\begin{bmatrix} T_x \\ T_y \end{bmatrix} - \begin{bmatrix} \omega_y f - \omega_z y \\ -\omega_x f + \omega_z x \end{bmatrix} \tag{B.9}$$

and

$$\begin{aligned} \frac{f}{Z}\begin{bmatrix} X \\ Y \end{bmatrix}\frac{\dot{Z}}{Z} &= -\begin{bmatrix} x \\ y \end{bmatrix}\frac{1}{Z}\left(T_z + \omega_x Y - \omega_y X\right) = -\begin{bmatrix} x \\ y \end{bmatrix}\left(\frac{T_z}{Z} + \omega_x\frac{y}{f} - \omega_y\frac{x}{f}\right) \\ &= \begin{bmatrix} x \\ y \end{bmatrix}\left(\frac{T_z}{Z}\right) - \begin{bmatrix} x \\ y \end{bmatrix}\left(\omega_x\frac{y}{f} - \omega_y\frac{x}{f}\right) \end{aligned} \tag{B.10}$$

Finally, we obtain the equation for the motion field due to ego-motion

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{f}{Z}\begin{bmatrix} -T_x + \frac{x}{f}T_z \\ -T_y + \frac{y}{f}T_z \end{bmatrix} + \begin{bmatrix} \omega_x\frac{xy}{f} - \omega_y\left(f + \frac{x^2}{f}\right) + \omega_z y \\ \omega_x\left(f + \frac{y^2}{f}\right) - \omega_y\frac{xy}{f} + \omega_z x \end{bmatrix} . \tag{B.11}$$

Appendix C

Camera Calibration, Latency and IMU Calibration

Camera Calibration The image obtained from a camera (either conventional or neuromorphic) results from the transformation of 3D points in the world to 2D points in the camera plane, and is constrained by the physical properties of the camera. Such properties include unwanted distortions relating to the lens' geometry, as well as unalignment between the lens and the camera sensor.

The goal of the calibration is to estimate the intrinsic (parameters relating to the camera itself, such as focal length, optical centre, and skew coefficient, which are fixed), extrinsic (parameters external to the camera, specifically translations and rotations between the camera and the world) and distortion (relating to the lens) parameters of the camera. A calibrated camera is needed for computer vision algorithms, as the relation between points in the image and world frames needs to be known for distance estimation, 3D reconstruction, depth estimation, ...

Distortion can be modelled as tangential and radial distortion. Radial distortion is caused by the elliptical geometry of the lens, as light rays bend more near the edges than at the optical centre. Smaller lenses cause greater distortion. It is possible to have three types of radial distortion, namely negative, none, and positive, as shown in Fig. C.1. Tangential distortion occurs when the lens is not perfectly parallel to the sensor, which happens during manufacture, as shown in Fig. C.1.

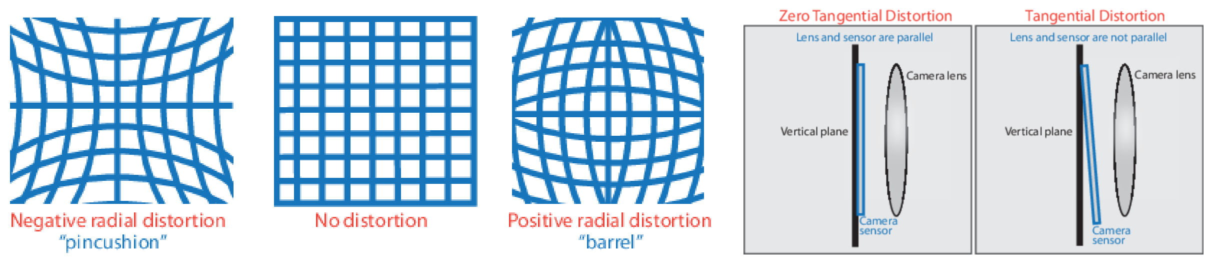

Negative radial distortion
"pincushion"
No distortion
Positive radial distortion
"barrel"
Zero Tangential Distortion
Lens and sensor are parallel
Camera lens
Vertical plane
Camera sensor
Tangential Distortion
Lens and sensor are not parallel
Camera lens
Vertical plane
Camera sensor

Figure C.1: Comparison of possible radial and tangential distortions

The process of finding the matrix K for a given camera is known as camera calibration. The calibration problem can be formulated as an optimization problem that matches 3D reference positions in the world (which can be either known (Direct Linear Transformation) or unknown), and the corresponding projections in the camera plane. We can assume centred image and world frames (ignore extrinsic parameters), and focus solely on intrinsic parameters, resulting in a total of 5 unknowns.

An example of calibration with known features is Direct Linear Transformation (DLT) [Tsai, 1987], in which we know the exact location of points in the 3D world and their corresponding image projection in the camera plane (as in Figs. 2.1 and 2.2, assuming that the scene/world point is known), and through linear equations that result from the mapping of multiple points through the camera matrix, in particular the projection equations (2.3) and (2.4), we can estimate the parameters. Though this technique provides good results, a very careful setup is needed, as greater precision leads to better parameter estimation. This is not always possible, or very practical (as the 3D coordinates of the points need to be known), which is why more flexible were proposed.

A popular technique is that of [Zhang, 2000], which relies on a checkerboard planar pattern (such as the one in Fig C.2) captured from at least two orientations, but usually more. The high contrast of the checkerboard pattern allow for easy detection of edges and corners, as well as the plane of the checkerboard. A typical workflow consists of capturing a few images of the checkerboard under different orientations, by either moving the camera or the checkerboard, then detect feature points in the images and use a closed form solution to match 3D and 2D points, and obtain an estimation of the parameters. Afterwards, an energy optimization based on the maximum-likelihood criterion is used to fine-tune the parameters.

Event camera calibration Event cameras follow the same optical principles described in Section 2.1.1, meaning the same models are still valid, and suffer from the same distortion parameters that need to be quantified through calibration. However, due to the nature of event cameras, the techniques from Section C cannot be applied directly. The static images of the checkerboard that are used in conventional camera calibration would just be blank images (because event cameras need movement or changes to produce output). Nevertheless, completely redesigning the last decades of computer vision calibration techniques is ill-advisable, and common techniques for event cameras merely replace the steps up until feature acquisition.

Event camera calibration is a two-stage procedure. First, a sharp image (such as the ones in Fig. C.2) is used, in order to focus the lenses (focus adjustment). With this procedure, the edges produced from moving the camera or the pattern should be sharp.

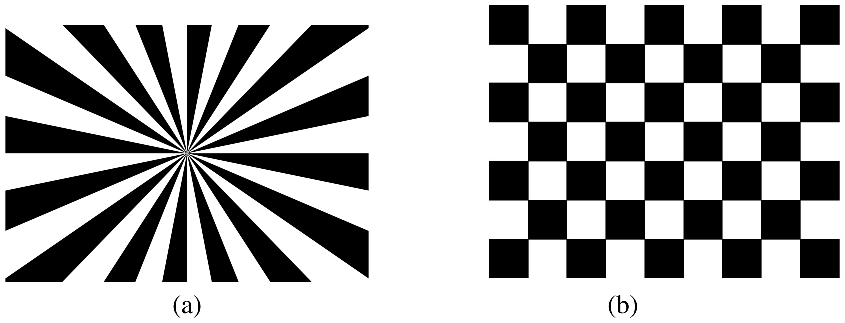
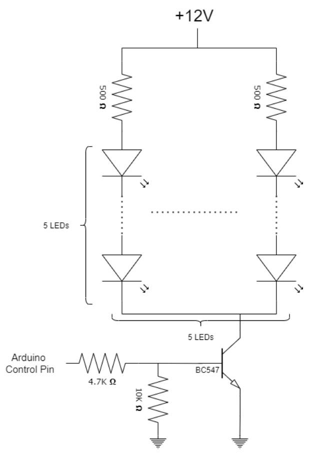

(a) (b)

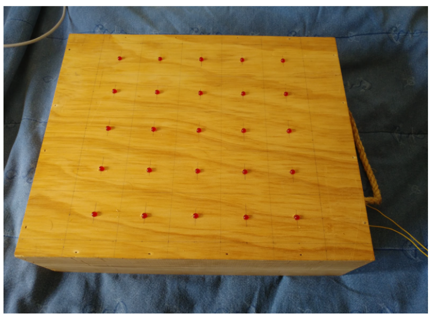
Figure C.2: Example images used for focus adjustment

Afterwards, a blinking LED pattern is used (Fig. C.3). This pattern attempts to mimic the checkerboard of conventional calibration, and the LEDs act as the corners of the checkerboard. Usually, a grid of 5x5 LEDs, spaced 5 cm between themselves, and blinking at a frequency of 500 Hz is used.

Since the LEDs are blinking, the event camera is able to detect them and generate events, even with a still



(a)



(b)

Figure C.3: (a) Schematic used for the connection of the LEDs, and (b) LED pattern in a rigid surface for calibration

environment and camera. These events (from the blinking) are accumulated over a period of time in order to create a pseudo-frame, and blob detection is used to provide these feature to the pipeline of conventional camera calibration, which would use the corners as features. From here, the calibration pipeline is preserved.

Newer generation neuromorphic cameras combine both event and conventional camera (grayscale) acquisition (see Section 2.2.1). For these cameras (such as the DAVIS240), since the optical parameters are the same (same lens and same sensor), calibration can be performed using conventional techniques, or event camera specific techniques. The result should be the same (or similar).

Latency evaluation of switching between event and frame modes The DVS240 camera being used allows the capture of frames and events, but only one at a time. During runtime, it is possible to switch between the two modes at any time, though this takes a non-negligible amount of time. In this experiment we try to measure exactly how long it takes to switch between modes.

The ROS package rpg_dvs_ros was used to acquire data from the camera, and to perform the switch between modes. In order to generate a controllable and predictable motion, a stepper motor with a wooden board attached to the shaft was used, in a clock-like mechanism. The stepper rotated at a frequency of 0.4 Hz (2.5 s per revolution). Fig. C.4(a) illustrates the setup used.

A continuous stream of data was captured, switching between events and frames with no fixed interval. For each time the mode switched, the angle between the last event line and the first frame was compared (and vice-versa, when switching from frames to events), and the time elapsed was computed using $\frac{\text{angle}}{\text{freq}} = \frac{\text{angle}}{0.4}$.

Using this approach, the average angle measured switching from frames to events was between 60-70°. This results in a latency of around 0.5 s. The switch between frames and events was not measurable with this method, as the switch in this case was seamless.

A strange artefact is worth noting, however. The first image captured was always a bright (full white) image (Fig. C.4(b,c)), with very little detail. It is not clear whether this is always the case because of insufficient exposure or because of the auto-focus adjusting (which was supposed to be disabled). Nevertheless, this has to be taken into account when relying on this switch for algorithms that need it.

Furthermore, the switch from frames to events was often accompanied by some random events contaminating the image. These were not as regular, however, as they occurred sometimes, but not always, and with a varying interval of time. This too should be taken into account for methodologies relying on switching between modes.

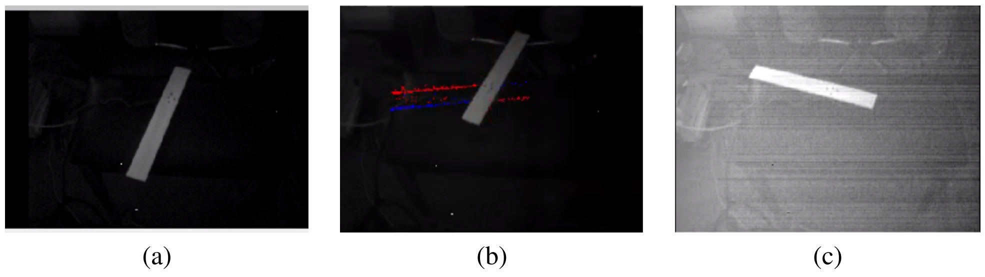


Figure C.4: Setup used, showing a rotating wood bar attached to a stepper motor. Overlapping image of frame and events, showing the elapsed time indirectly through the bar angle between the two modes, and first frame after switching, with low quality and very bright overall.

IMU calibration Just like the camera needs calibration, so do other sensors, in particular the IMU. In order to minimize the effect of noise and bias in the estimation, it is important to characterise these parameters. Ideally, they are supplied by the manufacturer in the datasheet. Oftentimes, however, they are not, and it is necessary to experimentally determine them.

We used the Kalibr framework [1] to characterise these parameters. [Board, 1998] introduces the technique used by Kalibr, which consists of recording sensor measurement while standing still for a large amount of time (at least 4 hours), and then creating a plot with the log-log Allan deviation, as shown in Fig. C.5. Two curves are then fitted to the plot, one with slope $-1/2$, and a second with slope $1/2$.

σ_{gyro} and σ_{acc} are taken directly at $\tau = 1$s, as we assume the noise power in most inertial sensors is dominated by noise at this frequency (point 1 in the plot). ω_b and a_b corresponds to the value at which the line with slope $1/2$ crosses $\tau = 3$s (point 2 in the plot).

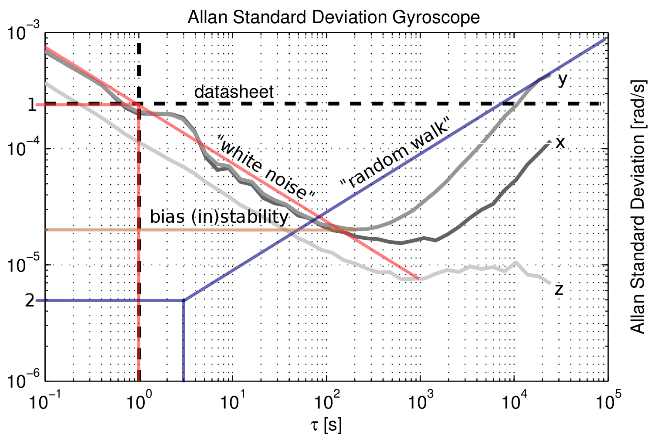


Figure C.5: Allan standard deviation of a gyroscope, showing the fitted lines, and noise characterisation, from the Kalibr wiki

[1] https://github.com/ethz-asl/kalibr

Bibliography

[Akolkar et al., 2018] Akolkar, H., Ieng, S., and Benosman, R. (2018). Real-time high speed motion prediction using fast aperture-robust event-driven visual flow. *arXiv preprint arXiv:1811.11135*.

[Asl et al., 2019] Asl, R. M., Hagh, Y. S., Simani, S., and Handroos, H. (2019). Adaptive square-root unscented Kalman filter: An experimental study of hydraulic actuator state estimation. *Mechanical Systems and Signal Processing*, 132:670–691.

[Barfoot and Furgale, 2014] Barfoot, T. D. and Furgale, P. T. (2014). Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693.

[Barrau and Bonnabel, 2015] Barrau, A. and Bonnabel, S. (2015). An EKF-SLAM algorithm with consistency properties. *arXiv preprint arXiv:1510.06263*.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer.

[Board, 1998] Board, I. (1998). IEEE standard specification format guide and test procedure for single-axis interferometric fiber optic gyros. *IEEE Std*, pages 952–1997.

[Bonnabel, 2012] Bonnabel, S. (2012). Symmetries in observer design: Review of some recent results and applications to EKF-based SLAM. *Robot Motion and Control 2011*, pages 3–15.

[Brossard et al., 2017] Brossard, M., Bonnabel, S., and Barrau, A. (2017). Unscented Kalman filtering on Lie groups for fusion of IMU and monocular vision. In *Proc. Int. Conf. Robot. Automat.(ICRA)*, pages 1–9.

[Censi and Scaramuzza, 2014] Censi, A. and Scaramuzza, D. (2014). Low-latency event-based visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 703–710. IEEE.

[Clady et al., 2015] Clady, X., Ieng, S.-H., and Benosman, R. (2015). Asynchronous event-based corner detection and matching. *Neural Networks*, 66:91–106.

[Cohen, 2015] Cohen, G. K. (2015). *Event-based feature detection, recognition and classification*. PhD thesis, Western Sydney University (Australia).

[Cook et al., 2011] Cook, M., Gugelmann, L., Jug, F., Krautz, C., and Steger, A. (2011). Interacting maps for fast visual interpretation. In *The 2011 International Joint Conference on Neural Networks*, pages 770–776. IEEE.

[Deray and Solà, 2020] Deray, J. and Solà, J. (2020). Manif: A micro Lie theory library for state estimation in robotics applications. *ArXiv*.

[Falanga et al., 2020] Falanga, D., Kleber, K., and Scaramuzza, D. (2020). Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40).

[Gallego et al., 2019] Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A., Conradt, J., Daniilidis, K., et al. (2019). Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*.

[Gallego et al., 2018] Gallego, G., Rebecq, H., and Scaramuzza, D. (2018). A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3867–3876.

[Gallego and Scaramuzza, 2017] Gallego, G. and Scaramuzza, D. (2017). Accurate angular velocity estimation with an event camera. *IEEE Robotics and Automation Letters*, 2(2):632–639.

[Gehrig et al., 2020] Gehrig, D., Rebecq, H., Gallego, G., and Scaramuzza, D. (2020). EKLT: Asynchronous photometric feature tracking using events and frames. *International Journal of Computer Vision*, 128(3):601–618.

[Giulioni et al., 2015] Giulioni, M., Corradi, F., Dante, V., and Del Giudice, P. (2015). Real time unsupervised learning of visual stimuli in neuromorphic VLSI systems. *Scientific reports*, 5(1):1–10.

[Harris et al., 1988] Harris, C. G., Stephens, M., et al. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer.

[Heeger and Jepson, 1992] Heeger, D. J. and Jepson, A. D. (1992). Subspace methods for recovering rigid motion I: Algorithm and implementation. *International Journal of Computer Vision*, 7(2):95–117.

[Horn et al., 1986] Horn, B., Klaus, B., and Horn, P. (1986). *Robot vision*. MIT press.

[Ieng et al., 2017] Ieng, S. H., Carneiro, J., and Benosman, R. B. (2017). Event-Based 3D Motion Flow Estimation Using 4D Spatio Temporal Subspaces Properties. *Frontiers in neuroscience*, 10:596.

[Julier and Uhlmann, 1997] Julier, S. J. and Uhlmann, J. K. (1997). New extension of the Kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

[Kim et al., 2008] Kim, H., Handa, A., Benosman, R., Ieng, S.-H., and Davison, A. J. (2008). Simultaneous mosaicing and tracking with an event camera. *J. Solid State Circ*, 43:566–576.

[Kim et al., 2016] Kim, H., Leutenegger, S., and Davison, A. J. (2016). Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *European Conference on Computer Vision*, pages 349–364. Springer.

[Li et al., 2018] Li, K., Frisch, D., Radtke, S., Noack, B., and Hanebeck, U. D. (2018). Wavefront Orientation Estimation Based on Progressive Bingham Filtering. In *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, pages 1–6. IEEE.

[Lichtsteiner et al., 2008] Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128$\times$ 128 120 dB 15μ s Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE journal of solid-state circuits*, 43(2):566–576.

[Liu et al., 2013] Liu, S.-C., van Schaik, A., Minch, B. A., and Delbruck, T. (2013). Asynchronous Binaural Spatial Audition Sensor With 2 x 64 x 4 Channel Output. *IEEE transactions on biomedical circuits and systems*, 8(4):453–464.

[Loianno et al., 2016] Loianno, G., Watterson, M., and Kumar, V. (2016). Visual inertial odometry for quadrotors on SE (3). In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1544–1551. IEEE.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

[Lucas et al., 1981] Lucas, B. D., Kanade, T., et al. (1981). An iterative image registration technique with an application to stereo vision. In *IJCAI*. Vancouver, British Columbia.

[Luo, 2015] Luo, L. (2015). *Principles of neurobiology*. Garland Science.

[Martins, 2019] Martins, M. (2019). Determining the orientation of a RGB camera embedded on an artificial eye.

[Mead and Mahowald, 1988] Mead, C. A. and Mahowald, M. A. (1988). A silicon model of early visual processing. *Neural networks*, 1(1):91–97.

[Mebius, 2007] Mebius, J. E. (2007). Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. *arXiv preprint math/0701759*.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. *AAAI/IAAI*, 593598.

[Mourikis and Roumeliotis, 2007] Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint Kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572. IEEE.

[Mueggler et al., 2017a] Mueggler, E., Bartolozzi, C., and Scaramuzza, D. (2017a). Fast event-based corner detection. In *BMVC*. University of Zurich.

[Mueggler et al., 2014] Mueggler, E., Huber, B., and Scaramuzza, D. (2014). Event-based, 6-DOF pose tracking for high-speed maneuvers. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2761–2768. IEEE.

[Mueggler et al., 2017b] Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., and Scaramuzza, D. (2017b). The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *The International Journal of Robotics Research*, 36(2):142–149.

[Müller, 2017] Müller, A. (2017). Coordinate mappings for rigid body motions. *Journal of Computational and Nonlinear Dynamics*, 12(2).

[Mur-Artal et al., 2015] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163.

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262.

[Nistér et al., 2004] Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. IEEE.

[Qiao et al., 2015] Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., and Indiveri, G. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141.

[Qin et al., 2018] Qin, T., Li, P., and Shen, S. (2018). Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020.

[Rebecq et al., 2018a] Rebecq, H., Gallego, G., Mueggler, E., and Scaramuzza, D. (2018a). EMVS: Event-based multi-view stereo—3D reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 126(12):1394–1414.

[Rebecq et al., 2018b] Rebecq, H., Gehrig, D., and Scaramuzza, D. (2018b). ESIM: an open event camera simulator. In *Conference on Robot Learning*, pages 969–982.

[Rebecq et al., 2017] Rebecq, H., Horstschaefer, T., and Scaramuzza, D. (2017). Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *BMVC*.

[Rebecq et al., 2019] Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. (2019). High speed and high dynamic range video with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Reinbacher et al., 2017] Reinbacher, C., Munda, G., and Pock, T. (2017). Real-time panoramic tracking for event cameras. In *2017 IEEE International Conference on Computational Photography (ICCP)*, pages 1–9. IEEE.

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443. Springer.

[Sanket et al., 2020] Sanket, N. J., Parameshwara, C. M., Singh, C. D., Kuruttukulam, A. V., Fermüller, C., Scaramuzza, D., and Aloimonos, Y. (2020). Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657. IEEE.

[Scaramuzza and Fraundorfer, 2011] Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92.

[Schönemann, 1966] Schönemann, P. H. (1966). A generalized solution of the orthogonal Procrustes problem. *Psychometrika*, 31(1):1–10.

[Schuenke et al., 2020] Schuenke, M., Schulte, E., Schumacher, U., MacPherson, B., and Stefan, C. (2020). *Head, Neck, and Neuroanatomy (THIEME atlas of anatomy)*. Thieme Medical Publishers.

[Siciliano and Khatib, 2016] Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. springer.

[Sun et al., 2021] Sun, S., Cioffi, G., De Visser, C., and Scaramuzza, D. (2021). Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. Events. *IEEE Robotics and Automation Letters*, 6(2):580–587.

[Thrun, 2002] Thrun, S. (2002). *Probabilistic robotics*. MIT press.

[Tsai, 1987] Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344.

[Van Der Merwe and Wan, 2001] Van Der Merwe, R. and Wan, E. A. (2001). The square-root unscented Kalman filter for state and parameter-estimation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volume 6, pages 3461–3464. IEEE.

[Vasco et al., 2016] Vasco, V., Glover, A., and Bartolozzi, C. (2016). Fast event-based Harris corner detection exploiting the advantages of event-driven cameras. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4144–4149. IEEE.

[Vidal et al., 2018] Vidal, A. R., Rebecq, H., Horstschaefer, T., and Scaramuzza, D. (2018). Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001.

[Wan and Van Der Merwe, 2000] Wan, E. A. and Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee.

[Wang and Chirikjian, 2008] Wang, Y. and Chirikjian, G. S. (2008). Nonparametric second-order theory of error propagation on motion groups. *The International Journal of Robotics Research*, 27(11-12):1258–1273.

[Weikersdorfer and Conradt, 2012] Weikersdorfer, D. and Conradt, J. (2012). Event-based particle filtering for robot self-localization. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 866–870. IEEE.

[Zhang, 2000] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334.

[Zhu et al., 2017] Zhu, A. Z., Atanasov, N., and Daniilidis, K. (2017). Event-based feature tracking with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4470. IEEE.

[Zihao Zhu et al., 2017] Zihao Zhu, A., Atanasov, N., and Daniilidis, K. (2017). Event-based visual inertial odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5391–5399.