# Event-based detection and tracking

David Reverter Valeiras

# Université Pierre et Marie Curie

Doctoral School: Sciences mécaniques, acoustique, électronique et robotique

Institut de la Vision, Vision and Natural Computation team

# Event-Based Detection and Tracking


Defended by:

David Reverter Valeiras

Specialty:

Computer Science

Directed by:

Prof. Ryad B. Benosman

Dr. Sio-Hoï Ieng


Jury composed of:

| | |
|---|---|
| Prof. Bernabé Linares Barranco | Reviewer |
| Dr. Laurent Perrinet | Reviewer |
| Prof. Stéphane Régnier | Examinator |
| Prof. Ryad B. Benosman | PhD Advisor |

Invited member:

| | |
|---|---|
| Dr. Sio-Hoï Ieng | Co-advisor |

# Acknowledgments

I would first like to thank my supervisors Prof. Ryad Benosman and Dr. Sio-Hoï Ieng for making this thesis possible. Their guidance and support have been a fundamental help for me to accomplish this work.

I would also like to stress the important role of the *loulous*, who have made the whole process a lot easier. Their kindness and sympathy have been a crucial support during all these years, and I am very grateful for their help, both moral and scientific. The great ambiance in the lab (or after hours in *Le Gamin*) has made this effort a pleasant one.

Asimismo, querría reconocer el apoyo de mis amistades de Cité Universitaire. En estos años he tenido la suerte de conocer a un puñado de personas a quienes espero ya conservar de por vida. A menudo me han escuchado quejarme de las dificultades que he ido encontrando en este proceso, y han sido una ayuda fundamental. Hemos compartido, además, muchos momentos de alegría y diversión, tan liberadores como necesarios.

También querría agradecer la ayuda de quienes, habiéndome conocido en otro tiempo y en otra ciudad, han tenido la amabilidad de seguir acordándose de mí y enviando mensajes de vez en cuando, para mandarme ánimos o preguntar qué tal la vida.

Por último, quiero agradecer a mi familia todo lo que han hecho por mí a lo largo de este tiempo. Su apoyo incodicional, su constante refuerzo positivo y su exagerada fe en mi capacidad han movido montañas, y han sido una de las razones principales para continuar adelante en los momentos más difíciles, tanto en la tesis como en la vida. A ellas y ellos va dedicada esta tesis.

# Publications

Reverter Valeiras, D., Lagorce, X., Clady, X., Bartolozzi, C., Ieng, S. H., & Benosman, R. (2015). "An Asynchronous Neuromorphic Event-Driven Visual Part-Based Shape Tracking". *IEEE Transactions on Neural Networks and Learning Systems*, 26(12), 3045-3059.

Akolkar[*], H., Reverter Valeiras[*], D., Benosman, R., & Bartolozzi, C. (2015, June). "Visual-Auditory saliency detection using event-driven visual sensors". *IEEE 2015 International Conference on In Event-based Control, Communication, and Signal Processing (EBCCSP)*, (pp. 1-6).

Reverter Valeiras, D., Orchard, G., Ieng, S. H., & Benosman, R. (2016). "Neuromorphic Event-Based 3D Pose Estimation". *Frontiers in Neuroscience*, 9, 522.

Reverter Valeiras, D., Kime, S., Ieng, S. H., & Benosman, R. B. (2016). "An Event-Based Solution to the Perspective-$n$-Point Problem". *Frontiers in Neuroscience*, 10.

Reverter Valeiras, D., Clady, X., Ieng, S. H., & Benosman, R. B. (2017) "Event-Based Least Squares Line Fitting and Segment Detection" (under preparation).

[*] equal contribution

# Détection et Suivi Événementiels


## Résumé:

Les caméras événementielles neuromorphiques sont un nouveau type de capteurs bioinspirés, dont le principe de fonctionnement s'inspire de la rétine. Contrairement aux caméras conventionnelles, ces dispositifs n'encodent pas l'information visuelle comme une séquence d'images statiques, mais comme un flux d'événements possèdant chacun un temps précis. Chaque pixel est indépendant et génère des événements de manière asynchrone lorsqu'un changement de luminosité suffisamment important est detecté à la position correspondante du plan focal. Cet échantillonnage en amplitude du signal lumineux permet d'accroître la résolution temporelle, sans augmenter la quantité des données à traiter ni la consommation énergétique. Cette nouvelle façon d'encoder l'information visuelle requiert de nouvelles méthodes pour la traiter, car les algorithmes classiques de traitement d'image ne parviennent pas à exploiter l'integralité du potentiel de cette information. L'objectif principal de cette thèse est le développement d'algorithmes événementiels pour la détection et le suivi d'objets. Ces algorithmes sont spécifiquement conçus pour traiter les données produites par des caméras neuromorphiques.

Dans un premier temps deux algorithmes 2D sont presentés. D'abord, un "*tracker*" plan est décrit. Cet algorithme associe à un objet une série de formes simples reliées par des ressorts. Le système mécanique virtuel résultant est mis à jour pour chaque événement. Le chapitre suivant présente un algorithme de détection de lignes et de segments, pouvant constituer une primitive (*feature*) événementielle de bas niveau. Ensuite, deux méthodes événementielles pour l'estimation de la pose 3D sont présentées. Le premier de ces algorithmes 3D est basé sur l'hypothèse que l'estimation de la pose est toujours proche de la position réelle, et requiert donc une initialisation précise et, dans un premier temps, manuelle. Le deuxième de ces algorithmes 3D est conçu pour surmonter cette limitation. Toutes les méthodes présentées mettent à jour l'estimation de la position (2D ou 3D) pour chaque événement. Ceci résulte en une série de *trackers* capables d'estimer la position de l'objet suivi avec une résolution temporelle de l'ordre de la microseconde. Chaque méthode est illustrée avec des expériences, et comparée avec d'autres algorithmes issus de l'état-de-l'art. Cette thèse montre que la vision événementielle permet de reformuler une vaste série de problèmes en vision par ordinateur, souvent donnant lieu à des algorithmes plus simples, donc moins coûteux, sans sacrifier la précision.



**Mots clés:** Vision Neuromorphique, Vision Artificielle, Suivi Visuel, Estimation de Position 3D.

# Event-based Detection and Tracking


## Abstract:

Neuromorphic event-based cameras are a new type of biomimetic vision sensors, whose principle of operation is inspired by the functioning of the retina. Unlike conventional cameras, these devices do not encode visual information as a sequence of static frames, but as a stream of precisely timestamped events. Every pixel is independent and asynchronously generates events when it detects a sufficient amount of change in the luminance at its corresponding field of view. This frame-free approach avoids redundant sampling of previously known information, resulting in a drastic increase of the temporal resolution without raising the amount of data to process or the energy consumption. This new way of encoding visual information calls for new processing methods, as classical image-based algorithms do not fully exploit the potential of event-based neuromorphic cameras. The main objective of this thesis is the development of truly event-based algorithms for visual detection and tracking.

In the first place two plane trackers are introduced. Firstly, a part-based shape tracking is presented. This method represents an object as a set of simple shapes linked by springs. The resulting virtual mechanical system is simulated with every incoming event. Next, a line and segment detection algorithm is introduced, which can be employed as an event-based low level feature. Two event-based methods for 3D pose estimation are then presented. The first of these 3D algorithms is based on the assumption that the current estimation is close to the true pose of the object, and it consequently requires a manual initialization step. The second of the 3D algorithms is designed to overcome this limitation. All the presented methods update the estimated position (2D or 3D) of the tracked object with every incoming event. This results in a series of trackers capable of estimating the position of the tracked object with microsecond precision. Experiments are provided in order to test each of the methods, comparing them against other state-of-the-art algorithms. This thesis shows that event-based vision allows to reformulate a broad set of computer vision problems, often resulting in simpler but accurate algorithms.



**Keywords:** Neuromorphic Vision, Artificial Vision, Visual Tracking, 3D Pose Estimation.

# Contents

# Chapter 1

# Introduction

*Neuromorphic Engineering* is an emerging field, whose goal is to build hardware and develop applications that mimic the functioning of the nervous system, using massively parallel hybrid analog/digital VLSI circuits [1]. The term *neuromorphic* was coined in the late eighties by Carver Mead [2], whose seminal work originated this interdisciplinary field [3] in collaboration with prominent scientists Max Delbrück, John Hopfield and Richard Feynman [4]. There are three main motivations for such an approach:

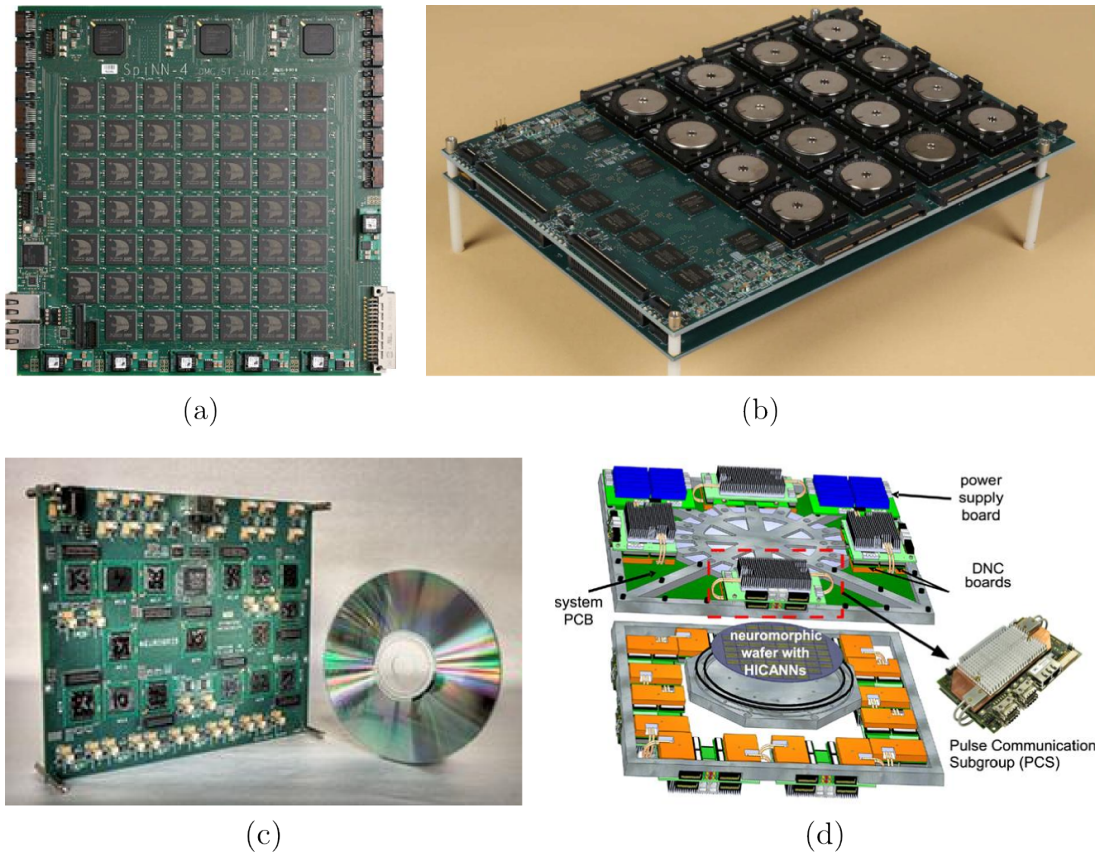
- In spite of their increasing capability and computational power, modern computers still fail to accomplish tasks that our brains seem to solve effortlessly. Consider, for example, object recognition or natural language processing: biological systems still outperform the most advanced and powerful computers available, especially when it comes to real-world scenarios. Despite the strong research effort and the big advances achieved in recent years, the robustness and adaptability of the biological nervous system remain out of reach for even the most developed and powerful artificial systems.

- Neurobiological processing systems are extremely efficient computational devices: in spite of its stunning capabilities, the energy consumption of the human brain is lower than that of a typical computer [5]. In a context hallmarked by the ubiquity of artificial silicon-based systems, biology has a lot to teach us about how to reduce power requirements. This can be a crucial characteristic for a number of applications, such as mobile battery-powered devices, drones or robots.

- Building bioinspired computational devices provides a valuable insight into the way the nervous system works. In this sense, neuromorphic engineering provides a powerful tool to test hypothesis and advance in our understanding of the computational principles governing the biological brain.

In recent years, the field of neuromorphic engineering has received increasing attention and a number of neuromorphic applications and devices have been developed. For a recent review of neuromorphic systems, sensors and processing, the interested reader can refer to [6], where neuromorphic devices are classified into two broad categories:

- **Processing subsystems** (see Fig. 1.1): they correspond to the biological brain. Some examples include Spinnaker [7], IBM's TrueNorth [8], Neurogrid [9] or BrainScaleS [10]-[11]. A recent review of neuromorphic computing systems can be found in [12].




(a) (b) (c) (d)

Figure 1.1: Processing subsystems. (a) A 48 chip SpiNNaker board[1]. (b) DARPA Synapse 16 chip board with IBM TrueNorth[2]. (c) Neurogrid board containing 16 neurocores[3]. (d) Schematic of the FACETS/BrainScaleS waferscale neuromorphic system[4].

- **Sensory subsystems** (see Fig. 1.2): they correspond to biological sensory organs. Examples include neuromorphic tactile sensing (skin) [13]-[14], olfactory systems [15], neuromorphic silicon retinas [16]-[21] and neuromorphic cochleae [22]-[24].

This PhD thesis focuses on the development of applications specifically designed to work on the output of neuromorphic silicon retinas. Unlike conventional cameras, neuromorphic retinas do not encode visual information as a sequence of static frames. Instead, they acquire this information asynchronously: every pixel is independent and autonomously encodes visual information in its field of view into precisely timestamped events. Thus, neuromorphic vision sensors do not sample data at an arbitrary frequency

[1]Source: Bhattacharya, B. S., Patterson, C., Galluppi, F., Durrant, S. J., & Furber, S. (2014). Engineering a thalamo-cortico-thalamic circuit on SpiNNaker: a preliminary study toward modeling sleep and wakefulness, *Frontiers in Neural Circuits*, 8. CC BY, https://doi.org/10.3389/fncir.2014.00046

[2]By DARPA SyNAPSE - http://www.darpa.mil/NewsEvents/Releases/2014/08/07.aspx, Public Domain, https://commons.wikimedia.org/w/index.php?curid=34614979

[3]By Ncousin - Own work, CC BY-SA 3.0, https://en.wikipedia.org/w/index.php?curid=37953993

[4]Source: Scholze, S., Schiefer, S., Partzsch, J., Hartmann, S., Mayr, C. G., Höppner, S., & Schüffny, R. (2011). VLSI implementation of a 2.8 Gevent/s packet-based AER interface with routing and event sorting functionality. *Frontiers in Neuroscience*, 5. CC BY, https://doi.org/10.3389/fnins.2011.00117

(a) (b)

Figure 1.2: Sensory subsystems. (a) iCub Robot equipped with tactile sensors (skin)[1]. (b) ATIS silicon retina.

imposed by an artificial clock signal that has no relation to the source of the visual information [19]. Instead, they are driven by "events" happening within the scene, and they transmit information in an asynchronous manner, as the biological eye does. This frame-free approach greatly reduces the amount of redundant information generated by the sensor, allowing for a drastic increase of its temporal resolution without raising the amount of data to process or the energy consumption.

As soon as change or motion is involved (which is the case for most machine vision applications), the universally accepted paradigm of stroboscopic frame acquisition entails a serious problem: when a conventional camera observes a dynamic scene, and because there is no relation between the frame rate controlling the pixel's data acquisition process and the dynamics present in the scene, over-sampling or under-sampling will occur, and moreover both will usually happen at the same time. As different parts of a scene usually have different dynamic contents, a single sampling rate governing the exposure of all pixels in an imaging array will naturally fail to adequately acquire all these different dynamics simultaneously present.

Consider a natural scene with a fast moving object in front of a static background: when acquiring such a scene with a conventional video camera, motion blurring of the moving object between adjacent frames will result from under-sampling the fast motion of the object, while repeatedly sampling and acquiring static background over and over again will lead to large amounts of redundant, previously known data that do not contain any new information. As a result, the scene is simultaneously under- and over-sampled. There is nothing that can be done about this sub-optimal sampling as long as all pixels of an image sensor share a common timing source that controls exposure intervals (such as a frame-clock).

Frame-based methods are of course suitable for many applications, as long as the frame rate is able to capture the motion. However, even if the frame rate is sufficient, it is always mandatory to process non-relevant information. Most vision algorithms, especially when dealing with dynamic input, have to deal with a mix of useless and bad quality data to deliver useful results, and continuously invest in power and resource-hungry complex processing to make up for the inadequate acquisition.

[1]By Xavier Caré - Wikimedia Commons, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=33368478

One of the first attempts of designing a neuromorphic vision sensor is the pioneering work of Mahowald [17] in the late eighties. Since then, a variety of these event-based devices have been created, including gradient-based sensors sensitive to static edges [25], temporal contrast vision sensors sensitive to relative luminance change [19]-[21], edge-orientation sensitive devices and optical-flow sensors [26]-[27]. Most of these sensors output compressed digital data in the form of asynchronous address events (AER) [28]. A complete review of the history and existing sensors can be found in [29].

The increasing availability and the improving quality of neuromorphic vision sensors open up the potential to introduce a shift in the methodology of acquiring and processing visual information in various demanding machine vision applications [30]-[32]. As this thesis will show, asynchronous acquisition allows us to introduce a series of novel computationally efficient techniques, that rely on the accurate timing of individual pixels' response to visual stimuli. Indeed, this new way of encoding visual information calls for new processing methods: classical algorithms, designed to work on images, do not take full advantage of the potential of neuromorphic cameras. Reconstructing frames from the events to feed classical image algorithms is always possible, but this leads to two sources of waste: first, extra resources are necessarily consumed when events are gathered into frames. Second, the temporal accuracy of the native signal is usually lost. Avoiding these two undesired effects is the main motivation for designing truly event-based algorithms, which is the primary goal of this thesis.

The Asynchronous Time-based Image Sensor (ATIS) [20] used in this work is one of these neuromorphic silicon retinas. Its event generation process is illustrated in Fig. 1.3: each pixel contains a change detector circuit that independently generates events when a sufficient amount of change is detected in the log of the luminance at the corresponding position on the focal plane. We distinguish between "positive" and "negative" events, depending on whether the luminance increased or decreased. The change detector then initiates the measurement of an exposure/gray scale value immediately after the luminance change has been detected. The exposure measurement circuit in each pixel individually encodes the absolute instantaneous pixel luminance into the timing of asynchronous event pulses, more precisely into inter-event intervals. All the techniques presented in this thesis employ the change detector events produced by an ATIS camera.

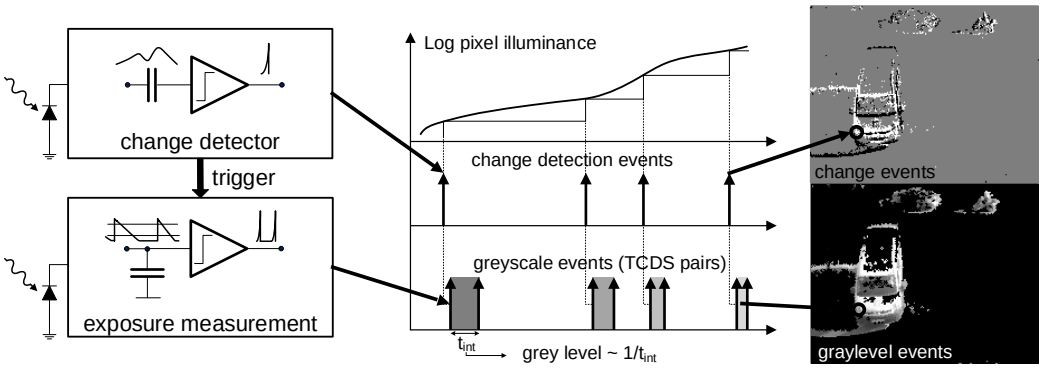



Figure 1.3: Functional diagram of an ATIS pixel [21]. Two types of asynchronous events, encoding change and brightness information, are generated and transmitted individually by each pixel in the imaging array.

This PhD thesis introduces a set of event-based techniques for visual detection, tracking and pose estimation, both 2D and 3D. The outline of this document is as follows:

first, Chapters 2 and 3 introduce the 2D tracking techniques developed in this thesis. In the first place, a part-based [33] shape tracking is presented in Chapter 2. This plane tracker, greatly inspired by the *Pictorial Structures* model introduced in [34], defines the model of an object as a set of simple shapes linked by springs. This results in a real-time algorithm tracking complex objects at a low computational cost. As it will be shown, the high temporal resolution of the event-based camera allows us to model the resulting virtual mechanical system in a simple yet robust fashion. Results are provided, including the application of the technique to face tracking.

Next, in Chapter 3, an event-based line and segment detection algorithm is presented. Lines and segments provide fundamental low level features for vision-based systems, and are often used as the input for different computer vision algorithms (see, e.g. [35]-[37]). This constitutes a fundamental previous step in order to build complex event-based vision applications. The accuracy of the method is first studied in a controlled scene. Next, results obtained from urban scenarios are shown.

An important motivation for this research was its application to visual servoing (i.e. controlling the motion of a robot using visual feedback [38]-[39]). A family of visual servoing techniques, usually referred to as *position-based visual servoing* (PBVS), requires the estimation of the camera's 3D pose relative to some tracked object. This motivates the research for the 3D pose estimation [40] algorithms presented in chapters 4 and 5. The 3D tracker presented in Chapter 4 provides an estimation of the relative pose of a known object with respect to a calibrated neuromorphic camera. The method is intuitively simple, and iteratively updates the estimated pose with every incoming event, using the distance to the line of sight of events. This tracker is able of successfully estimating the pose of different objects, including very fast moving objects or in the case of ego-motion.

The pose estimation algorithm presented in Chapter 4 assumes that an initial estimate of the pose of the tracked object is available. As a result, a manual initialization step is usually required. Moreover, it requires a complex 3D structure for the model of the tracked object. The event-based solution to the Perspective-$n$-Point problem [41] introduced in Chapter 5 is designed to overcome these limitations. The method is based on the minimization of a given error function. In order to solve this problem, a virtual mechanical system is built (as in Chapter 2). This system is constructed in such a way that its potential energy is equal to the error function that needs to be minimized. Consequently, since mechanical systems tend to a minimum of their potential energy, when we simulate the behavior of this system we will tend to minimize the error, obtaining an accurate 3D pose of the tracked object.

A brief State-of-Art on the corresponding research topic is included at the beginning of each chapter.

Finally, in Chapter 6, the discussion and conclusions are presented. In particular, the main contributions of this thesis are discussed, which can be summarized in the following points:

- This work advances in the development of purely event-based techniques. All the methods presented in this thesis update their state with every incoming event. In this sense, this thesis establishes and tests a number of methods and tools that can be applied to a wide number of event-based machine vision applications.

- It progresses in the development of event-based 2D tracking techniques. When this work began, very few event-based visual tracking algorithms had been developed. The most common technique was to consider a blob tracker, whose position is given

by the average of the incoming events [42]-[43]. This thesis introduces much more elaborated tracking techniques.

- It presents some of the first purely event-based techniques for 3D pose estimation. To our knowledge, the algorithm introduced in Chapter 5 is the only event-based solution to the P$n$P problem available today.

# Chapter 2

# Asynchronous Part-based Shape Tracking

## 2.1 Introduction

Object tracking is a fundamental task in many computer vision applications, including video-based surveillance systems [44]-[45], human-computer interaction [46], augmented reality [47] or traffic monitoring [48]. Despite being extensively studied over the last decade, fast object recognition and tracking remains a challenging and computationally expensive problem.

A major application of visual tracking is face detection and tracking. This area has been dominated in the recent years by *appearance-based* methods [49]. These techniques learn a global representation of the object from a set of training images and have been deeply studied, originating from the pioneering work of Viola & Jones introduced in [50], later improved by different researchers such as [51]-[57]. A complete review of the recent advances in face detection is presented in [58]. In the field of event-based vision, an example can be found in [59].

*Appearance-based* methods implicitly assume a rigid spatial relation between the parts that constitute the object. Other methods that overcome this limitation rely on modeling an object as a set of simple parts with a flexible geometric relation between them. These techniques, known as *part-based* methods, have received a lot of attention as they allow the recognition of generic classes of objects and the estimation of their pose. An early example of this technique can be found in [34], in which a model known as *Pictorial Structures* is introduced. In the *Pictorial Structures* framework the local elements of the object are assumed to be rigid and linked by springs. A probabilistic model is defined and looks for the best match that minimizes two cost functions: one for the matching of individual parts and another one for the geometric relations between them. The main limitation of this technique, which has prevented its use for a long time, is the high computational cost needed to solve the resulting minimization problem. With the increase in available computational power, this method has received renewed attention (e.g. in [60]-[63]).

In spite of the recent improvements, these techniques are fundamentally limited by the low dynamics imposed by the frame rates of current cameras. In this chapter we present a new approach to the problem, which relies on event-based vision to provide high temporal resolution and sparse output, thus yielding a true dynamic framework to the problem [19]-[20].

Despite its promising characteristics, few object tracking algorithms exploiting the

possibilities of event-driven acquisition have been developed so far. An event clustering algorithm is introduced for traffic monitoring, where clusters can change in size but are restricted to a circular form [42]-[43]. The sensor has been recently applied to track particles in microrobotics [64] and in fluid mechanics [65]. It was also used to track micro-gripper's jaws to provide real-time haptic feedback on the micro meter scale [66]. In [67] a blob tracker is described which is capable of adapting its shape and position to the distribution of incoming events, assuming that it follows a bivariate Gaussian distribution.

The part-based shape tracking introduced in this chapter is designed to track more complex shapes. It defines the model of an object as a set of blob tackers linked by springs. Event-based vision, with its asynchronous acquisition, allows a data-driven part-based update of the model iteratively for every incoming detected event. Thus, instead of explicitly minimizing the energy function, we simply let the system evolve. As we apply the effects of the springs, their elastic energy tends to get smaller, in the same way as it would in a real mechanical system. Therefore the resulting algorithm is simple and still robust.

This chapter is organized as follows: in the first place we give a mathematical description of the stream of visual events generated by neuromorphic cameras in Section 2.2. Next, we describe our part-based shape tracking in Section 2.3, while Section 2.4 presents the experiments carried out to validate the method. Finally, in Section 2.5 we briefly discuss the obtained results.

## 2.2 Stream of Visual Events

As previously explained in Chapter 1, neuromorphic cameras encode visual information as a stream of asynchronous events. Here, events indicate a sufficient change of luminance at a given position on the focal plane and at a precise instant. The stream of visual events can then be mathematically described as follows: let $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T$ be a quadruplet describing an event occurring at time $t_k$ at the spatial position $\boldsymbol{u}_k = [x_k, y_k]^T$ on the focal plane. The *polarity* $p_k$ can take values 1 or -1, depending on whether a positive or negative change of luminance has been detected. The time $t_k$ is often referred to as the *timestamp* of the event, and is expressed with microsecond precision.

For the remainder of this document, the subindex $k$ will always indicate the time dependency of a given event-based variable (i.e. its value at time $t_k$).

## 2.3 Part-Based Shape Tracking

We next present our part-based shape tracking. Let us remind the reader that the method presented in this chapter describes an object as a set of simple shapes linked by springs. Here, the simple shapes are the Gaussian trackers developed in [67] and explained in Section 2.3.1. The spring-like connections are then presented in Section 2.3.2.

### 2.3.1 Gaussian Blob Trackers

When an object moves, the pixels generate events which geometrically form a point cloud that represents the spatial distribution of the observed shape. The event-based Gaussian tracker developed in [67] assumes that these events are normally distributed. According to this assumption, the event cloud approximates a bivariate Gaussian distribution, whose

parameters can be iteratively updated with the incoming events. Thus, the Gaussian tracker is driven by the asynchronous events, causing it to move and to deform so as to approximate the event cloud's spatial distribution.

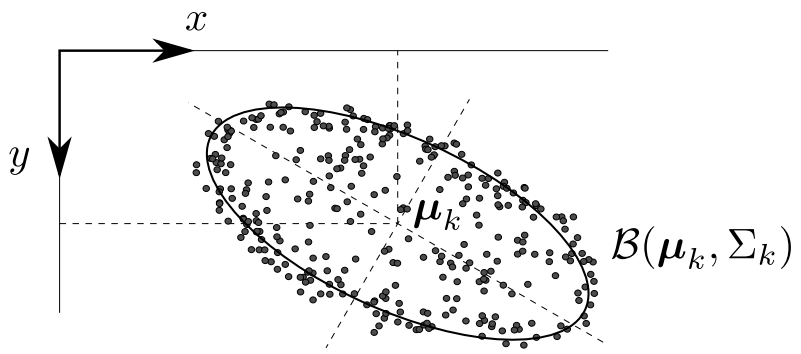



Figure 2.1: A Gaussian tracker $\mathcal{B}$ following a cloud of events is defined by its location $\boldsymbol{\mu}_k$ and covariance matrix $\Sigma_k$.

In order to illustrate the update procedure, let us first assume that we are observing a scene with just one object. Let $\mathcal{B}(\boldsymbol{\mu}_k, \Sigma_k)$ be the Gaussian tracker shown in Fig. 2.1, defined by its mean $\boldsymbol{\mu}_k \in \mathbb{R}^2$ and its covariance matrix $\Sigma_k \in \mathbb{R}^{2\times 2}$. Here, the mean represents the object's position, and the covariance matrix is used to compute its size and orientation (the interested reader can refer to [67] for details). Following the standard convention in this document, the subindex $k$ indicates the time dependency of these variables (i.e. $\boldsymbol{\mu}_k$ represents the value of $\boldsymbol{\mu}$ at time $t_k$, etc.).

The position $\boldsymbol{\mu}_k$ is then computed as the weighted mean of the position of the events previously assigned (i.e. spatio-temporally close) to the tracker. Without loss of generality, and just to ease the notation during the mathematical development, let us assume that all the previous events have been assigned to this Gaussian tracker. This yields the following expression for $\boldsymbol{\mu}_k$:

$$\boldsymbol{\mu}_k = \frac{1}{\Omega_k} \sum_{j=0}^{k} \omega_{k,j} \boldsymbol{u}_{k-j}, \tag{2.1}$$

which is the usual expression for the weighted mean of an event-based variable, as explained in the Appendix A. Here, $\boldsymbol{u}_{k-j}$ is the position of the event $\mathbf{e}_{k-j}$ happening $j$ steps before the current one, with $j = 0, 1, ..., k$. The weight of the corresponding event is then denoted $\omega_{k,j}$, verifying:

$$\omega_{k,j} \geq 0, \quad \forall j = 0, 1, ..., k, \tag{2.2}$$

$$\Omega_k = \sum_{j=0}^{k} \omega_{k,j}. \tag{2.3}$$

We can think of a number of different weighting strategies but, in general, we will set the weights $\omega_{k,j}$ so that they decrease with $j$, giving a greater importance to the most recent events. Here, $\Omega_k$ is just a normalizing factor. The weighting strategy most commonly used in this thesis, that we denote the *standard weighting strategy*, is given by:

$$\omega_{k,j} = \omega_j = \omega_0 (1 - \omega_0)^j. \tag{2.4}$$

A detailed explanation of this weighting strategy is given in the Appendix A.1.

Analogously to $\boldsymbol{\mu}_k$, the covariance matrix takes the value:

$$\Sigma_k = \frac{1}{\Psi_k}\sum_{j=0}^{k}\psi_{k,j}(\boldsymbol{u}_{k-j}-\boldsymbol{\mu}_{k-j})(\boldsymbol{u}_{k-j}-\boldsymbol{\mu}_{k-j})^T, \tag{2.5}$$

with $\psi_k, j$ the corresponding set of weights, and $\Psi_k = \sum_{j=0}^{k}\psi_{k,j}$.

As proven in the Appendix A.1, if the standard weighting strategy is chosen we obtain the following update laws for $\boldsymbol{\mu}_k$, $\Sigma_k$:

$$\boldsymbol{\mu}_k = \omega_0\boldsymbol{\mu}_{k-1} + \big(1-\omega_0\big)\boldsymbol{u}_k, \tag{2.6}$$

$$\Sigma_k = \psi_0\Sigma_{k-1} + \big(1-\psi_0\big)(\boldsymbol{u}_k-\boldsymbol{\mu}_k)(\boldsymbol{u}_k-\boldsymbol{\mu}_k)^T, \tag{2.7}$$

which are the iterative expressions used in [67]. Here, $\omega_0$ and $\psi_0$ are update factors and should be tuned according to the event rate. Let us note that these expressions still hold when not all the previous events have been assigned to the same Gaussian tracker, provided that the position and the covariance matrix of a tracker are only updated when an event is assigned to it.

Next, let us imagine that the scene we are observing contains several objects. Consequently, let us assume that several Gaussian trackers have been initialized. These Gaussian trackers are identified by their index $i$, and denoted $\mathcal{B}^{(i)}(\boldsymbol{\mu}_k^{(i)}, \Sigma_k^{(i)})$. The probability that a Gaussian tracker $\mathcal{B}^{(i)}$ generates an event $\mathbf{e}_k$ at the spatial position $\boldsymbol{u}_k$ is then given by:

$$p_k^{(i)} = \frac{1}{2\pi}|\Sigma_{k-1}^{(i)}|^{-\frac{1}{2}}\exp\Big(-\frac{1}{2}\big(\boldsymbol{u}_k-\boldsymbol{\mu}_{k-1}^{(i)}\big)^T\big(\Sigma_{k-1}^{(i)}\big)^{-1}\big(\boldsymbol{u}_k-\boldsymbol{\mu}_{k-1}^{(i)}\big)\Big), \tag{2.8}$$

where the probability is computed with the last available position and covariance matrix (i.e. the values at $t_{k-1}$). Then, assuming that all the Gaussian trackers have the same prior probability of generating an event, Bayes' theorem [68] allows us to conclude that the tracker that has most likely generated $\mathbf{e}_k$ is the one with the highest Gaussian probability $p_k^{(i)}$. Each incoming event will then be assigned to the tracker with the highest $p_k^{(i)}$, provided that this probability is greater than a predefined threshold, $p_k^{(i)} > \delta$ (usually set to 0.1). Once the most probable tracker has been identified, its parameters are updated by integrating the last distribution with the current event information, as described in (2.6) and (2.7).

Finally, the activity $\mathcal{A}_k^{(i)}$ of each tracker $\mathcal{B}^{(i)}$ is updated with each incoming event $\mathbf{e}_k$, following an exponential decay function which describes the temporal dimension of the Gaussian kernel.

$$\mathcal{A}_k^{(i)} = \begin{cases} \mathcal{A}_{k-1}^{(i)}\exp\Big(-\dfrac{\Delta t_k}{\tau}\Big) + p_k^{(i)}, & \text{if } \mathbf{e}_k \text{ belongs to tracker } i \\ \mathcal{A}_{k-1}^{(i)}\exp\Big(-\dfrac{\Delta t_k}{\tau}\Big), & \text{otherwise,} \end{cases} \tag{2.9}$$

where $\Delta t_k = t_k - t_{k-1}$ is the inter-event time, and $\tau$ is a factor tuning the temporal activity decrease. If the activity $\mathcal{A}_k^{(i)}$ of a tracker $\mathcal{B}^{(i)}$ is greater than a predefined threshold $\mathcal{A}^{(up)}$, then $\mathcal{B}^{(i)}$ is labeled as *active*. Thus, a tracker is said to be active when it is correctly following a cloud of events. Otherwise it is labeled as *inactive*.

### 2.3.2 Spring-Like Connections

The Gaussian tracker introduced in the previous section produces robust results when dealing with simple objects, specially in the case of ellipse-like shapes. When attempting to track a more complex object, we can assume that this object is composed of a set of simple shapes linked by geometric relations. These relations, however, cannot be fixed, as the movement of the object in the 3D space will cause its projection onto the focal plane to be modified.

In order to build a tracker capable of following the structure of observed objects, we model our system as a set of simple trackers linked by springs. Thus, each tracker of the set will be driven both by the incoming events and by the elastic connections linking it to other elements.

#### Euclidean Configuration

The force $\boldsymbol{F}$ exerted by a linear spring is given by the well known Hooke's law [69], which states that the direction of the force is that of the axis of the spring, and its magnitude is given by the expression:

$$\|\boldsymbol{F}\| = C\Delta l \tag{2.10}$$

where $\Delta l = l - l_0$ represents the *elongation* of the spring, which is the difference between its current length $l$ and the equilibrium length $l_0$. $C$ is a characteristic of the spring known as its *stiffness*.

Fig. 2.2(a) shows a linear spring to which an object of mass $m$ has been attached. An energy dissipation mechanism is also added and is represented in this case by an ideal damper. The damping force $\boldsymbol{D}$ applied by a damper is modeled as being proportional and opposed to the speed between its opposite sides:

$$\|\boldsymbol{D}\| = B\frac{d\Delta l}{dt}, \tag{2.11}$$

where $B$ is known as the *viscous damping coefficient* of the damper.

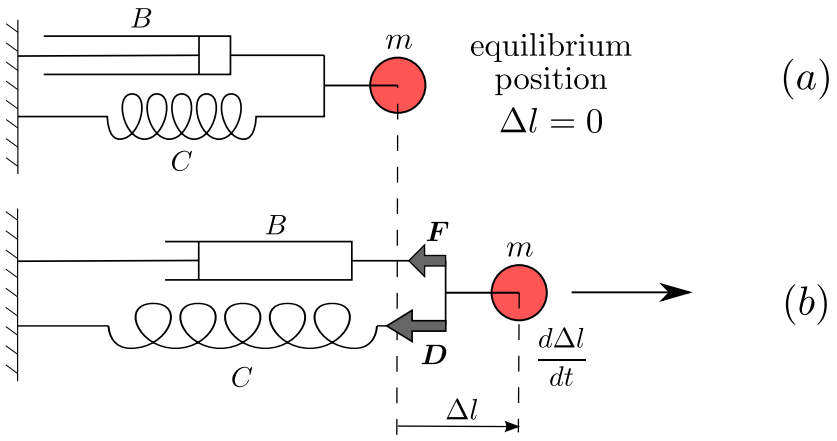



Figure 2.2: Principle of a damped spring: (a) A mass $m$ is attached to a mechanical system composed of a linear spring and a linear damper. (b) When the system is out of its equilibrium position two forces $\boldsymbol{F}$ (elongation) and $\boldsymbol{D}$ (frictional) appear. Their directions are opposed respectively to those of the displacement and the velocity.

Fig. 2.2(b) shows the system out of its equilibrium position and with a certain speed. In that case, two forces $\boldsymbol{F}$ and $\boldsymbol{D}$ appear, their directions being opposed to those of the displacement and the velocity respectively. Applying Newton's second law we obtain the differential equation of the system, needed to solve in order to calculate the object's acceleration, velocity and position:

$$m\frac{d^2\Delta l}{dt} = -C\Delta l - B\frac{d\Delta l}{dt}. \tag{2.12}$$

This is a typical problem in classical mechanics, and its solutions are well known and studied [69]-[70]. If a series of connections is set between the different trackers, assigning masses to these trackers allows us to actually model the behavior of this virtual dynamic system. Even if we are not modeling a real system, keeping the concept of masses for the trackers allows us to control their relative displacements, assigning bigger masses to the elements that we wish to be more stable.

Let $\mathcal{C}^{(ij)}$ be a connection bounding the Gaussian trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$. As shown in Fig. 2.1, the position of the trackers is represented by $\boldsymbol{\mu}_k^{(i)}$, $\boldsymbol{\mu}_k^{(j)}$. Fig. 2.3(a) shows this connection in its equilibrium state, where $l_0^{(ij)}$ represents the equilibrium distance and $\theta_k^{(ij)}$ the angle formed by the axis of the connection and the horizontal axis. In what follows, we will use a simplified representation of the connection, showing only the spring.

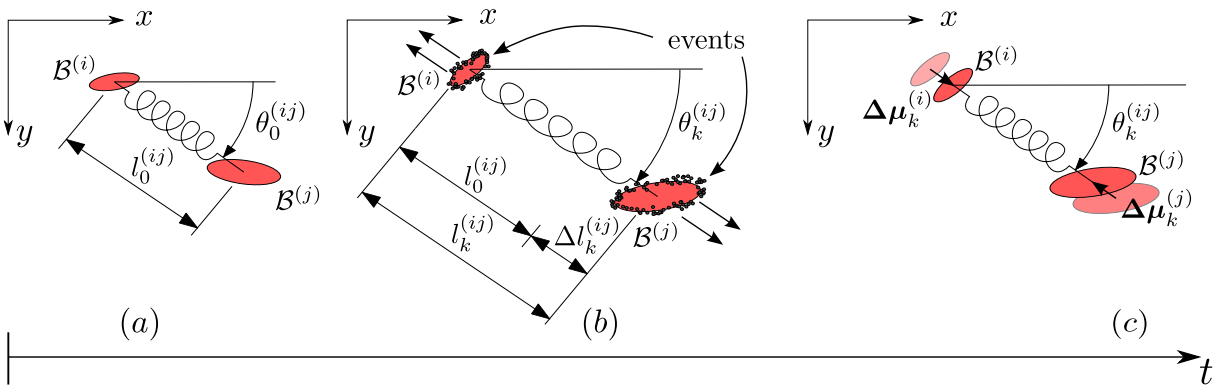



Figure 2.3: (a) Connection $\mathcal{C}^{(ij)}$ (that links the Gaussian trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$) in its initial equilibrium state: $l_0^{(ij)}$ represents the initial length of the spring, and $\theta_0^{(ij)}$ the angle formed by the axis of the connection and the horizontal axis. (b) The trackers follow incoming events, moving the connection away from its equilibrium position. The difference between the current length of the connection $l_k^{(ij)}$ and the initial length is known as the elongation of the spring $\Delta l_k^{(ij)}$. (c) The trackers are driven by the connection, which tries to recover its initial length.

Let us assume that this connection behaves as a single linear damped spring that can freely rotate around its ends, where it is connected to the respective trackers. When modeling the system this way, we are only taking into account the euclidean distance between the trackers, and not at all the direction of the connection. From now on, we will refer to this configuration as the *euclidean configuration*.

Fig. 2.3(b) and Fig. 2.3(c) illustrate the evolution of the trackers from equilibrium, as they are driven by both the incoming events and the spring-like connections. As the events start arriving, they will cause the trackers to move away from their initial positions, eventually producing a certain elongation of the spring. Fig. 2.3(b) shows the state of

the system after the trackers have been displaced by the events, where $l_k^{(ij)}$ represents the current distance between the trackers and $\Delta l_k^{(ij)}$ the corresponding elongation, given by:

$$\Delta l_k^{(ij)} = l_k^{(ij)} - l_0^{(ij)}. \tag{2.13}$$

As a consequence of this elongation, the trackers will then be driven by the spring-like connection, which tries to recover its initial equilibrium length. Fig. 2.3(c) shows the corresponding displacement of the trackers $\boldsymbol{\Delta\mu}_k$, which are always in the opposite direction to the elongation. Thus, when we apply the effect of the springs we update the position of the trackers making:

$$\begin{aligned} \boldsymbol{\mu}_k^{(i)} &\leftarrow \boldsymbol{\mu}_k^{(i)} + \boldsymbol{\Delta\mu}_k^{(i)}, \\ \boldsymbol{\mu}_k^{(j)} &\leftarrow \boldsymbol{\mu}_k^{(j)} + \boldsymbol{\Delta\mu}_k^{(j)}. \end{aligned} \tag{2.14}$$

In order to compute these displacements, we will simply assume that they are proportional to the elongation. This approximation, causing an exponential decay towards equilibrium, is valid under the conditions described in the Appendix B, and will result in the following values for the displacements:

$$\begin{aligned} \boldsymbol{\Delta\mu}_k^{(i)} &= \frac{\alpha^{(ij)}}{m^{(i)}} \Delta l_k^{(ij)} \begin{bmatrix} \cos\left(\theta_k^{(ij)}\right) \\ \sin\left(\theta_k^{(ij)}\right) \end{bmatrix}, \\ \boldsymbol{\Delta\mu}_k^{(j)} &= -\frac{\alpha^{(ij)}}{m^{(j)}} \Delta l_k^{(ij)} \begin{bmatrix} \cos\left(\theta_k^{(ij)}\right) \\ \sin\left(\theta_k^{(ij)}\right) \end{bmatrix}, \end{aligned} \tag{2.15}$$

where $\alpha^{(ij)}$ is a scaling factor that controls the stiffness of the connection, and $m^{(i)}$, $m^{(j)}$ represent the masses associated with the $i^{th}$ and $j^{th}$ trackers respectively. Here, $\theta_k^{(ij)}$ denotes the angle between the axis of the connection and the horizontal line.

### Torsional Configuration

If we want to keep the angle of each connection close to the equilibrium value, we can imagine the trackers as being linked by torsion springs. The force applied by a torsion spring is proportional to the difference between the current angle and the equilibrium angle. Adding an ideal prismatic joint between the trackers allows us to avoid taking into account the distance between them. The equivalent mechanical system can be seen in Fig. 2.4, where $\theta_0^{(ij)}$ represents the initial equilibrium angle of the connection and $\theta_k^{(ij)}$ its current value.

We will refer to this configuration as the *torsional configuration*. In this case, the torsional elongation is given by:

$$\Delta\theta_k^{(ij)} = \theta_k^{(ij)} - \theta_0^{(ij)}, \tag{2.16}$$

which can be corrected by subtracting its mean value along the connections, in order to make the system insensitive to rotation:

$$\widetilde{\Delta\theta}_k^{(ij)} = \Delta\theta_k^{(ij)} - \overline{\Delta\theta}_k, \tag{2.17}$$

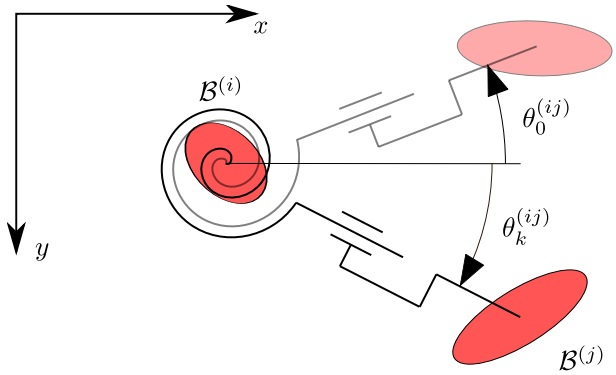



Figure 2.4: Torsional configuration: trackers are linked by torsion springs. The equilibrium angle is initially $\theta_0^{(ij)}$, while $\theta_k^{(ij)}$ denotes its position after torsion.

where $\overline{\Delta\theta}_k$ is the mean torsional elongation. Thus, if all the connections rotate through the same angle in the same direction, the torsional elongation will be zero for all of them, making the system insensitive to rotation.

Next, from the value of the elongation, we compute the corresponding displacements to be applied to the the trackers. As in the case of the euclidean configuration, we simplify the effect of the spring using a first order approximation. Then, if the current relative position is $l_k^{(ij)}[\cos(\theta_k^{(ij)}),\ \sin(\theta_k^{(ij)})]^T$, the new angle will be $\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}$, and the new relative position will be given by:

$$l_k^{(ij)} \begin{bmatrix} \cos\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) \\ \sin\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) \end{bmatrix} \tag{2.18}$$

From here, the displacements to be applied to the trackers are equal to:

$$\begin{aligned} \boldsymbol{\Delta\mu}_k^{(i)} &= \frac{l_k^{(ij)}}{m^{(i)}} \begin{bmatrix} \cos\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) - \cos\left(\theta_k^{(ij)}\right) \\ \sin\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) - \sin\left(\theta_k^{(ij)}\right) \end{bmatrix}, \\ \boldsymbol{\Delta\mu}_k^{(j)} &= -\frac{l_k^{(ij)}}{m^{(j)}} \begin{bmatrix} \cos\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) - \cos\left(\theta_k^{(ij)}\right) \\ \sin\left(\theta_k^{(ij)} + \alpha^{(ij)}\widetilde{\Delta\theta}_k^{(ij)}\right) - \sin\left(\theta_k^{(ij)}\right) \end{bmatrix}. \end{aligned} \tag{2.19}$$

### Cartesian Configuration

If we want to keep both the distance and the angle of each connection close to those of the equilibrium, we can set a horizontal and a vertical equilibrium distances. The equivalent mechanical system is represented in Fig. 2.5, where $\delta x_0^{(ij)}$ and $\delta y_0^{(ij)}$ are the horizontal and vertical equilibrium distances respectively. $\Delta x_k^{(ij)}$ and $\Delta y_k^{(ij)}$ represent the horizontal and vertical elongations, given by the equation:

$$\begin{bmatrix} \Delta x_k^{(ij)} \\ \Delta y_k^{(ij)} \end{bmatrix} = \boldsymbol{\mu}_k^{(i)} - \boldsymbol{\mu}_k^{(j)} - \begin{bmatrix} \delta x_0^{(ij)} \\ \delta y_0^{(ij)} \end{bmatrix} \tag{2.20}$$

When modeling the system this way, the computation becomes very simple. Keeping the same simplifications as in the previous cases, the displacements to be applied to the

trackers are given by:

$$\begin{aligned}\boldsymbol{\Delta\mu}_k^{(i)} &= \frac{\alpha^{(ij)}}{m^{(i)}}\begin{bmatrix}\Delta x_k^{(ij)}\\ \Delta y_k^{(ij)}\end{bmatrix}\\ \boldsymbol{\Delta\mu}_k^{(j)} &= -\frac{\alpha^{(ij)}}{m^{(j)}}\begin{bmatrix}\Delta x_k^{(ij)}\\ \Delta y_k^{(ij)}\end{bmatrix}.\end{aligned} \tag{2.21}$$

We will refer to this configuration as the *cartesian configuration*.

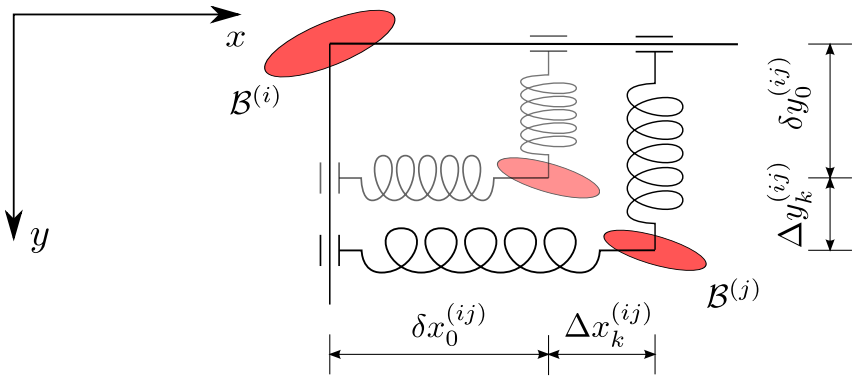



Figure 2.5: Cartesian configuration: the initial distances between the trackers are equal to $\delta x_0^{(ij)}$ and $\delta y_0^{(ij)}$, which correspond to the horizontal and vertical equilibrium distances of the connection. The tracker activity following events originates an elongation given by $\Delta x_k^{(ij)}$ and $\Delta y_k^{(ij)}$ causing a change of distance between trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$.

### 2.3.3 Using the energy as a matching criterion

The elastic energy of a mechanical spring is given by the expression:

$$E = \frac{1}{2}C\big(\Delta l\big)^2. \tag{2.22}$$

In the case of our spring-like connections, and depending on the type of configuration considered, we obtain the following expressions for the energy:

- Euclidean configuration:
$$E_k^{(ij)} = \frac{1}{2}\alpha^{(ij)}\big(\Delta l_k^{(ij)}\big)^2. \tag{2.23}$$
- Torsional configuration:
$$E_k^{(ij)} = \frac{1}{2}\alpha^{(ij)}\Big(\widetilde{\Delta\theta}_k^{(ij)}\Big)^2. \tag{2.24}$$
- Cartesian configuration:
$$E_k^{(ij)} = \frac{1}{2}\alpha^{(ij)}\big((\Delta x_k^{(ij)})^2 + (\Delta y_k^{(ij)})^2\big). \tag{2.25}$$

The elastic energy of a connection constitutes a measure of its deformation. Consequently, the energy of the spring-like connections can be used as a quality criterion for the tracking: if our tracker is correctly following the desired object, chances are that the energy of all the connections will be coherent. Thus, we define two energy-based criteria designed to yield the tracker more robust to partial occlusions.

### Preventing a single tracker from following the wrong cloud of events

Let us imagine that our part-based tracker is correctly tracking an object. Fig. 2.6(a) shows the state of the system in such a situation. As a certain degree of deformation is acceptable, the energy of its connections will typically be different from zero. However, we will assume them to be relatively stable and similar to each other as long as the system is correctly tracking the desired object. Next, let us imagine that a partial occlusion occurs, generating a cloud of events that does not correspond to the tracked object. In a first step, let us imagine that a single tracker $\mathcal{B}^{(i)}$ starts following this wrong cloud of events, while the rest of the trackers are unaffected. Fig. 2.6(b) shows the resulting situation: in this case, the energy of all the connections $\mathcal{C}^{(ij)}$ bounding $\mathcal{B}^{(i)}$ will grow to be much bigger than the rest. Thus, when the energy of all the connections bounding a certain tracker is bigger than a threshold, we will make this tracker stop following events. As we wish to allow stable growth of the elastic energy, this threshold will be defined as proportional to the mean elastic energy of all the connections. The criterion will therefore be expressed by:

$$\text{if } E_k^{(ij)} \geq s\overline{E}_k \;\forall\; j \text{ s.t. } \mathcal{C}^{(ij)} \text{ exists} \Rightarrow \mathcal{B}^{(i)} \text{ insensitive}, \tag{2.26}$$

where $\overline{E}_k$ represents the mean elastic energy of all the connections and $s$ is a positive scaling factor. An insensitive tracker cannot have events assigned to it. As this condition is evaluated for every incoming event, the tracker will be insensitive until the energy of one of its connections drops below the threshold.

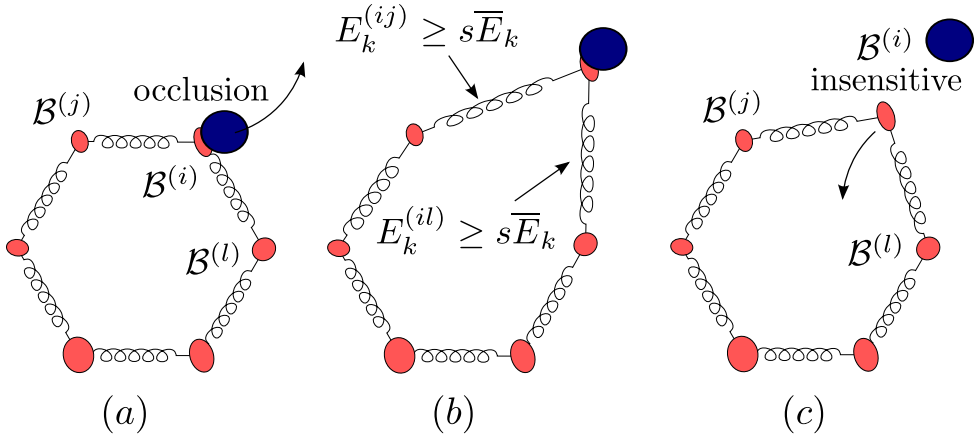



Figure 2.6: (a) The system is correctly tracking an object, until an occlusion occurs. (b) If this occlusion attracts only one tracker $\mathcal{B}^{(i)}$ the energy of every connection linking this tracker will grow to be bigger than the rest. (c) If the elastic energy of every connection linking the tracker $\mathcal{B}^{(i)}$ gets bigger than a threshold (defined as proportional to the mean elastic energy) then the tracker becomes insensitive. This means that no event can be assigned to the tracker. Consequently, it will be driven exclusively by its connections, and quickly recover its equilibrium position relative to its neighbors.

As a consequence of the tracker being insensitive to the incoming events, it will exclusively be driven by the spring-like connections. This will cause it to quickly recover its equilibrium position relative to its neighbors, typically finding the desired object again.

### Preventing a group of trackers from following the wrong cloud of events

The second mechanism is designed to avoid a group of trackers following the wrong cloud of events. Fig. 2.7(a) shows the same system as in the previous case, correctly tracking the desired object. In this case, however, the cloud of events generated by the partial occlusion will attract two trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$. As we can see in Fig. 2.7(b), the previous mechanism will not be activated by this situation, as the energy of the connection bounding $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$ remains stable.

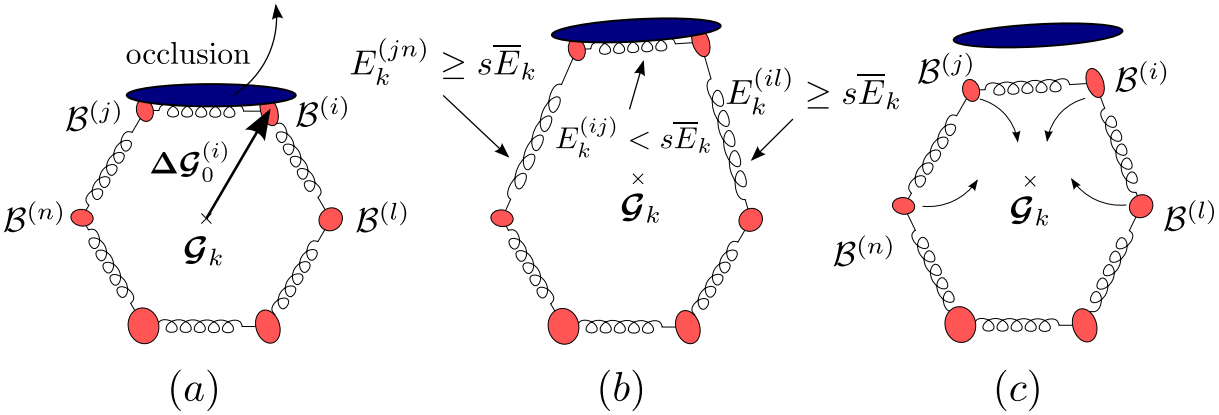



Figure 2.7: (a) The system is correctly tracking an object, until an occlusion occurs. In this case, we will suppose the occlusion to attract two trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$. (b) In this case, $E_k^{(ij)}$ remains stable, and the previous mechanism will not be activated. However, the energy of the rest of connections linking these trackers will grow to be bigger than the energy threshold. (c) If the elastic energy $E^{(ik)}$ of any connection $\mathcal{C}^{(ik)}$ gets bigger than a threshold (defined as proportional to the mean elastic energy) then both of the trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(k)}$ linked by the connection are attracted towards their equilibrium position, relative to the center of mass of the set of feature trackers.

Let $\boldsymbol{\mathcal{G}}_k$ denote the coordinates of the center of mass of the set of trackers at time $t_k$, and let $\boldsymbol{\Delta\mathcal{G}}_0^{(i)} = \boldsymbol{\mu}_0^{(i)} - \boldsymbol{\mathcal{G}}_0$ represent the difference between the center of mass and a generic tracker $\mathcal{B}^{(i)}$ at the initial equilibrium position (see figure 2.7(a)). If the energy $E_k^{(ij)}$ of a connection $\mathcal{C}^{(ij)}$ is bigger than a certain multiple of the mean elastic energy, then we will displace both trackers $\mathcal{B}^{(i)}$ and $\mathcal{B}^{(j)}$ towards their equilibrium position, relative to the current position of the center of mass. In the same way as for the spring-like connections, the displacements applied to the trackers will be proportional to the distance to the equilibrium position (relative, in this case, to the center of mass).

$$\text{if } E_k^{(ij)} \geq s\overline{E}_k \Rightarrow \begin{cases} \boldsymbol{\Delta\mu}_k^{(i)} = \alpha^{(en)}\left(\boldsymbol{\mathcal{G}}_k + \boldsymbol{\Delta\mathcal{G}}_0^{(i)} - \boldsymbol{\mu}_k^{(i)}\right), \\ \boldsymbol{\Delta\mu}_k^{(i)} = \alpha^{(en)}\left(\boldsymbol{\mathcal{G}}_k + \boldsymbol{\Delta\mathcal{G}}_0^{(i)} - \boldsymbol{\mu}_k^{(i)}\right), \end{cases} \tag{2.27}$$

where $\alpha^{(en)}$ is the proportionality factor, equivalent to the stiffness of our spring-like connections. This mechanism is in fact quite similar to that of the spring-like connections. However, there is a fundamental difference: it is just applied when the connections surpass the energy threshold, and its equilibrium distance is defined relatively to the center of mass.

Let us note that the parameter $s$ has an strong impact on the behavior of the system, and it should be chosen carefully. A detailed study of its impact will be discussed in the next section.

### 2.3.4 Remarks

When building the model of an object we are not constrained to a unique configuration. Instead, we can imagine any combination of the three of them and assign them different stiffnesses, in order to obtain the desired behavior. The system is built such that there is no constraint in this sense.

However, we need to be careful when applying the mechanisms described in the previous section. As explained in Appendix B, the stiffness $\alpha$ of our spring-like connections, as well as the masses of the trackers, are nothing but dimensionless scaling factors. This means that, when computing the Elastic Energy of a connection using (2.23), (2.24) or (2.25), the results we are obtaining do not have dimensions of energy. Instead, they simply are a weighted sum of the square of the elongations of each connection. The units of these elongations are pixels for the *Cartesian* and *Euclidean* configurations, and radians for the *Torsional* configuration. As a result, making a comparison between these different types of energy does not have any physical interpretation. Consequently, we will be careful to only compare the same types of energy.

### 2.3.5 Global algorithm

The general algorithm of our method is given below (Algorithm 1).

**Algorithm 1** Global algorithm

**Require:** $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T, \quad \forall k \geq 0$

**Ensure:** $\boldsymbol{\mu}_k^{(i)}, \Sigma_k^{(i)}, \quad \forall i$

```
for every incoming event e_k do
    for every tracker B^(i) do
        Compute the Gaussian probability p_k^(i) using (2.8).
    end for
    Find the best candidate B^(b), with b = arg max_i p_k^(i)
    for every tracker B^(i) do
        Update the activity A_k^(i) using (2.9).
        Decide if the tracker is active, comparing A_k^(i) with A^(up).
    end for
    if p_k^(b) > δ then
        Update μ_k^(b), Σ_k^(b) using (2.1) and (2.5).
    end if
    for every connection C^(ij) do
        Compute the displacements Δμ_k^(i) and Δμ_k^(j) using (2.15), (2.19) or (2.21).
        Update the positions of B^(i) and B^(j) using (2.14).
        Compute the elastic energy E_k^(ij) using (2.23), (2.24) or (2.25).
    end for
    Compute the mean elastic energy Ē_k.
    Apply the energy-based criteria.
end for
```

## 2.4 Results

All the experiments presented in this chapter operate on recordings produced by an ATIS sensor. The tracking method is implemented in C++ and runs on a standard computer.

### 2.4.1 Tracking a planar grid

In this section we present a series of experiments, where the neuromorphic sensor observes a computer screen that displays a moving $3 \times 3$ grid of fixed sized blobs. Distortions are applied to the grid to simulate a free evolution in a 3D space. Fig. 2.8 shows a sample of the stimulus recorded by the neuromorphic camera. The active trackers (in green) are being deformed by incoming events generated by the moving stimulus. The update factors from (2.1) and (2.5) were set to $\omega_0 = 0.02$ and $\psi_0 = 5 \times 10^{-5}$. The inactive trackers are represented in black. The position of the springs is represented by the grid's edges, connecting two neighboring circles together. The stimulus is updated every 10 ms (100 fps) allowed by the technology of the used screen.

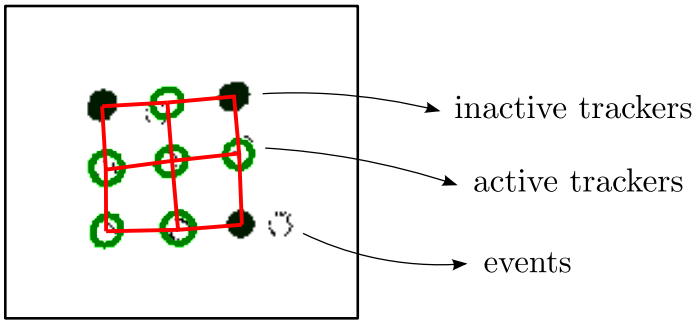



Figure 2.8: Snapshot created from the output of the neuromorphic camera showing the state of the trackers for a configuration of the moving grid. The snapshot shows events happening over a time period of 10 ms. Active trackers are represented by green ellipses while the inactive ones are shown in solid black. The overimposed lines show the connections established between the trackers.

In the first experiment, the grid is displayed while alternating a rotation around the vertical axis with a range of $-75°$ and $+75°$ at a constant speed of $15°/s$. A vertical movement has been added, in order to generate events in all directions. Cartesian connections are set between the trackers, and the effect of their stiffness $\alpha$ tested. Fig. 2.9(a) shows the state of the system at four temporal locations, for two different values of the stiffness $\alpha$. When $\alpha = 0.004$, the set of trackers is capable of following the grid up to a large angle until a subset of trackers (shown in black) are unable to follow the target.

Fig. 2.9(b) shows the maximum tracking angle as a function of the chosen stiffness. It is, as expected, a decreasing relation: when the stiffness of the connections is increased, the trackers cease the tracking at a much earlier stage. Choosing the right $\alpha$ requires a trade-off between adaptability to the distortions of the scene and robustness to disturbances.

In the second experiment, the grid is shown rotating around the $X$ and $Y$ axes simultaneously, at $15°/s$ and $10°/s$ respectively. The boundary values of each angle are set to $-30°$ and $30°$ for the $X$ axis, and $-45°$ and $45°$ for the $Y$ axis. We added vertical and horizontal motions as in the previous case. We also added random disturbances to the stimulus: an element of the grid is artificially moved away to test the reaction of the system to deformable objects. This also allows the simulation of occlusions that usually

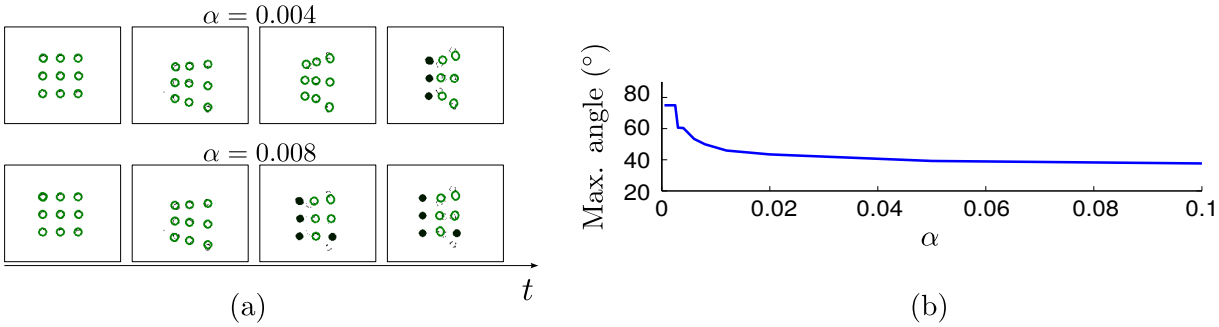



Figure 2.9: (a) Results of the experiment for two different values of the stiffness: $\alpha = 0.004$ and $\alpha = 0.008$ (b) The maximum tracking angle as a function of $\alpha$.

cause a subset of the trackers to follow the wrong cloud of events. Fig. 2.10 shows six snapshots of the stimulus, in which an element of the grid is separated and then returns to its position according to the applied motion.

Figure 2.10: Snapshots of the stimulus video: as the grid keeps following projective transformations, one of its blobs is taken apart.

The ground truth is computed from the positions of the points on the screen, a homography is estimated between the screen and the focal plane of the neuromorphic camera. The error of the element of the grid simulating an occlusion is computed by comparing its current position to its desired one (the one it would have been located at if the artificial elongation was not applied).

Fig. 2.11(a) shows the tracking results of a single tracker, with $\alpha = 0.001$ and $s = 4.5$. Fig. 2.11(b) shows the moment at which one element of the grid starts being pulled from the rest. At the beginning, the tracker successfully follows the element (from (b)-(d)). As the element keeps moving further away from the rest of the grid, the attraction of the springs is bigger than the attraction due to the events. As a result, the tracker will stop following this cloud of events, as shown in Fig. 2.11(e). After that, it quickly recovers its relative position to its neighbors. As shown in Fig. 2.11(g), the equilibrium position to which the point returns to will be close to the real position of the element in the scene.

We define the instantaneous error $\xi_k$ at time $t_k$ as the mean error over all the trackers. We then characterize the accuracy of the system by the mean of this error computed over the whole recording, and expressed as a percentage of the characteristic length of the object:

$$\overline{\xi}(\%) = \frac{100}{LK} \sum_{k=0}^{K} \xi_k, \tag{2.28}$$

where $K$ denotes the total number of events, and $L$ the characteristic length of the considered object.

Fig. 2.12 shows the evolution of the mean error $\overline{\xi}$ with the value of $s$, for a fixed value of $\alpha = 0.001$ (let us remind the reader that the parameter $s$ represents the scaling factor of the mean elastic energy, in order to activate the energy-based criteria). The best value obtained is $\overline{\xi} = 2.74\%$ for $s = 4.5$. The error is stable for $s \geq 12$, because above this

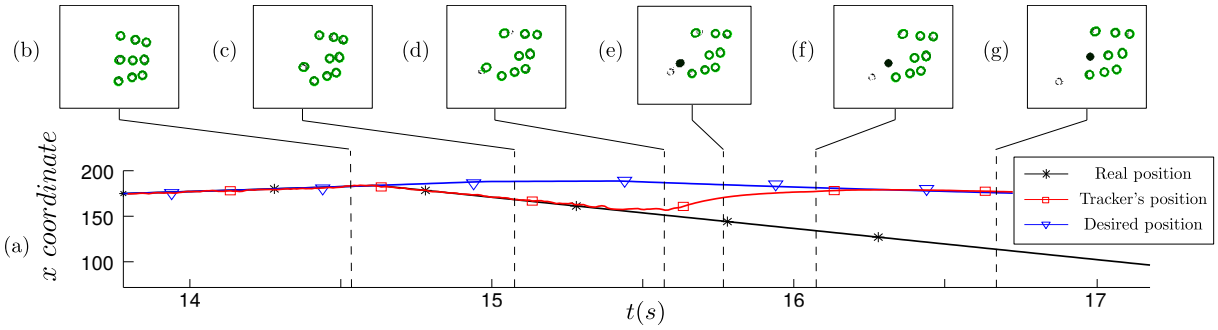



Figure 2.11: (a) Position in $x$ of a blob of the grid. The real position is known from the scene generation process. The second line shows the position of the tracker following the blob. Finally, the desired position represents the position where the tracker should be if the grid was not deformed (e.g. pulled away from the grid structure). As we can observe, the tracker initially follows the blob in its movement away from the grid. However, once the deformation goes beyond some threshold, the tracker recovers its equilibrium position.

value the energy threshold gets so high that no connection overpasses it. As a result, the mechanisms defined in Section 2.3.3 are never activated. This figure shows the positive effect of the energy criteria on the performance of the system. This shows that one can use use small values for the stiffness, allowing the system to be robust and thus efficiently tracking the moving target.

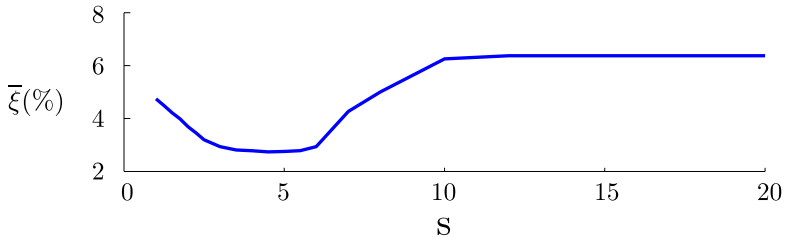



Figure 2.12: Evolution of the mean error with the value of s. This parameter is a proportionality factor, that sets the value of the threshold for the energy-based criteria defined in Section 2.3.3 to be activated. When $s$ is too large, these criteria are never activated leading to high tracking errors.

We repeat the same experiment for different values of the stiffness $\alpha$ and evaluate the error produced. Table 2.1 (left) shows the minimum error for each tested value of $\alpha$ when the cartesian configuration is selected and the corresponding value of $s$ at which it is obtained. The smaller error is obtained for $\alpha = 0.001$ and $s = 4.5$.

In order to compare the tracking performance of the different spring configurations, we repeat the same experiment connecting neighboring trackers by both a euclidean and a torsional spring, imposing $\alpha$ to be the same for both of them. Table 2.1 (right) shows the tracking error for this configuration. In this case, the smaller error is also obtained for $\alpha = 0.001$ and $s = 4.0$.

Fig. 2.13 shows the evolution of the tracking errors with the stiffness for both types of connections. We can see that the cartesian configuration slightly outperforms the combination of torsional and euclidean springs for every value of the stiffness.

Another experiment is carried out to test the reaction of the system to rotations of the tracked object on the image plane. Here, the grid rotates around the normal to the screen between 45° and -45° while it translates in the focal plane. The tracking errors are

| | Cartesian | | Euclidean + torsional | |
|---|---|---|---|---|
| $\alpha \times 10^5$ | $\min(\bar{\xi})$ | s | $\min(\bar{\xi})$ | s |
| 50 | 2.93 | 3.5 | 3.01 | 3.0 |
| 100 | 2.74 | 4.5 | 2.79 | 4.0 |
| 150 | 2.84 | 4.5 | 2.88 | 4.5 |
| 200 | 3.36 | 4.5 | 3.75 | 3.5 |
| 400 | 4.30 | 6.0 | 4.84 | 3.5 |
| 800 | 4.79 | 6.0 | 5.46 | 3.0 |

Table 2.1: Mean tracking error produced for different values of the stiffness.

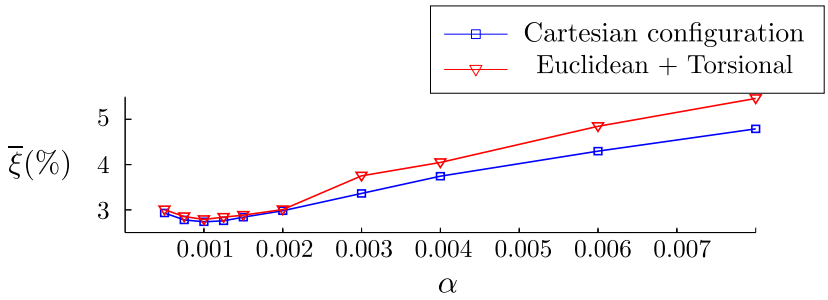



Figure 2.13: Tracking error obtained as a function of the stiffness. The grid experiments general deformation and motion, and the cartesian configuration slightly outperforms the combination of a torsional and a euclidean connection.

computed for both cartesian connections and a combination of torsional and euclidean connections. Fig. 2.14 shows the tracking error with respect to the springs parameters for the rotating grid. In such case, we can observe how the euclidean + torsional configuration is clearly more suited: the error is steady and much lower than in the case of the cartesian configuration, for which it increases with $\alpha$.

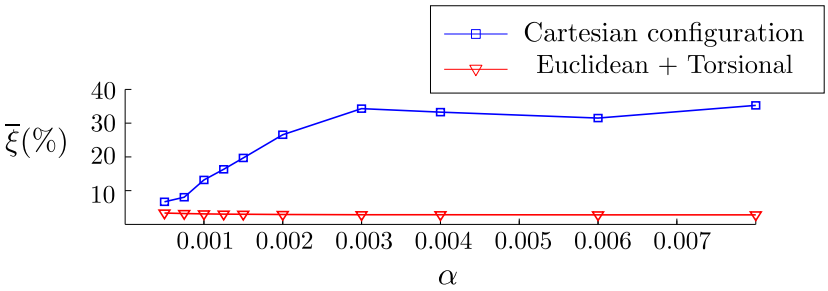



Figure 2.14: Tracking error as a function of the stiffness, when the grid experiments a rotation on the image plane. In this case, the combination of torsional and euclidean connections clearly outperforms the cartesian configuration.

This allows us to conclude that the cartesian configuration guarantees more robustness in keeping the desired shape. However, the system becomes sensitive to rotation on the image plane.

### 2.4.2 Face tracking

The second experiment tests the tracking technique on a real human face in an indoor environment. The target is a moving face as shown in Fig. 2.15. The motion of the face is subjected to complex dynamics including phase of steep acceleration changes and scale variations as the target is waving and moving towards the camera.

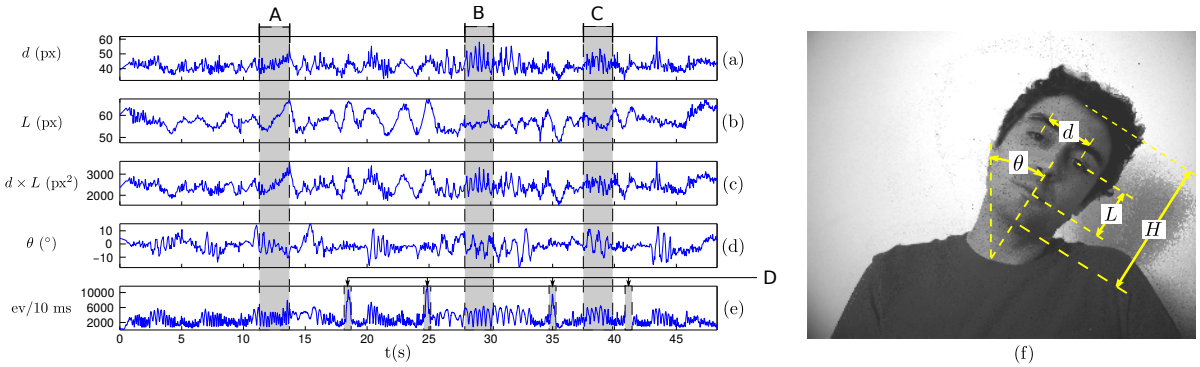



Figure 2.15: Geometric parameters observed during the sequence. (a-b) $d$ and $L$ are the distances (in pixels) between the two eyes and between the eyes and the mouth. (c) $d \times L$ represents roughly the area of the face in the image. Its value increases when the face gets closer to the camera: the section tagged as A, where both $d$ and $L$ increase, shows such a typical case. Changes in $d$ for a constant value of $L$ (or viceversa) correspond to rotations of the face around the X or Y axis. This is represented by the section tagged as B. Rapid oscillations around the yaw axis of the head are producing such results. (d) $\theta$ is the angle (in degrees) between the vertical axis and the central axis of the face (or the roll angle of the head). (e) The number of incoming events is directly related to the dynamics of the scene. The number of events is shown on a time scale of ten milliseconds. The temporal locations of high number of events shown by D are the result of occlusions generated when moving hands cover partially the face (see Fig. 2.17). (f) Snapshot extracted from the grayscale output of the ATIS camera, illustrating these parameters.

To characterize this complex sequence, several measurements are defined and measured during the sequence: $d$ is the distance between the two eyes, $L$ is the distance of the eyes to the mouth and $\theta$ is the angle between the vertical axis and the central axis of the face (usually known as the roll head angle). Fig. 2.15(left) shows the measured values of each parameter for the entire sequence. The number of events for a binning of 10 ms is also shown in Fig. 2.15(e). The labels A, B, C and D shown in the Fig. 2.15 outline interesting temporal locations of the experiment.

A moving face was recorded by the ATIS camera and we produced a snapshot from the graylevel output from which we handcrafted a set of trackers and the connections between them. The initial size and orientation of the Gaussian trackers can be chosen by modifying their covariance matrix, so they better adjust to the corresponding element of the face. Fig. 2.16 shows the actual mask used in the experiment, composed of 26 blob trackers (two for the eyebrows, two for the eyes, two for the nostrils, one for the mouth and nineteen to create the outline of the face), joined together by 40 connections. For every connection between a pair of trackers (shown in Fig. 2.16 as the thin lines between the ellipses) we chose to impose both a euclidean and a torsional connection. Their equilibrium distance and angle were computed from their initial positions, and the stiffness $\alpha$ was experimentally set to 0.02 for both mechanisms in every connection. The

update factors from (2.1) and (2.5) were set to $\omega_0 = 0.2$ and $\psi_0 = 0.0002$. The value of $s$ was tested in the same way as in the previous experiment, and set to 2.5. During the recording, we asked the subject to wave his hand in front of his face to introduce occlusions. The total time of the recorded stimulus is 49 seconds.

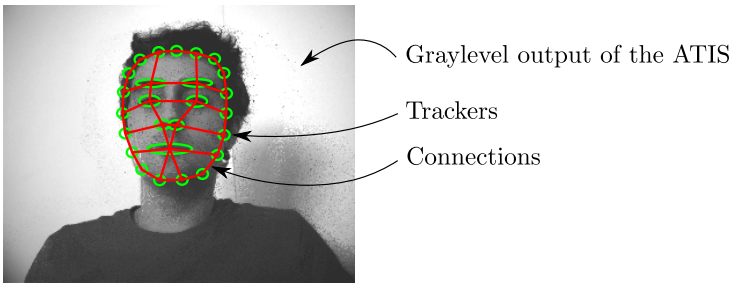



Figure 2.16: The set of trackers and the structure of their connections used to follow a face from incoming events. The ellipses show the position of the trackers, and the lines the connections set between them. Each connection is a combination of a euclidean and a torsional connection, with $\alpha = 0.02$ for both connections.

Fig. 2.17 shows the state of the system whilst tracking a face. It shows how the system reacts to partial occlusions. As the hand passes in front of the face, it first attracts the trackers, displacing them from their correct position. The system is sufficiently robust to compensate by attracting the trackers to the correct position again, without losing track of the face.

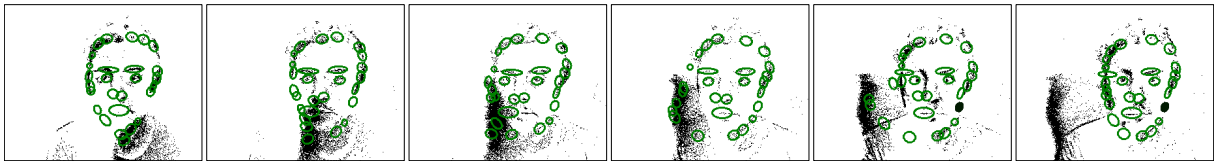


Figure 2.17: The set of connected trackers are disturbed by a dynamic occlusion introduced by waving a hand in front of the face. As the hand passes in front of the face, it first attracts the trackers, displacing them from their right position. However, the system is sufficiently robust to compensate by attracting the trackers to the right position again, without loosing track of the face.

The ground truth is obtained by manually selecting seven points of the face: eyebrows, eyes, nostrils and mouth. The error is defined in the same way as in the previous experiment: first, we define the error of each individual tracker as the distance (in pixels) between the position of the tracker and the position of the corresponding feature in the image plane. Next, we define the error for each instant as the mean of the errors of the ground truth locations. It is expressed as a percentage of the vertical length of the face. Fig. 2.18 shows the temporal evolution of the error. We characterize the system by the temporal mean of this error, which is equal for this experiment to 5.42%.

### 2.4.3 Computational Time

Let us next evaluate the computational time required by the current C++ implementation of the algorithm. These tests were performed in a standard computer running Debian Linux, equipped with a Intel Core i7-4790 processor. The code was not paralellized and just one core was used.

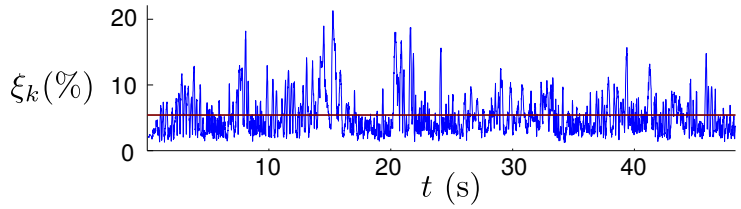



Figure 2.18: Temporal evolution of the error. This error is equal to the mean error of the seven internal trackers (eyebrows, eyes, nostrils and mouth), relative to the vertical length of the mask. Its temporal mean (indicated by the horizontal line) is equal to 5.42%

In order to characterize the computational time required by the algorithm we measure the time it takes to process the previously presented recordings. To that end, we measure the processing time for every time period of one millisecond (without overlapping) and compute the ratio of processing time to the length of the considered periods (i.e. the number of milliseconds it takes to process one millisecond of events). Thus, if this ratio is smaller than 1, the algorithm can process the corresponding event stream faster than it is acquired, without increasing latency (i.e. in "real time"). In order to obtain stable values we process each recording 10 times and average the obtained results.

Let us show in Fig. 2.19 the ratio of processing time to the length of the recording for the face tracking task, with the same set of parameters given in Section 2.4.2. Let us note that we are considering the processing time required by the whole processing chain. This includes, in addition to the shape tracking, reading the recording file and processing the events for noise removal.

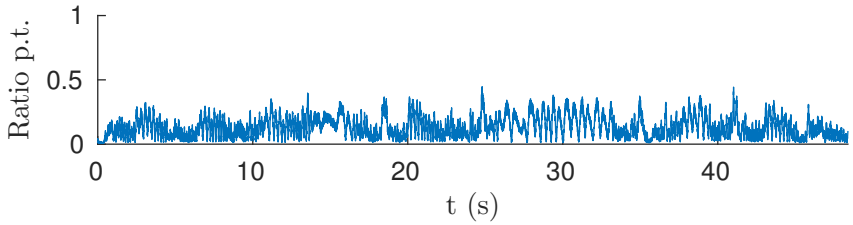



Figure 2.19: We measure the processing time every ms and compute the ratio of processing time to the length of the considered portion of the recording. We show the results obtained for the face tracking task: since this ratio is always smaller than 1, we can conclude that we are processing the events faster than they are acquired for this recording.

We verify that the ratio of processing time to the lenght of the recording is always smaller than one, which allows us to conclude that we can process this stream of events faster than it is acquired. These results are of course dependent on the precise implementation of the algorithm and the computational power available. However, we consider them to be useful as a means of comparison. The computational time will also be influenced by the following factors:

- The complexity of the object: more detailed objects with a greater number of trackers and connections will require more computations.
- The event rate: a greater number of events to process per unit of time will necessary increase the computational time required.

According to this last point, it is more interesting to display the ratio of processing time as a function of the event rate, as in [71]. We show this results in Fig. 2.20: we verify that the computational time is increasing with the number of events, and can be approximated by a linear function. This allows us to extrapolate and compute the maximum rate of events that can be processed in real time. We obtain, for this object and set of parameters, a maximum event rate of 2076 ev/ms. Let us note that this is a big value for the event rate, not usually reached in standard conditions (as a reference, the recording of the face has a mean event rate of 303.4 ev/ms).

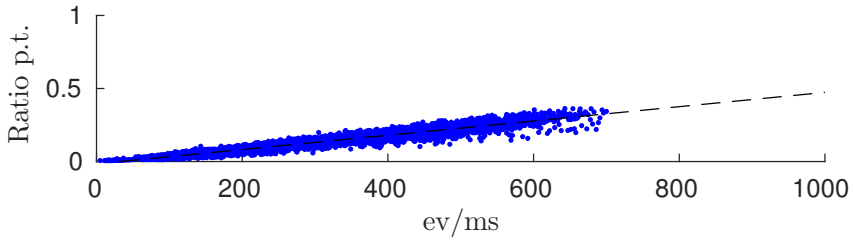



Figure 2.20: Computational time per ms as a function of the number of events per ms. The computational time grows with the event rate, and it can be approximated by a linear function.

## 2.5 Discussion

This chapter introduces a new method for visual tracking of complex objects using neuromorphic vision and the time of arrival of events. This work introduces for the first time a true real-time part-based model running at the native resolution of scenes' dynamics and updating its energy at several hundreds kHz.

Experiments are provided showing the operation of the method. We first assess the accuracy of the algorithm in a controlled environment, where we obtain tracking errors below 3% (relative to the characteristic length of the tracked object). Then, as a demonstration of its possible uses, we apply the algorithm to a face tracking task showing its accuracy and robustness.

As we model the behavior of our virtual mechanical system, its tends to its minimum. This is an important property, that will be further studied in Chapter 5. The method presented in the current chapter uses the energy as a quality criterion for the tracking. When correctly tuned, the energy-based criteria increase the performance of the method, making it more robust to partial occlusions.

In the current state, the model of the object needs to be manually defined. Future improvements of the algorithm include the introduction of learning techniques to determine the model of the object: the pool of trackers would then be connected based on their temporal coactivations, thus making use of the high temporal resolution of the sensor. A possible improvement of the system would be to use more specialized trackers, allowing them to track specific space-time patterns rather than just following incoming blobs of events. This should make the method more robust to occlusions as it is very improbable that occlusions have the same shape as the tracked stimulus.

The part-based shape tracker presented in this chapter represents an advance with respect to the previously available event-based tracking techniques. While it is designed to track complex objects, numerous computer vision algorithms require the extraction of

some low-level feature from the scene. This motivates the development of the line and segment detector introduced in the next chapter.

# Chapter 3

# Event-Based Line and Segment Detection

## 3.1 Introduction

In computer vision, line detection is the task of identifying lines along straight edges present in a visual scene, and estimating their parameters. In man-made environments, lines (and segments) provide fundamental low-level features for vision-based systems and they are frequently used in robotic visual odometry [35] and vision-based structure from motion (SFM) [36], [72]. More recently, the two problems are combined into the larger application of the Simultaneous Localization and Mapping (SLAM) [73]-[74]. Various approaches for the detection of lines and segments have been proposed since the early age of computer vision: one of the most known and successful algorithms is the Hough transform [75], that can be generalized to the detection of generic shapes [76]. It is still widely used and studied as shown in in the recent State-of-Art proposed in [77]. Other classical approaches include gradient-based methods [78] or statistical-based techniques [79]. In the field of segment detection, LSD [80] is one of the most popular algorithms. However, neither of these methods applies to the stream of asynchronous events provided by a neuromorphic camera, where the information is not conveyed by frames.

A fundamental step leading to higher level processing of the event-based visual information is to define, detect and recognize accurately low-level features. The low-level event-based features are a difficult concept, costly to build and extremely velocity dependent. Lately, event-based corner detectors have been proposed [81]-[83]. An event-based Hough Transform implemented with a Spiking Neural Network has also been introduced in [84]. This approach offers a limited robustness to velocity changes as only a linear decay of the activity of the neurons is considered. As pointed out in [83], the decay function weighting the contribution of an event must be tuned according to the expected velocities of the observed objects on the focal plane. In spite of this precaution, small structures such as corners are inherently velocity dependent: when the motion is parallel to one of the corner's edges, the corner is no longer seen by the sensor. To alleviate the velocity dependence, we propose rather to track larger structures like lines. Because of their size, lines are seen for a longer time and are more easily detected by the sensor. Finally, using lines as visual features is also a reasonable constraint since human made structures are often made of rectilinear edges.

An event-based segment detector has been published in [85], called Event-Based Line Segment Detector (ELiSeD) and derived from the LSD [80]. The reported accuracy, when

tracking a single line, is about 1.36 pixels compared with the original LSD algorithm. The ELiSeD method requires events to be stored on a circular buffer of a fixed size (they report typical sizes of 2500 to 8000 events). As a result, fast moving objects, that generate globally more events at constant contrast, tend to dominate the buffer contents.

To address the problem of robustness to velocity changes, we propose an alternative event-based line and segment detector: the parameters of a given line are obtained by weighted least squares fitting of the events attributed (i.e. spatio-temporally close) to the line. The weight of the past events follows a speed-tuned exponentially decaying function, which makes the method velocity-independent, as shown in [83]. Parameters of a line are iteratively updated with each event in a truly event-based fashion.

This line detection technique can be refined into a segment detection algorithm in order to provide pixel-wise feature detection. The main reason to develop this segment detector is twofold: the first is that many computer vision algorithms require pixel-wise feature detection to achieve feature identification and tracking (e.g. [86]). The second is that the existing event-based pixel-wise feature detectors are based on corners which, as stated before, are extremely sensitive to velocity change. Corners simply cannot be seen by the sensor if the motions are perpendicular to the contours that subtend the corners. As we claim that lines are more robust (but not immune) to such problem thanks to the speed-tuned decay approach, we localize segment endpoints through the activity analysis along the detected lines. The validity and the accuracy of the endpoint detection are validated by the experiments, showing that they can be used for pixel-wise feature detection.

Our approach results in a real-time algorithm iteratively estimating line parameters as exposed in Section 3.2. The approach is then extended to the detection of segments in Section 3.3. In Section 3.4, we provide experiments assessing the accuracy of our algorithm in a controlled environment. Visual results obtained on non controlled outdoor and indoor real scenes are also provided. Finally, we briefly discuss the obtained results in Section 3.5.

## 3.2 Event-Based Line Detection

Let us consider a stream of events generated by a neuromorphic camera. Let us remind the reader that an event is defined as a quadruplet $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T$, which represents a change of luminance (increase, $p_k = 1$, or decrease, $p_k = -1$) occurring at time $t_k$ at the position $\boldsymbol{u}_k = [x_k, y_k]^T$ on the focal plane. Here, we will be looking for the set of lines that best fits the stream of recent past events, in the sense of minimizing the sum of the squared distances from the events to the lines.

As the first events arrive, we initialize some *line models* passing through them. Then, as more events are detected, we can either assign them to some pre-existing model or generate a new one. This process results in the generation of a certain number of line models, many of which will not be reliable as they are estimated from few events. These line models are initially marked as *inactive*, meaning that we keep updating the models but the number of events supporting them has not reached a sufficient high value yet. A line model switches from label *inactive* to *active* if, within a short period of time, a sufficient number of events has contributed to the update of the model.

We impose two conditions for an event to be assigned to the update of a line model:

- The distance between the event and the line has to be smaller than a threshold.
- The *visual flow* of the event [32] must be perpendicular to the line.

The first condition is trivial, since the event is spatio-temporally related to the line. The second condition is also a logical one, because the visual flow is the event-based equivalent of the optical flow estimated in frames, i.e. the pattern of apparent motion of objects in a visual scene, caused by their motion relative to the camera (the specific naming is to emphasize its event-based nature: the visual flow is not estimated from the standard luminance consistency criterion). Thus, the visual flow estimation is subject to the same aperture problem explained in [87]: when the flow is computed locally, only the normal component to the local contours of the object can be obtained. Consequently, the direction of the flow is encoding the direction of rectilinear edges, i.e. lines on the focal plane. The proposed event-based line detection technique then fits line parameters according to these two constrains upon arrival of each event.

### 3.2.1 Event-Based Visual Flow

The first step to track lines from events is to compute the visual flow of the incoming events. To that end, we apply the technique described in [32], where the visual flow is obtained from the normal to a 3D plane locally approximating the spatio-temporal surface described by the incoming events. In the standard plane parametrization of $Ax + By + t + C = 0$, the normal to a plane defined by data $[x, y, t]^T$ is directly related to the parameters $A$ and $B$. To estimate these parameters we apply least squares fitting. This is done on the centered data to avoid ill conditioned systems [88]. The plane equation is then equivalent to:

$$\widetilde{t} = A\widetilde{x} + B\widetilde{y}, \tag{3.1}$$

where the tilde means that the average values are subtracted from each variable. Here, we consider the $m$ most recent events in a given spatial neighborhood of the current event $\mathbf{e}_k$ (as in [81]), and we denote $\mathcal{I}_k$ the set of indices identifying these events (i.e. if $i \in \mathcal{I}_k$ then $\mathbf{e}_i$ is one of the $m$ most recent events in a spatial neighborhood of $\mathbf{e}_k$). We obtain the following linear system:

$$\begin{bmatrix} \sum\limits_{i\in\mathcal{I}_k} \widetilde{x}_i^2 & \sum\limits_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{y}_i \\ \sum\limits_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{y}_i & \sum\limits_{i\in\mathcal{I}_k} \widetilde{y}_i^2 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} \sum\limits_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{t}_i \\ \sum\limits_{i\in\mathcal{I}_k} \widetilde{y}_i\widetilde{t}_i \end{bmatrix}. \tag{3.2}$$

This being a $2 \times 2$ linear system, when the system matrix is invertible we can apply Cramer's rule to obtain a closed form solution, allowing us to efficiently compute $A$ and $B$. Since the visual flow estimation is subject to the aperture problem, only the normal velocity vector can be obtained, which is computed as:

$$\boldsymbol{v}_k = \frac{1}{A^2 + B^2} \begin{bmatrix} -A \\ -B \end{bmatrix}. \tag{3.3}$$

The output of this first step will be a stream of "oriented" events $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]$, where $\boldsymbol{v}_k$ denotes the normal flow of the corresponding event. The oriented event is then the initially defined event augmented with the velocity $\boldsymbol{v}_k$.

### 3.2.2 Event-Based Least-Squares Line Fitting

With the flow, next comes the event-based algorithm for least squares line fitting. This method is an iterative event-based adaption of the classical least squares fitting of lines with perpendicular offsets [89].

### Line Parametrization

Lines in 2D space are defined by two parameters. We choose here the $\rho$-$\theta$ parametrization [90], which avoids the infinite or close to infinite slope that one can meet with the slope-intercept parametrization [91]. Line models are identified by their index $i$ and denoted $\mathcal{L}^{(i)}\big(\rho_k^{(i)}, \theta_k^{(i)}\big)$, where $\rho_k^{(i)}$ is the distance from the line to the origin and $\theta_k^{(i)}$ the angle between the normal $\boldsymbol{n}_k^{(i)}$ to the line and the horizontal (see Fig. 3.1). Following the standard notation in this document, the subindex $k$ relates the line to the time, i.e. $\rho_k^{(i)}$ is the angle of line $\mathcal{L}^{(i)}$ at time $t_k$, etc. The equation of the line is then:

$$x\cos(\theta_k^{(i)}) + y\sin(\theta_k^{(i)}) - \rho_k^{(i)} = 0, \tag{3.4}$$

where $\theta_k^{(i)} \in [-\pi/2, \pi/2]$. The unit vector normal to the line $\boldsymbol{n}_k^{(i)}$ is then given by:

$$\boldsymbol{n}_k^{(i)} = \big[\cos(\theta_k^{(i)}),\ \ \sin(\theta_k^{(i)})\big]^T. \tag{3.5}$$

Let us note here that, while the lines are defined by the angle $\theta_k^{(i)}$, we will not need to calculate it explicitly. The algorithm can be applied directly on $\cos(\theta_k^{(i)})$, $\sin(\theta_k^{(i)})$, as it will be shown in Section 3.2.3. This constitutes an optimization strategy that reduces the number of required computations, as it will be explained in Section 3.2.3.

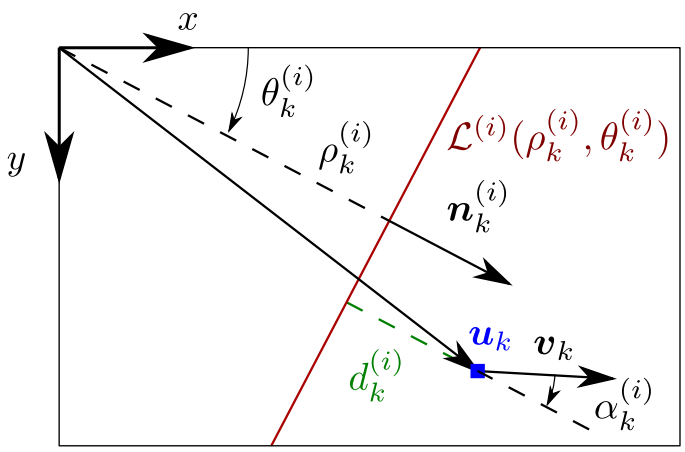



Figure 3.1: A line, identified by its index $i$, is denoted $\mathcal{L}^{(i)}\big(\rho_k^{(i)}, \theta_k^{(i)}\big)$ and defined by two parameters: the distance $\rho_k^{(i)}$ between the line and the origin and the angle $\theta_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the horizontal. Incoming events are assigned to previously existing lines based on two conditions: the euclidean distance $d_k^{(i)}$ between the line and the position of the event $\boldsymbol{u}_k$, and the angular distance $\alpha_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the visual flow of the event $\boldsymbol{v}_k$.

### Activity

Like the gaussian trackers in the previous chapter, each possible line model has a certain level of activity, that we denote $\mathcal{A}_k^{(i)}$. The activity of every model is updated with the

incoming events $\mathbf{e}_k$, following a speed-tuned exponential decay function:

$$\mathcal{A}_k^{(i)} = \begin{cases} \mathcal{A}_{k-1}^{(i)} e^{-\|\boldsymbol{v}_k\|\Delta t_k} + 1 & \text{if } \mathbf{e}_k \text{ is assigned to } \mathcal{L}^{(i)}, \\ \mathcal{A}_{k-1}^{(i)} e^{-\|\boldsymbol{v}_k\|\Delta t_k} & \text{otherwise,} \end{cases} \tag{3.6}$$

where $\Delta t_k = t_k - t_{k-1}$. Let us remind the reader that in the previous chapter the activity followed a fixed decay exponential (see Section 2.3.1 for details). Here, we choose a speed-tuned decreasing strategy instead (as in [83]), which makes the activity of the lines independent to their respective velocities. We consider this to be an improvement with respect to the fixed decaying strategies, as it provides an automatic and adaptive way to characterize each line.

As in the previous chapter, if the activity $\mathcal{A}_k^{(i)}$ of a line model $\mathcal{L}^{(i)}$ is greater than a predefined threshold $\mathcal{A}^{(up)}$, then $\mathcal{L}^{(i)}$ is said to be *active*, and a line is assumed to be actually present at that position.

### Assignment of events

Let $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ be an oriented event occurring at time $t_k$ at the position $\boldsymbol{u}_k$ on the focal plane, with normal flow $\boldsymbol{v}_k$. The assignment of $\mathbf{o}_k$ to a line is based on two criteria:

- The euclidean distance $d_k^{(i)}$ from the event to the line has to be smaller than a threshold $d^{(max)}$.
- The angle $\alpha_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the visual flow $\boldsymbol{v}_k$ must be smaller than a threshold $\alpha^{(max)}$ (see Fig. 3.1), in order to assure orthogonality of the flow to the line.

The two criteria are translated into the following inequalities that an event has to satisfy in order to be assigned to a line $\mathcal{L}^{(i)}$:

$$\begin{cases} d_k^{(i)} = \left| \boldsymbol{u}_k^T \boldsymbol{n}_{k-1}^{(i)} - \rho_{k-1}^{(i)} \right| < d^{(max)}, \\ \left| \cos(\alpha_k^{(i)}) \right| = \left| \boldsymbol{v}_k^T \boldsymbol{n}_{k-1}^{(i)} \right| > \cos(\alpha^{(max)}). \end{cases} \tag{3.7}$$

Here, we directly compare the cosinus of the angle, which yields an equivalent result and avoids the computation of an arc cosinus. If several line models verify these two conditions, a competition mechanism is established and the event is assigned to the model with the highest activity. If none of the existing models verifies them, we initialize a new one.

### Initialization of line models

If an oriented event $\mathbf{o}_k$ cannot be assigned to any of the pre-existing line models, we generate a new model. The parameters of this new model, indexed by $i$, are determined by the event $\mathbf{o}_k$ in the following form:

$$\begin{cases} \boldsymbol{n}_k^{(i)} = \boldsymbol{v}_k, \\ \rho_k^{(i)} = \boldsymbol{u}_k^T \boldsymbol{v}_k \end{cases} \tag{3.8}$$

To limit the computational time required by the algorithm we fix a maximum of $N$ line models to be tracked simultaneously. If $N$ line models have already been created, then the new one replaces the model among the $N$ with the smallest activity.

### Optimal parameters

When an event $\mathbf{o}_k$ is assigned to a line model, the parameters of this model are updated accordingly. The new optimal parameters must minimize the sum of the squared distances between the line and the past events assigned to it. As in [86], these distances are weighted in order to give a greater importance to the most recent events. Without loss of generality, let us assume that all past events have been assigned to the same line $i$. Hence, the function $E_k^{(i)}(\theta, \rho)$ is obtained as the weighted mean of the errors committed for the past events assigned to the line:

$$E_k^{(i)}(\theta, \rho) = \frac{1}{\Omega_k} \sum_{j=0}^{k} \omega_{k,j} \big(x_{k-j} \cos(\theta) + y_{k-j} \sin(\theta) - \rho\big)^2, \tag{3.9}$$

where we are applying the formula of the weighted mean previously introduced in Section 2.3.1 and detailed in the Appendix A.

To minimize $E_k^{(i)}$ w.r.t. $\rho$ and $\theta$, we look for the two values that cancel the respective partial derivatives of $E_k^{(i)}$. As shown in the Appendix C, the optimal $\rho_k^{(i)}$ is then obtained as:

$$\rho_k^{(i)} = \frac{\widehat{x}_k^{(i)} \cos(\theta) + \widehat{y}_k^{(i)} \sin(\theta)}{\Omega_k}, \tag{3.10}$$

while $\theta_k^{(i)}$ is deduced after simplifying the vanishing derivative into the equation:

$$a_k \sin\left(2\theta_k^{(i)}\right) + b_k \cos\left(2\theta_k^{(i)}\right) = 0, \tag{3.11}$$

where:

$$\begin{aligned} a_k &= \Omega_k \big(\widehat{yy}_k^{(i)} - \widehat{xx}_k^{(i)}\big) + \big(\widehat{x}_k^{(i)}\big)^2 - \big(\widehat{y}_k^{(i)}\big)^2, \\ b_k &= 2\big(\Omega_k \widehat{xy}_k^{(i)} - \widehat{x}_k^{(i)} \widehat{y}_k^{(i)}\big). \end{aligned} \tag{3.12}$$

Here, $\widehat{x}_k^{(i)}$, $\widehat{y}_k^{(i)}$, $\widehat{xy}_k^{(i)}$, etc. denote the weighted sum of the corresponding coordinates of the events previously assigned to the line:

$$\begin{cases} \widehat{x}_k^{(i)} = \displaystyle\sum_{j=0}^{k} w_{k,j} x_{k-j}, & \widehat{y}_k^{(i)} = \displaystyle\sum_{j=0}^{k} w_{k,j} y_{k-j} \\ \widehat{xx}_k^{(i)} = \displaystyle\sum_{j=0}^{k} w_{k,j} x_{k-j}^2, & \widehat{yy}_k^{(i)} = \displaystyle\sum_{j=0}^{k} w_{k,j} y_{k-j}^2 \\ \widehat{xy}_k^{(i)} = \displaystyle\sum_{j=0}^{k} w_{k,j} x_{k-j} y_{k-j}. \end{cases} \tag{3.13}$$

We will refer to these values as the *auxiliary parameters* of a line, which are required to compute its optimal $\rho$, $\theta$. Let us note that (3.11) yields two possible solutions for $\theta_k^{(i)}$ corresponding to two perpendicular lines, namely the ones maximizing and minimizing

the error. The minimum error is obtained when the second derivative is greater than zero, which yields the following condition:

$$\cos\left(2\theta_k^{(i)}\right)\left(\widehat{yy}_k^{(i)}-\widehat{xx}_k^{(i)}\right)-2\sin\left(2\theta_k^{(i)}\right)\widehat{xy}_k^{(i)}+\rho_k^{(i)}\left(\widehat{x}_k^{(i)}\cos\left(\theta_k^{(i)}\right)+\widehat{y}_k^{(i)}\sin\left(\theta_k^{(i)}\right)\right)>0. \quad (3.14)$$

Alternatively, and thanks to the high temporal resolution of the neuromorphic camera, we can consider that the angle $\theta_k^{(i)}$ will change smoothly at each iteration step. Consequently, among the two possible solutions to (3.11), we can simply choose the one closer to the previous value $\theta_{k-1}^{(i)}$.

The development so far is independent of the weighting strategy being used. In this chapter, we will choose the weights to follow a speed-tuned exponentially decaying strategy as in [83]:

$$w_{k,j}=\begin{cases}\displaystyle\prod_{i=0}^{j-1}e^{-\|\boldsymbol{v}_{k-i}\|\Delta t_{k-i}}=e^{-\|\boldsymbol{v}_k\|\Delta t_k}\omega_{k-1,j-1}, & \text{if } 0<j\leq k\\ 1, & \text{if } j=0.\end{cases} \quad (3.15)$$

According to this strategy, similar weights (close to 1, because $(t_k-t_{k-j})\simeq 0$) will be associated to events $\mathbf{e}_j$ (quasi)-simultaneously occurring at the last event's timing $(t_k)$ and belonging to the current line. The weights of older events, corresponding to older locations of the line, will rapidly tend to 0. The speed-tuning automatically adapts the decay according to the speed of the contour line.

As in the Equation (3.6) about activity, this weighting strategy allows us to compute the auxiliary parameters in the following iterative form, where $\delta_k=e^{-\|\boldsymbol{v}_k\|\Delta t_k}$ (see Appendix E for a complete proof):

$$\begin{cases}\widehat{x}_k^{(i)}\approx\delta_k\widehat{x}_{k-1}^{(i)}+x_k\\ \widehat{y}_k^{(i)}\approx\delta_k\widehat{y}_{k-1}^{(i)}+y_k\\ \widehat{xx}_k^{(i)}\approx\delta_k\widehat{xx}_{k-1}^{(i)}+x_k^2\\ \widehat{yy}_k^{(i)}\approx\delta_k\widehat{yy}_{k-1}^{(i)}+y_k^2\\ \widehat{xy}_k^{(i)}\approx\delta_k\widehat{xy}_{k-1}^{(i)}+x_ky_k\end{cases} \quad (3.16)$$

Additionally, when applying this set of weights we obtain:

$$\Omega_k=\mathcal{A}_k^{(i)}. \quad (3.17)$$

Let us note that expressions in (3.16) still hold when the past events have been assigned to different lines, since the auxiliary parameters of a given model are only updated when an event is assigned to it.

### 3.2.3 Optimization strategy

As previously stated, the actual value of $\theta_k^{(i)}$ is never directly required. Instead, we just need its sinus and cosinus. This avoids the computation of an arctangent, a sinus and a cosinus, at the cost of computing three square roots. Let us remind the reader that we are updating the parameters of the line with every event assigned to it, and consequently

it is of great importance to limit the number of operations carried out every time. As shown in the Appendix D, from (3.11) and after some trigonometry we obtain:

$$\sin\left(\theta_k^{(i)}\right) = \pm\sqrt{\frac{1-\beta_k}{2}}, \quad \cos\left(\theta_k^{(i)}\right) = +\sqrt{\frac{1+\beta_k}{2}}, \tag{3.18}$$

where:

$$\beta_k = \pm\sqrt{\frac{a_k^2}{a_k^2+b_k^2}}. \tag{3.19}$$

Here, we are just keeping the positive solution for $\cos\theta_k^{(i)}$, because $\theta \in [-\pi/2,\ \pi/2]$. This yields a total of four possible combinations for $\sin\theta_k^{(i)}$, $\cos\theta_k^{(i)}$. To disambiguate the right solution from the four possible ones, a procedure is given in the Appendix F. Let us also remark that these equations are always well defined, since $\frac{a_k^2}{a_k^2+b_k^2} \geq 0$ and $\beta_k \leq 1$ for all values of $a_k$, $b_k$.

This is the method used in our implementation of the algorithm tested in the experiments.

### 3.2.4 Event-Based Line Detection Algorithm

A summary of the complete event-based line detection procedure is given in Algorithm 2. At this point, the detection provides the starting point of the next objective: how to detect segments from the lines.

**Algorithm 2** Event-Based Line Detection

**Require:** $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T, \ \forall k \geq 0$

**Ensure:** $\rho_k^{(i)}, \theta_k^{(i)} \ \ \forall i = 1...N$

```
for every oriented event o_k do
  for every line L^(i) do
    Apply the decay to A_k^(i) using (3.6)
  end for
  if ∃ any lines verifying (3.7) then
    i ← arg max_j A_k^(j) s.t. L^(j) verifies (3.7)
    A_k^(i) ← A_k^(i) + 1
    Update sin θ_k^(i), cos θ_k^(i) and ρ_k^(i) using (3.18) and (3.10)
    if A_k^(i) > A^(up) then
      Output the line
    end if
  else
    Initialize a new line model using (3.8)
  end if
end for
```

## 3.3 Event-Based Segment Detection

In this section we expand the line detection algorithm to a segment detection. Segments, from a line, are detected by localizing discontinuities which are actually the segments'

endpoints. Hence, we add to the line detection algorithm some procedure to output the positions of these endpoints at the same time the line is detected and tracked. In our implementation, the algorithm produces a pair of *endpoint events* for each segment. These events signal the position in space and time of the endpoints. As such, the output of this segment detector is a stream of endpoint events that can be used as feature for matching and learning tasks.

### 3.3.1 Activity of each pixel

Let us define, for each line $\mathcal{L}^{(i)}$, two vectors $\boldsymbol{\mathcal{X}}_k^{(i)} \in \mathbb{R}^w$ and $\boldsymbol{\mathcal{Y}}_k^{(i)} \in \mathbb{R}^h$. These vectors contain the activity of each pixel of the line, projected on the $x$ and $y$ axis respectively, where $w$, $h$ are the width and height of the sensor. The activity of each pixel is increased whenever an event is assigned to the line at a distance smaller than $l$ pixels, and it follows a speed-tuned exponential decay afterwards. Thus, when an oriented event $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ is assigned to a line $\mathcal{L}^{(i)}$, we update $\boldsymbol{\mathcal{X}}_k^{(i)}$ and $\boldsymbol{\mathcal{Y}}_k^{(i)}$ according to:

$$\boldsymbol{\mathcal{X}}_k^{(i)}(x) = \begin{cases} \boldsymbol{\mathcal{X}}_{k-1}^{(i)}(x) e^{-\|\boldsymbol{v}_k\| \Delta t_k} + 1 & \text{if } |x - x_k| < l, \\ \boldsymbol{\mathcal{X}}_{k-1}^{(i)}(x) e^{-\|\boldsymbol{v}_k\| \Delta t_k} & \text{otherwise,} \end{cases} \tag{3.20}$$

$$\boldsymbol{\mathcal{Y}}_k^{(i)}(y) = \begin{cases} \boldsymbol{\mathcal{Y}}_{k-1}^{(i)}(y) e^{-\|\boldsymbol{v}_k\| \Delta t_k} + 1 & \text{if } |y - y_k| < l, \\ \boldsymbol{\mathcal{Y}}_{k-1}^{(i)}(y) e^{-\|\boldsymbol{v}_k\| \Delta t_k} & \text{otherwise,} \end{cases} \tag{3.21}$$

where $x = 1, 2, ..., w$, and $y = 1, 2, ..., h$. Here, $l$ is a tuning parameter controlling the size of the local neighborhood for the computation of the pixel's activity.

### 3.3.2 Generation of Endpoint Events

Pixels of a line are labeled as active if their activity is greater than a predefined threshold $\mathcal{P}^{(up)}$ (where we choose the symbol $\mathcal{P}^{(up)}$ in order to distinguish it from the activity threshold for the lines $\mathcal{A}^{(up)}$). The search for neighboring active pixels is then performed on the $x$ or $y$ projection, depending on the orientation of the line (see Fig. 3.2)

Roughly speaking, if $|\theta_k^{(i)}| > \pi/4$ $\left(\text{or } \cos(\theta_k^{(i)}) < \cos(\pi/4)\right)$ we can say that the line is closer to being horizontal than vertical: in that case, each pixel of the line corresponds to a unique $x$ coordinate, and we consequently perform our search on $\boldsymbol{\mathcal{X}}_k^{(i)}$. Otherwise we employ $\boldsymbol{\mathcal{Y}}_k^{(i)}$.

When an oriented event $\mathbf{o}_k$ is assigned to a line $\mathcal{L}^{(i)}$ we look for active pixels in its neighborhood, starting at the position of the event. Thus, if $|\theta_k^{(i)}| > \pi/4$ we look for recent pixels in $\boldsymbol{\mathcal{X}}_k^{(i)}$ starting at $x_k$. Otherwise we perform our search in $\boldsymbol{\mathcal{Y}}_k^{(i)}$ starting at $y_k$. The endpoints are then given by the furthest active pixel in each direction, as shown in Fig. 3.2, where we impose a minimum length of $l/2$ to the segments. Two *endpoint events* are finally generated, giving the endpoints positions in space and time after a smoothing process using a blob tracker similar to the one introduced in [67].

### 3.3.3 Event-Based Segment Detection Algorithm

The event-based segment detection procedure is summarized in the Algorithm 3.

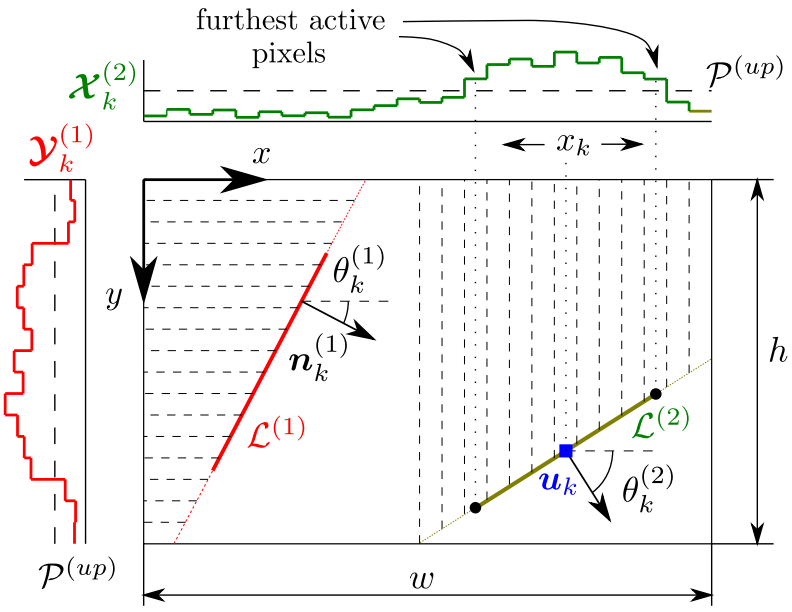



Figure 3.2: The normal to $\mathcal{L}^{(1)}$ forms a small angle with the horizontal: $|\theta_k^{(1)}| < \pi/4$. We can then say that this line is closer to being vertical than horizontal, and each pixel of the line corresponds to a unique $y$ position. The opposite holds true for $\mathcal{L}^{(2)}$: if the event $\mathbf{e}_k$ at position $\boldsymbol{u}_k$ is assigned to $\mathcal{L}^{(2)}$, we look for neighboring active pixels on $\mathcal{X}_k^{(2)}$ starting at $x_k$. The furthest active pixel in each direction gives us an endpoint, and we generate two endpoint events at their positions.

## 3.4 Results

In this section, we present two experiments to assess the accuracy of the algorithm with scenes captured by an Asynchronous Time-Based Image Sensor (ATIS) with $304 \times 240$ pixels resolution [20]. In the first one, we applied the line and segment detection to a monitored scene for which we have built a ground truth. In the second experiment, we used two real scenes while moving the sensor. The algorithm was implemented in C++ and tested in a standard computer running Debian Linux.

### 3.4.1 Controlled scene

In order to numerically evaluate the results of the algorithm we first apply it to a controlled environment, containing an object composed of 13 lines oriented between 0° and 90° at 7.5° intervals, as shown in Fig. 3.3(a). These lines are printed on a piece of paper and fixed on a horizontal rail, which can be moved at different controlled speeds. The scene is then recorded with an ATIS sensor fixed above (see Fig. 3.3(b)), while the rail moves back and forth in the field of view of the camera. The movement is repeated eight times at increasing speeds.

Ground truth values for numerical evaluation of the algorithm are obtained from reconstructed frames, created by plotting together every 1000 events. Since the movement of the object is always contained on a plane, its position w.r.t. the camera can be inferred by matching a planar template [92] estimated from a homographic transform. This is achieved with the Matlab implementation of the ECC algorithm [93] available online.

https://fr.mathworks.com/matlabcentral/fileexchange/27253-ecc-image-alignment-algorithm–

**Algorithm 3** Event-Based Segment Detection

**Require:** $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ assigned to $\mathcal{L}^{(i)}$

**Ensure:** stream of endpoint events

**for** every oriented event $\mathbf{o}_k$ assigned to $\mathcal{L}^{(i)}$ **do**
  Update $\boldsymbol{\mathcal{X}}_k^{(i)}$ and $\boldsymbol{\mathcal{Y}}_k^{(i)}$ using (3.20) and (3.21)
  **if** $\mathcal{A}_k^{(i)} > \mathcal{A}^{(up)}$ **then**
    **if** $\cos\theta_k^{(i)} > \cos\frac{\pi}{4}$ **then**
      $x_1 \leftarrow x_k + 1$
      **while** $x_1 < w$ **and** $\boldsymbol{\mathcal{X}}_k^{(i)}[x_1] > \mathcal{P}^{(up)}$ **do**
        $x_1 \leftarrow x_1 + 1$
      **end while**
      $x_2 \leftarrow x_k - 1$
      **while** $x_2 \geq 0$ **and** $\boldsymbol{\mathcal{X}}_k^{(i)}[x_2] > \mathcal{P}^{(up)}$ **do**
        $x_2 \leftarrow x_2 - 1$
      **end while**
      **if** $x_1 - x_2 > l/2$ **then**
        $y_1 \leftarrow (x_1 \cos\theta_k^{(i)} - \rho_k^{(i)}) / \sin\theta_k^{(i)}$
        Generate an endpoint event $[x_1, y_1, t_k, p_k]^T$
        $y_2 \leftarrow (x_2 \cos\theta_k^{(i)} - \rho_k^{(i)}) / \sin\theta_k^{(i)}$
        Generate an endpoint event $[x_2, y_2, t_k, p_k]^T$
      **end if**
    **else**
      $y_1 \leftarrow y_k + 1$
      **while** $y_1 < h$ **and** $\boldsymbol{\mathcal{Y}}_k^{(i)}[y_1] > \mathcal{P}^{(up)}$ **do**
        $y_1 \leftarrow y_1 + 1$
      **end while**
      $y_2 \leftarrow y_k - 1$
      **while** $y_2 \geq 0$ **and** $\boldsymbol{\mathcal{Y}}_k^{(i)}[y_2] > \mathcal{P}^{(up)}$ **do**
        $y_2 \leftarrow y_2 - 1$
      **end while**
      **if** $y_1 - y_2 > l/2$ **then**
        $x_1 \leftarrow (y_1 \sin\theta_k^{(i)} - \rho_k^{(i)}) / \cos\theta_k^{(i)}$
        Generate an endpoint event $[x_1, y_1, t_k, p_k]^T$
        $x_2 \leftarrow (y_2 \sin\theta_k^{(i)} - \rho_k^{(i)}) / \cos\theta_k^{(i)}$
        Generate an endpoint event $[x_2, y_2, t_k, p_k]^T$
      **end if**
    **end if**
  **end if**
**end for**

From these ground truth values we compute the true flow of the object, displayed in Fig. 3.4 (top). We verify that the observed velocity in the $y$ axis remains approximately zero for the whole recording, except for some small vibrations. The $x$ velocity takes much

image-registration-

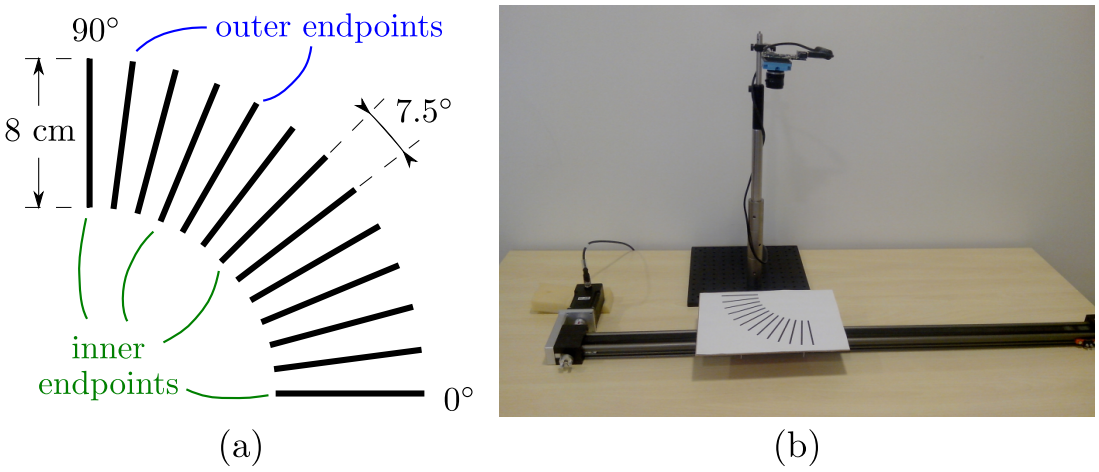



Figure 3.3: (a) Simple model used for the generation of the first scene, containing 13 lines between 0° and 90° at 7.5° intervals. When referring to its endpoints, we will distinguish the *outer endpoints* and the *inner endpoints*. (b) Experimental setup for the recording of the scene: the model is printed on a piece of paper and mounted on a horizontal rail, which can be moved at different controlled speeds in the field of view of the ATIS camera, placed above.

bigger values, going from positive to negative as the object moves back and forth. The maximum absolute value is equal to 654.2 px/s. As a comparison, in the first slowest movement the value of the flow in the $x$ direction is around 60 px/s. Thus, the velocity in our scene varies by more than a factor of 10. Analogously, from these ground truth values we can obtain the apparent acceleration of the object on the image plane, denoted $[\alpha_x, \alpha_y]^T$ and expressed in px/s$^2$, which can be observed in Fig. 3.4 (bottom).

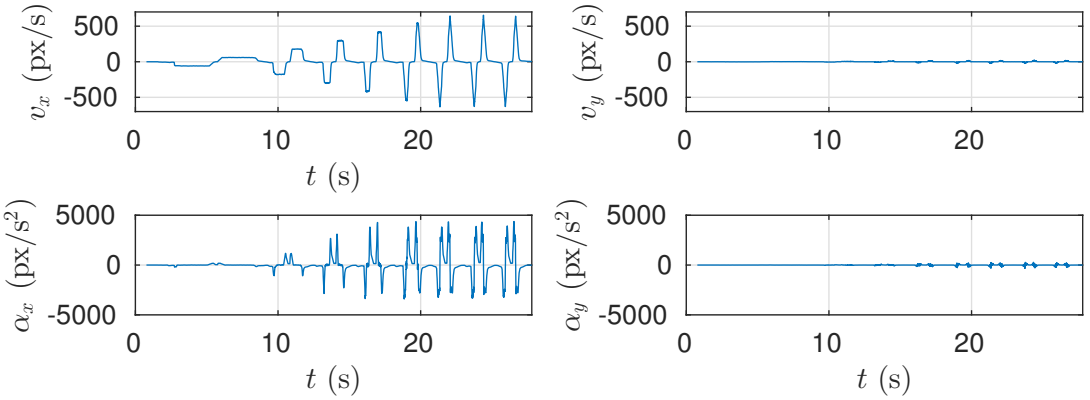



Figure 3.4: Top: observed velocity of the object on the focal plane, in px/s. The velocity on the $y$ axis remains close to zero for the whole recording, while the $x$ velocity varies between negative and positive values, with a maximum absolute value of 654.2 px/s. Bottom: observed acceleration in px/s$^2$

### Line detection

In the first place we evaluate just the line detection part of the algorithm, without performing segment detection. The tuning parameters used in this experiment are shown in the Table 3.1 given below.

Table 3.1: List of parameters for line detection

| $d^{(max)}$ (px) | $\alpha^{(max)}$ (°) | $\mathcal{A}^{(up)}$ | $N$ |
|---|---|---|---|
| 3 | 18 | 75 | 100 |

We show in Fig. 3.5(a) three snapshots depicting the recorded scene at three characteristic instants, where frames are reconstructed by plotting events happening within a 10 ms window. Fig. 3.5(b) shows the output of the normal flow algorithm at these same three instants, where the flow of each oriented event is represented by a straight line whose length is proportional to the norm of the flow. We verify that the flow is always normal to the lines, while its norm is smaller for the lines at smaller angles.

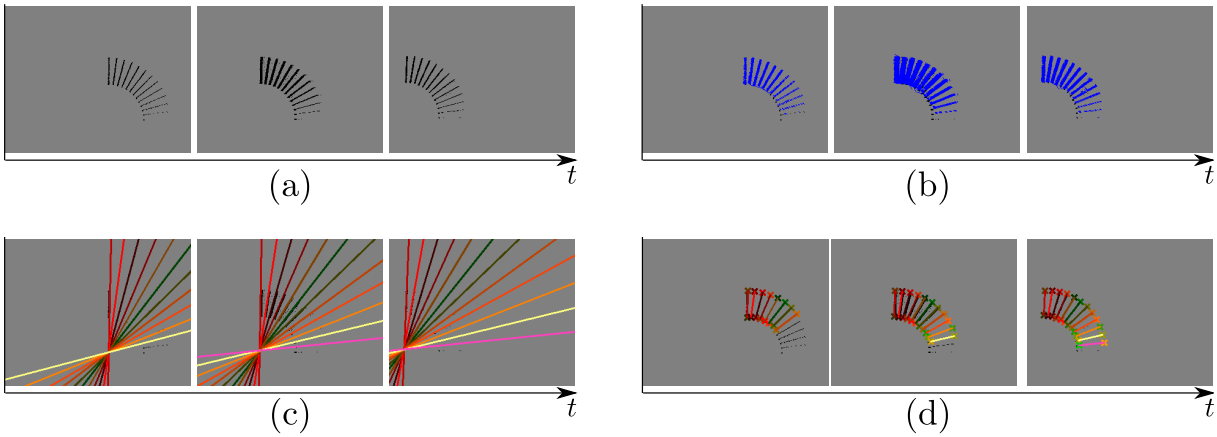



Figure 3.5: (a) Frames reconstructed by plotting events happening within a 10 ms window, showing the recording at three characteristic instants. (b) Output of the normal flow algorithm: the flow is always normal to the local contours, while its norm is smaller for the lines at smaller angles. (c) Output of the line detection step. An index is assigned to each line as they are created, and we plot each line index with a different color. The horizontal line can seldom be tracked, as it generates very few events, preventing the computation of their flow. We are able of keeping track of the rest of the lines for the whole recording. (d) Output of the segment detection step. As the movement of the lines changes its direction we sometimes loose track of part of the endpoints, but we recover them again.

Fig. 3.5(c) shows the output of the line detection step at the same three instants, where active line models are superimposed on the events using a different color for each line index. In the leftmost snapshot we show the state of the system as the object starts its movement: we verify that all lines are detected, except the ones at 0° and 7.5°. This happens because the activity of these lines has not reached the threshold $\mathcal{A}^{(up)}$ yet, for they are almost perpendicular to the direction of the movement and they consequently generate very few events. However, the norm of their flow is also smaller, which implies that their activity follows a slower decay: consequently, as the movement continues, the line at 7.5° reaches a sufficient level of activity and it is correctly tracked. The horizontal line, however, can seldom be detected. Let us remind the reader that the first step of our algorithm is the computation of the visual flow of the incoming events, which cannot be accomplished if the number of recent events in the neighborhood of the current event is

not enough.

We display in Fig. 3.6 the evolution of the speed-tuned activity $\mathcal{A}_k$ for six different lines: the ones oriented at 90°, 75°, 60°, 45°, 30° and 15° (the rest of lines are not shown for simplicity, but equivalent results are obtained). We verify that the activity of the different lines is very similar, even though the ones with greater angles generate many more events. In the same way, the activity is stable for the different velocities, which allows us to keep track of the lines for most of the recording.

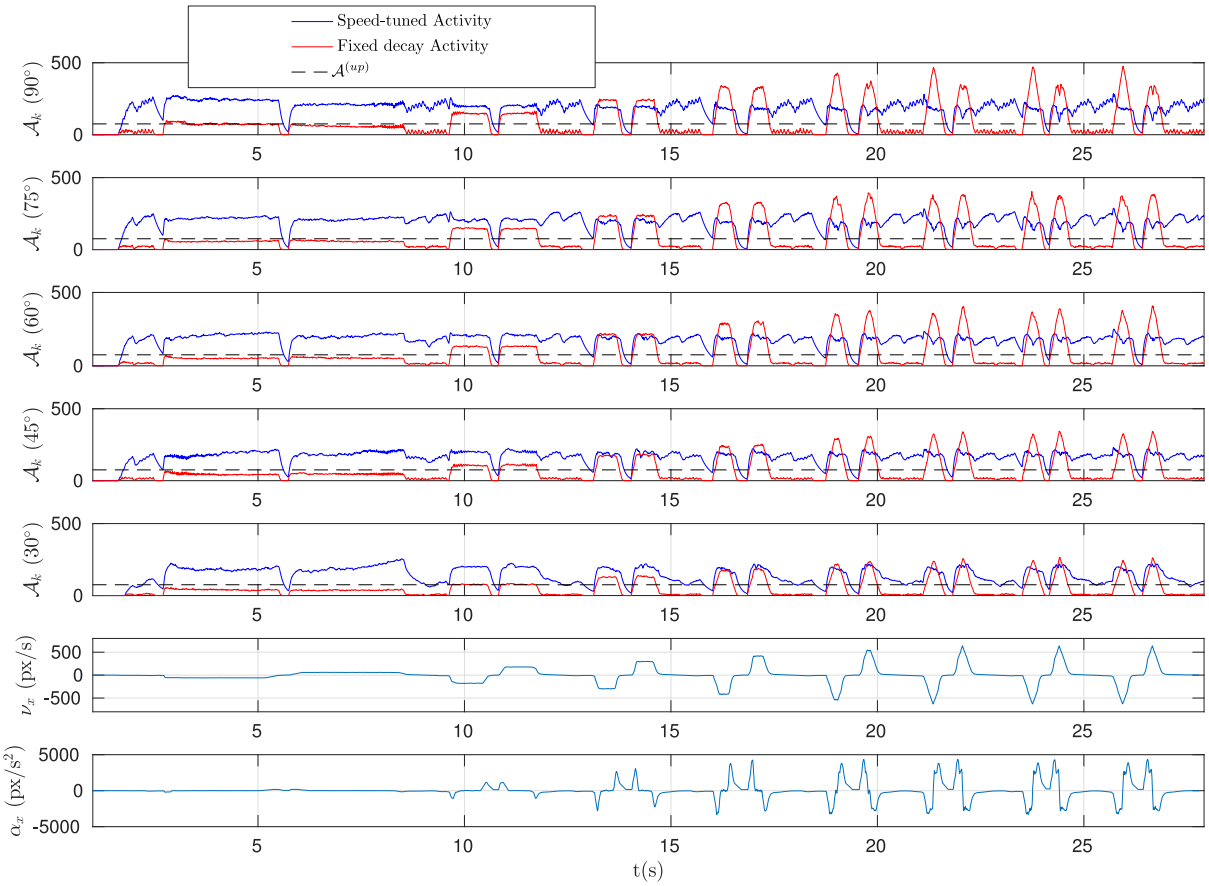



Figure 3.6: Evolution of the activity for six different lines. We display the results obtained with both the speed-tuned decreasing strategy and the fixed decreasing strategy. We verify that the speed-tuned strategy is much more stable, in spite of the different orientations and apparent velocities. The dashed line indicates the threshold $\mathcal{A}^{(up)}$.

Additionally, we compare the results of the speed-tuned decay activity to a constant decay one. The constant decay function is given by $e^{-\Delta t_k/\tau}$, with $\tau$ a tuning parameter. Here, we choose $\tau = 20000$ µs, which yields values for the fixed decay activity in the same order of magnitude as the speed-tuned one. The obtained values are displayed in Fig. 3.6: we verify that the fixed decay activity is very sensitive to the orientation and the apparent velocity of the lines, imposing the need of adjusting the tuning parameter $\tau$ according to the velocity and the direction of movement of the observed objects. We show in Fig. 3.7 the percentage of time for which the different lines are active when applying both the speed-tuned and the fixed decay strategies with the current set of parameters. As we can see, the speed-tuned strategy is more reliable as it yields higher percentages.

However, we can observe in Fig. 3.6 that the activity decreases during deceleration periods: this corresponds to an expected behavior, due to the adaption of the speed-tuned

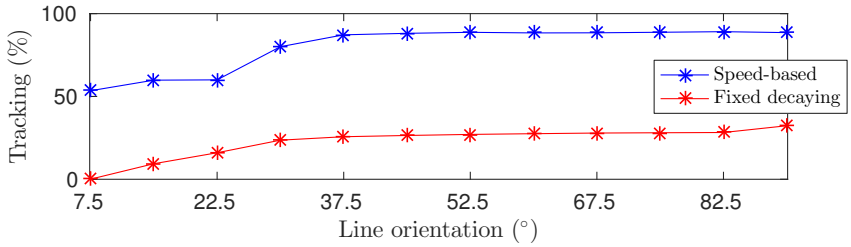



Figure 3.7: Percentage of the recording for which we are able of keeping track of the lines, for both the speed-tuned and the fixed decay strategies. We verify that the speed-tuned method is more stable

decay strategy. This is particularly significant when the apparent motion is globally slower (in particular when looking at the activity curve for the line at 30°, Fig. 3.6), because the number of events generated by the silicon retina is then smaller. The algorithm is then more sensitive to error in velocity estimation and internal noise of the sensor's (mismatches, thermal noise, etc.). These observations are confirmed regarding the errors in model estimation.

Let us next numerically evaluate the error in the estimation of the line parameters. Here, we evaluate $\rho$ and $\theta$ independently and we define two errors: $\xi_k^{(\rho)}$ (in pixels) and $\xi_k^{(\theta)}$ (in degrees), given by the absolute value of the difference between the estimated and the ground truth values of the corresponding parameters. We show in Fig. 3.8 the obtained values for the lines oriented at 90°, 75°, 60°, 45°, 30° and 15°, where the error is computed only when the corresponding line is active. From this figure we can extract the following conclusions:

- In general, the algorithm is capable of correctly estimating the parameters of the observed lines. The values of the errors remain small for all lines for the whole duration of the recording.
- Lines parallel to the motion are harder to detect and track since the motion is almost not triggering events. This hampers the computation of the optical flow, yielding less stable results.
- We loose lines at smaller angles more often. This usually happens when the lines stop their movement.

In order to establish a clearer comparison between the accuracy produced for the different orientations let us plot in Fig. 3.9 the values of the mean errors obtained for the different lines for the whole duration of the recording. We verify that these errors are bigger for smaller angles. As an example, for the line at 90° (which moves perpendicularly to its contour) we obtain mean errors of just 0.31° and 0.90 px.

### Segment detection

The parameters for the line detection are the same used in the previous experiment (indicated in Table 3.1), while the extra parameters required for segment detection are: $l = 10$ px, $\mathcal{P}^{(up)} = 1.5$.

We show in Fig. 3.5(d) the output produced by the segment detection step, where the tracked endpoints are indicated by crosses in different colors for each endpoint. In the leftmost snapshot we observe the state of the detection as the object initiates its

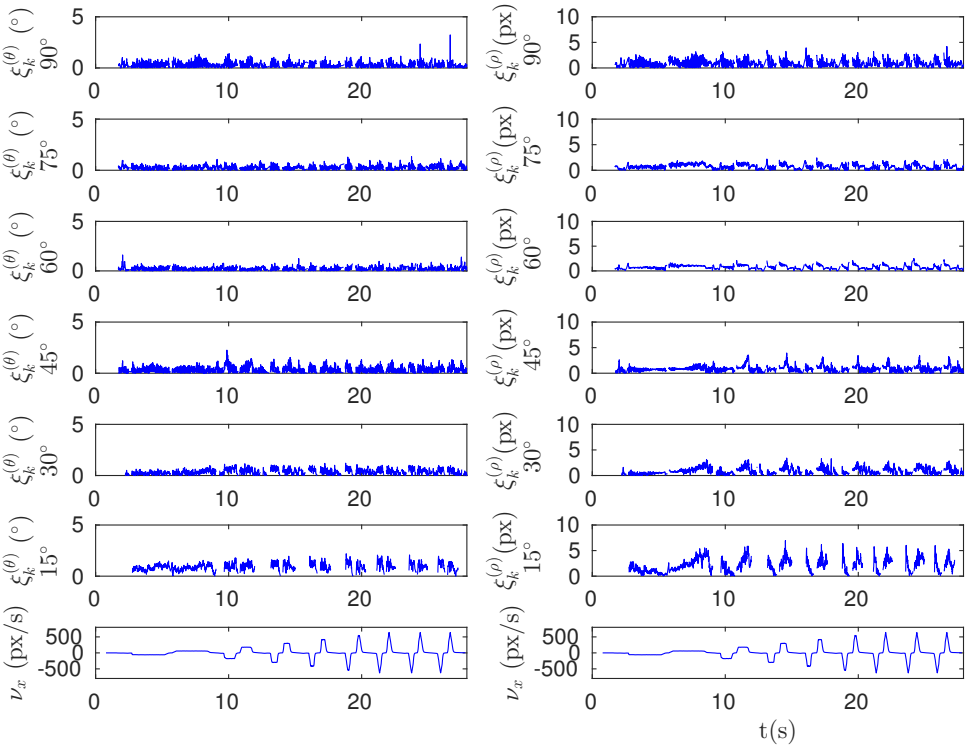



Figure 3.8: Errors in the estimation of $\rho$ and $\theta$ for six of the lines contained in the scene. Results for almost horizontal lines are less stable, but we are still able of correctly tracking these lines.

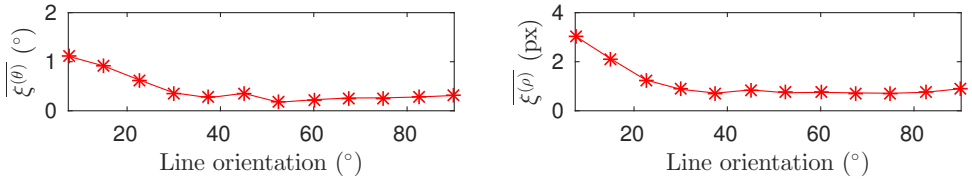



Figure 3.9: Evolution of the mean error in the estimation of $\theta$ and $\rho$ with the orientation of the line. We verify that the best results are obtained for lines oriented perpendicularly to the direction of the movement.

movement: we verify that some of the endpoints are not detected in this first instant, since their activity has not reached its threshold yet. Like in the previous experiment, as the object continues moving, we are able of detecting the missing endpoints. All endpoints, except the ones belonging to the horizontal line, are correctly detected and tracked for the whole duration of the recording.

We next compare the obtained results with the ground truth values. We plot in Fig. 3.10 the tracking results obtained for three endpoints, namely the outer endpoints of the lines at 90°, 45° and 15°. In the figure, ground truth values are indicated with a dashed line and compared with the obtained results. As we can see, they are very similar, allowing us to conclude that our method can correctly detect the endpoints in this simple scene.

We provide also a quantitative evaluation of the tracking by showing the tracking error $\xi_k^{(t)}$ (in pixels) as the distance between the tracked endpoints and the corresponding ground truth values. We only take into account the instants when the endpoint trackers are active. Consequently, we will also evaluate the percentage of the recording for which the endpoints are detected. We show in Fig. 3.11 the mean tracking errors produced

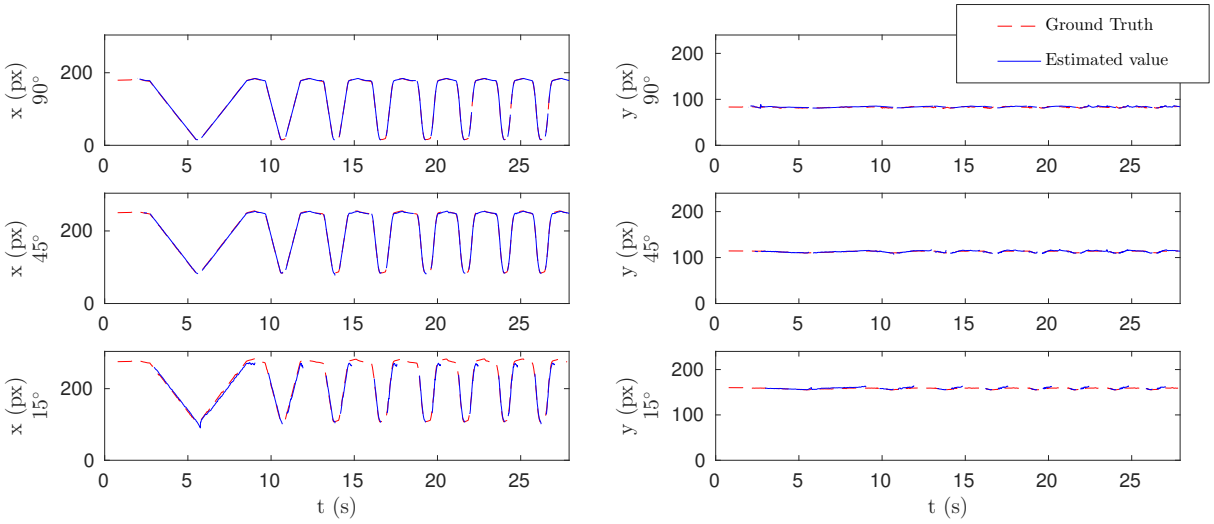



Figure 3.10: Tracking results for the outer endpoints of the lines at 90°, 45° and 15°.

over the whole recording for the different endpoints. As we can see, errors are greater for smaller angles. As an example, the mean tracking errors committed for the endpoints belonging to the vertical line are 1.49 px for the outer endpoint, and 1.54 px for the inner endpoint.

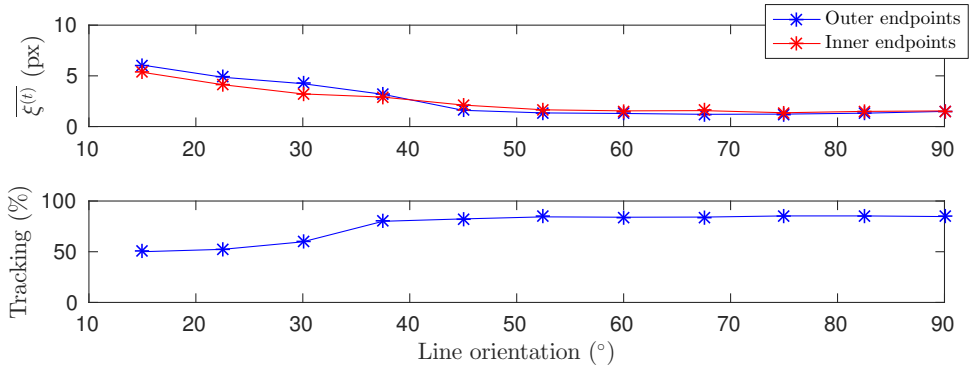



Figure 3.11: Accuracy of the endpoint tracking for the different endpoints. We obtain more accurate results and greater percentage of active time for lines with greater angles.

In the same way, the smaller the angle the smaller the percentage of the time that we can track the endpoints. We are able of tracking the endpoints in the vertical line for 84.6% of the time.

### 3.4.2 Urban scenes

We next show the output of the algorithm applied to two urban scenes. Both recordings were performed with a hand-held ATIS camera moving in circles. We process them using the same parameters as in the previous case, except for the maximum number of line models, which is increased until $N = 1000$. In this case, we do not provide any quantitative measurement of the accuracy produced, as ground truth values are not easily attainable. Instead, these results are just shown as a qualitative demonstration of what can be achieved with our algorithm. This is also a key opportunity to discuss some properties of the proposed method, regarding its potential applications.

### Outdoors

We first apply the method to a recording of an urban landscape, which contains a large number of lines. We show in Fig. 3.12(a) three snapshots reconstructed from the recording, while Fig. 3.12(b) depicts the output of the line detection step. We observe that horizontal or vertical lines tend to predominate depending on the direction of the movement at the corresponding instant.

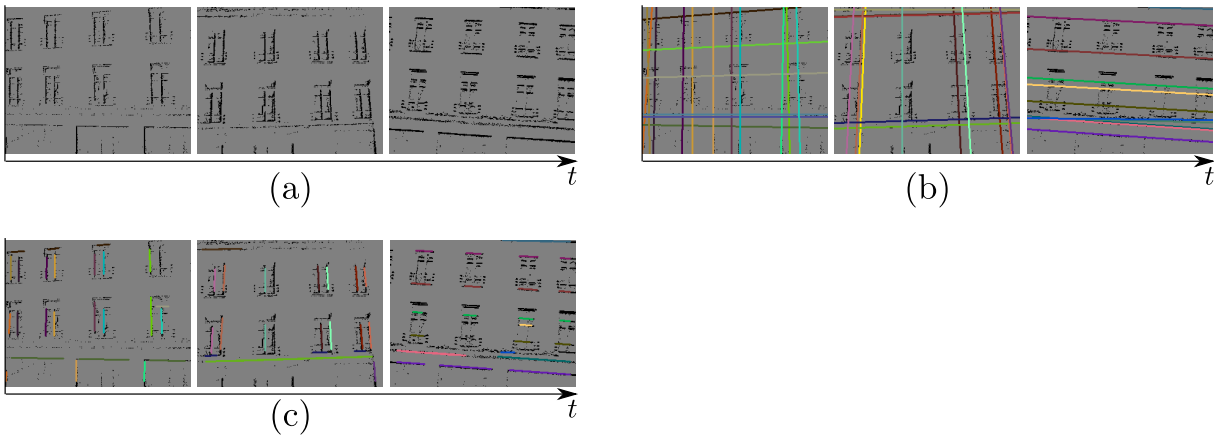



Figure 3.12: (a) Frames reconstructed from the recording. (b) Output of the line detection step. (c) Output of the segment detection step.

In Fig. 3.12(c) the output of the segment detection is also shown. For visibility reasons, we do not display any crosses marking the endpoints' positions. We verify that our algorithm is capable of recovering an important part of the segments present in the recorded scene.

### Indoors

The second real scene tested with the detection/tracking algorithm is the recording of an office. Fig. 3.13(a) shows three snapshots obtained from the recording: we observe that the scene contains a great number of lines, whose lengths are smaller than in the urban scene recording. As we can see in Fig. 3.13(b), many of these shorts segments are not detected because their activities are not normalized with respect to their lengths, which causes the longer segments to be more likely to be detected.

We show in Fig. 3.13(c) the output of the segment detection algorithm when processing this scene with the same set of parameters as for the urban environment. We verify that many of the small segments are not detected, for the reasons previously explained. It is then possible to adjust the parameters, reducing the activity threshold $\mathcal{A}^{up}$ and the length of the window $l$. We show in Fig. 3.13(d) the output produced after tuning the parameters: as we can see, we are able of detecting a greater number of small segments. Of course, the algorithm's parameters can be automatically adjusted, requiring a given number of lines and a maximum number of segments allowed per line.

## 3.4.3 Computational time

We next discuss the computational time required by the current C++ implementation of the algorithm. These tests were performed in the same conditions described in Section

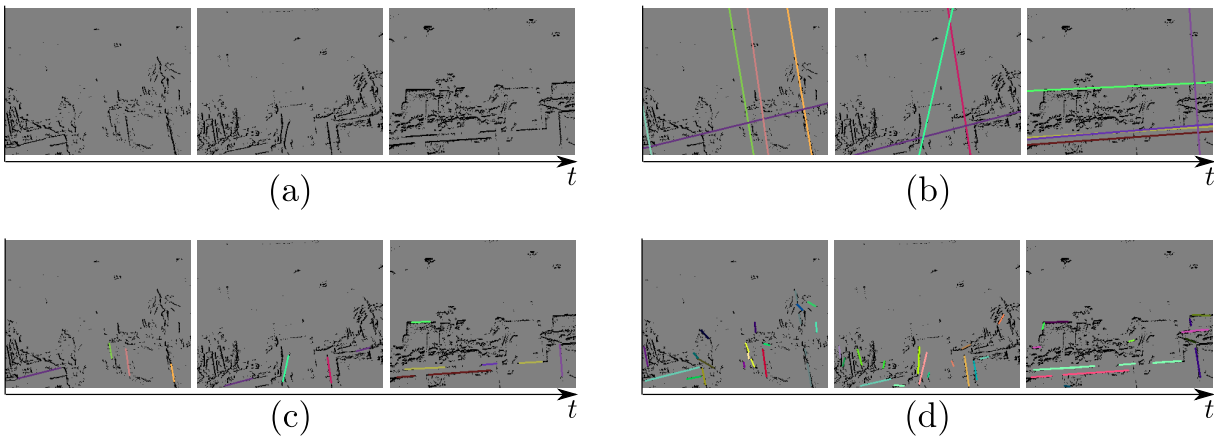



Figure 3.13: (a) Frames reconstructed from the recording. (b) Output of the line detection step of the algorithm. (c) Output of the segment detection part of the algorithm. (d) Output of the segment detection part of the algorithm after tweaking the parameters.

2.4.3, where the computational time of the shape tracking algorithm was studied. Let us remind the reader that we are performing our tests in a standard computer running Debian Linux and equipped with a Intel Core i7-4790 processor. As in the previous chapter, the code was not paralellized and just one core was used.

As in Section 2.4.3, we measure the processing time every millisecond, and study the ratio of processing time to the length of the considered portion of the recording. Let us show in Fig. 3.14 the ratio obtained when processing the first recording (controlled scene), using the set of parameters given in Table 3.1. Let us note that we are considering the processing time required by the whole processing chain, which includes reading the recording file, processing the events for noise removal and computing the visual flow of the incoming events, in addition to the line (or segment) detection. We display at the top of Fig. 3.14 the computational time required to perform line detection, while the results for segment detection are shown at the bottom.

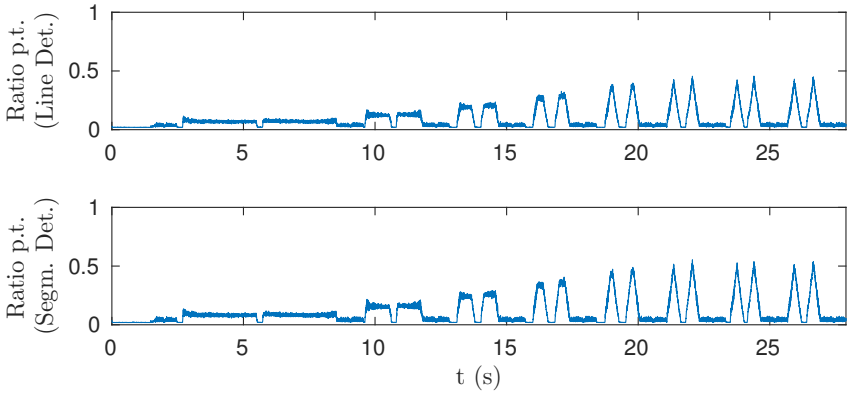



Figure 3.14: Ratio of processing time to the length of the recording for the first recording (controlled scene) for both line and segment detection. Since this ratio is always smaller than 1, we can conclude that we are processing this recording in real time. The computational time is related to the velocity of the object, because faster moving objects generate a greater number of events.

From Fig. 3.14 we can extract the following conclusions:

- The ratio of processing time to the length of the recording is always smaller than 1. This implies that we are processing the event stream faster than we acquire it (both for segment and line detection).
- The segment detection algorithm is around 10 percent more computationally demanding, on average.
- This ratio is clearly correlated to the speed of the object (see Fig. 3.4). This can be easily explained because faster moving objects generate a greater number of events that need to be processed.

As we did in Section 2.4.3, let us next display the ratio of processing time as a function of the event rate. Here, we consider the computational time required to process the outdoors recording, which contains a greater number of events. We show in Fig. 3.15 the ratio of processing time for three different values of the parameter $N$ (i.e. the maximum number of lines that can be initialized). As in the previous chapter, we verify that the computational time is increasing with the number of events, and can be approximated by a linear function. This allows us to extrapolate and compute the maximum rate of events that can be processed in real time by our algorithm, depending on the value of $N$.

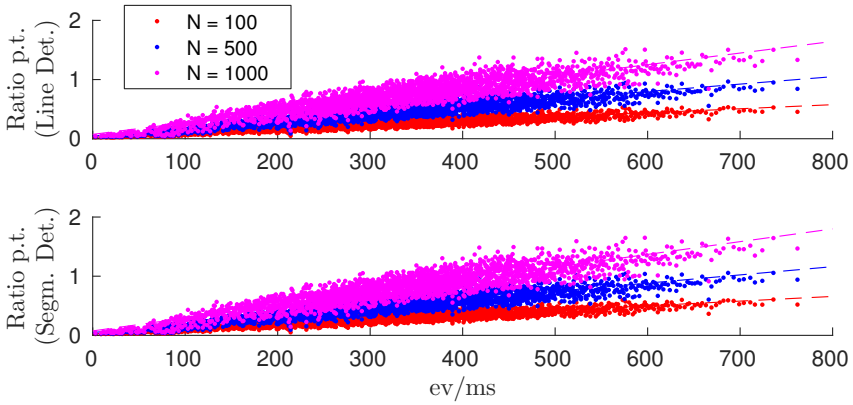



Figure 3.15: Computational time per ms (for line detection -top Figure- and segment detection -bottom Figure) as a function of the number of events per ms. The required computational time increases with the event rate: regressions of these data are represented as dashed lines in order to highlight their linear dependency, related to the parameter $N$ (i.e. the number of considered line models).

We then show in Fig. 3.16 the evolution of the maximum event rate that can be processed in real time by the algorithm, both for line detection (top) and segment detection (bottom). Indeed, we verify that, even for big values of $N$, we are able to process big event rates in real time. As an example, for $N = 500$ we can perform real-time line detection for event rates up to 760 ev/ms, and real-time segment detection for event rates up to 684 ev/ms.

## 3.5 Discussion

This chapter introduces a new event-based line and segment detection algorithm. Based on an iterative process, each event is used to update the current model and forgotten afterwards. This latter is obtained by applying a speed-tuned exponential decay to the

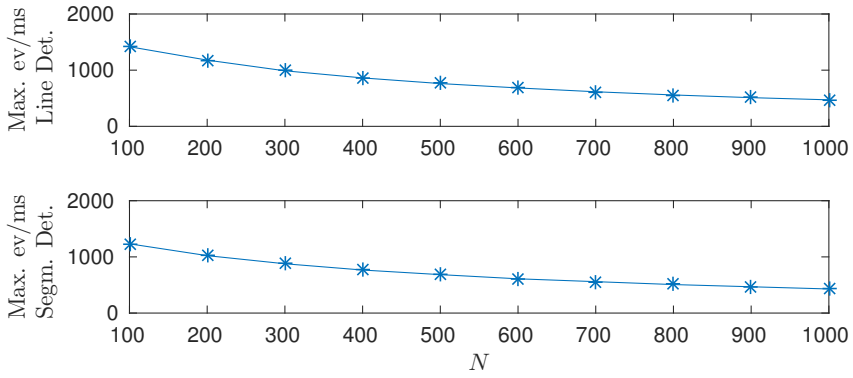



Figure 3.16: Maximum event rate that can be processed in real time, as a function of the parameter $N$ (Top: line detection. Bottom: segment detection).

contribution of each event to the line model, and then to the segment model. Our approach takes into account the dynamics of the visual information through a local computation of the optical flow, ensuring that the models are essentially estimated on the current visual information.

The method considers a speed-tuned exponential decay for the update of the line models. We prove this strategy to be an important advantage with respect to fixed decay approaches, as it provides an automatic adaptation mechanism to the different dynamics present in the scene.

These properties are validated through an experimental protocol, involving a controlled scene for which ground truth values are available. This allows us to assess the accuracy and the robustness of the algorithm in a controlled environment. Indeed, the mean accuracy obtained for the line model parameters, i.e. the distance to the origin and the angle, are in most cases lower than 1 pixel and 1° respectively. The results for the segment endpoints are also accurate: for example, the mean tracking errors committed for a line perpendicular to the motion are around 1.5 pixels. In addition, a complete analysis of the computational time required by the presented algorithm is proposed, showing that our current implementation is able to process event streams in real time.

These characteristics make our event-based line and segment detection algorithms suitable for real-time applications, such as navigation of robotic platforms in real environments (particularly for line-based SLAM algorithms [73]-[74]).

We have introduced, in the previous and the present chapters, two event-based plane tracking algorithms. According to these results, we can now tackle the problem of 3D pose estimation, which often requires some previous tracking technique (see, e.g., Chapter 5). The motivation for this next step comes mainly from visual servoing: position-based visual servoing techniques require the knowledge of the camera's 3D pose relative to some tracked object. We thus present in the next two chapters two event-based 3D pose estimation algorithms.

# Chapter 4

# Event-Based 3D Pose Estimation

## 4.1 Introduction

This chapter addresses the problem of 3D pose estimation of an object from the visual output of a calibrated event-based camera, assuming that an approximate 3D model of the object is known [40]. 3D pose estimation is the problem of finding the 3D translation and rotation of a given object relative to the camera observing it. It is a fundamental issue with various applications in machine vision and robotics such as Structure From Motion (SFM) [94]-[95], object tracking [96], augmented reality [97] or visual servoing [98]-[99]. Numerous authors have tackled finding a pose from 2D-3D correspondences: methods range from simple approaches like DLT [100] to complex ones like PosIt [101]. We can distinguish between between two classes of techniques: iterative [101]-[102] or non iterative [100], [103]. However, most techniques are based on a linear or nonlinear system of equations that needs to be solved, differing mainly by the estimation techniques used to solve the pose equations and the number of parameters to be estimated.

Existing algorithms differ in speed and accuracy. Some provide a fixed computation time independent of the number of points of the object, e.g. [103]. The DLT [100] is the simplest, slowest and weakest approach for estimating the 12 parameters in the projection matrix. However, it can be used to provide an initial estimate of the pose. PosIt [101] is a fast method that does not use a perspective projection, but instead relies on an orthographic projection to estimate fewer parameters. The method was later extended [104] to take into account planar point clouds. More recently, CamPoseCalib has been introduced [105]. It is based on the Gauss-Newton method and nonlinear least squares optimization [106]. Other methods are based on edge correspondences [96], [107], or photometric information [108]. When the pose problem is solved from point correspondences it is known as the P$n$P (Perspective-$n$-Point) problem [41], [109]. This task will be further explored in the next chapter.

In spite of this field being extensively studied, most current 3D pose estimation algorithms are designed to work on images acquired at a fixed rate. As previously stated, frame based stroboscopic acquisition induces massively redundant data and temporal gaps that make it difficult to estimate the 3D pose of an object without computationally expensive iterative optimization techniques [106].

This chapter introduces a new approach, solving the 3D pose estimation problem employing the output of an asynchronous event-based camera. This algorithm is one the first 3D pose estimation methods designed to work on the output of such a sensor. In [110], an event-based iterative closest point (ICP) like tracking algorithm is introduced,

where the pattern is a 2D point cloud which is updated with every incoming event, so that it matches the projection of a given object. However, the method presented in [110] assumes that the pattern is undergoing some affine transformation [111], and therefore it does not account for a more general transformation due to perspective projection that an object freely evolving in the 3D space can experiment. Furthermore, the pose of the object in the 3D space is never estimated. The work presented in this chapter is an extension to the 3D space of the method introduced in [110]. As we will show, the asynchronous and sparse output of the sensor allows us to tackle the 3D pose estimation problem in a simple and intuitive way, resulting in a real-time algorithm providing accurate results.

The outline of the current chapter is as follows: we first present our method in Section 4.2. The technique is then experimentally validated in Section 4.3, where four different experiments are presented. Finally, we briefly discuss the obtained results in Section 4.4.

## 4.2 Event-based 3D pose estimation

Let us next explain the method for 3D pose estimation from the output of a neuromorphic silicon retina.

### 4.2.1 Problem formulation

Let us consider a moving rigid object observed by a calibrated silicon retina. The movement of the object generates a stream of events on the focal plane of the camera. Attached to this object is a frame of reference, known as the *object-centered reference frame*, whose origin we denote as $\mathbf{V}^{(0)}$. The pinhole projection [112] maps 3D points $\mathbf{V}$ expressed in the object-centered reference frame into $\mathbf{v}$ on the camera's focal plane (see Fig. 4.1), according to the relation:

$$\begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix} \sim \Lambda \begin{bmatrix} R & \boldsymbol{T} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ 1 \end{bmatrix}, \tag{4.1}$$

where $\Lambda$ is the $3 \times 3$ matrix defining the camera's intrinsic parameters—obtained through a prior calibration procedure— while $\boldsymbol{T} \in \mathbb{R}^3$ and $R \in SO(3)$ are the extrinsic parameters. The sign $\sim$ indicates that the equality is defined up to a scale [112]. $\boldsymbol{T}, R$ are also referred to as the relative pose between the object and the camera [113], where $\boldsymbol{T}$ represents the translation vector and $R$ the rotation matrix. As the object moves, only the pose changes and needs to be estimated.

An estimation of the pose can be found by minimizing the orthogonal projection errors on the line of sight for each 3D point, as illustrated by Fig. 4.1. Thus, we minimize a cost function directly on the 3D structure rather than computing it on the focal plane [114]. The advantage of this approach is that a correct match of the 3D points leads to a correct 2D projection on the focal plane, but the reverse is not necessarily true.

The high temporal resolution of the camera allows us to acquire a smooth trajectory of the moving object. We can then consider each event generated by the moving object as relatively close to the previous position of the object. Since the event-based camera detects temporal contours, all moving objects can be represented by a set of vertices and edges. We can then set the following convention: let $\{\mathbf{V}^{(i)}\}$ be the set of 3D points defining an object, identified by their index $i$. These 3D points are vertices and their projections onto the retina focal plane are noted as $\mathbf{v}^{(i)}$. The edge defined by vertices $\mathbf{V}^{(i)}$, $\mathbf{V}^{(j)}$ is noted as $\varepsilon^{(ij)}$. Fig. 4.1 shows a general illustration of the problem.

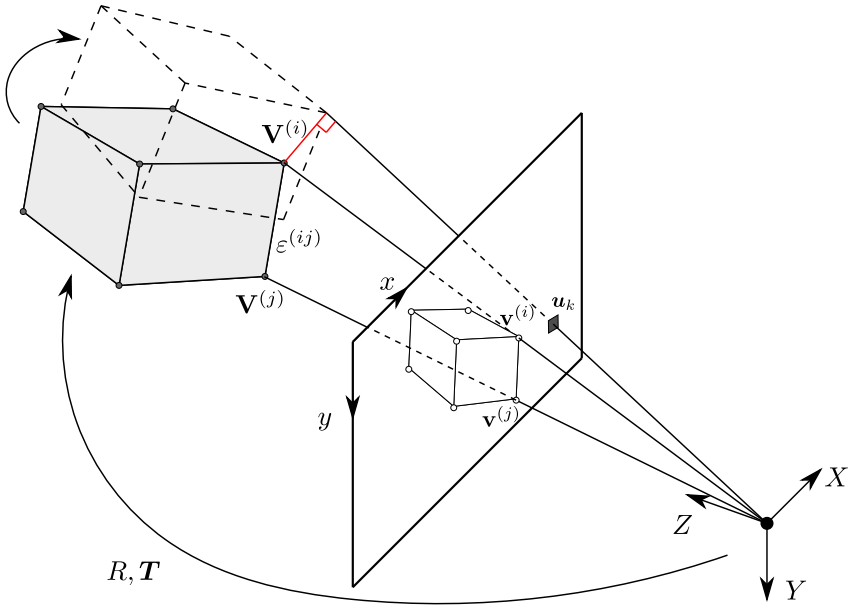



Figure 4.1: The 3D pose of an object is given by a rotation matrix $R \in SO(3)$ and a translation vector $\boldsymbol{T} \in \mathbb{R}^3$. If an event $\mathbf{e}_k$ has been generated by a point $\mathbf{V}^{(i)}$ of the model, then $\mathbf{V}^{(i)}$ lies on the line of sight passing through the camera center and the position of the event on the focal plane $\boldsymbol{u}_k$.

Using the usual computer graphics conventions [115]-[116], an object is described as a polygon mesh. This means that all the *faces* of the model are simple polygons, triangles being the standard choice. The boundaries of a *face* are defined by its *edges*.

**Remark:** in the classical computer vision literature, the intrinsic parameters matrix is often denoted $K$. However, in this document we are reserving the letter $k$ for indicating the index of an event, and the capital letter $K$ for the total number of events contained in a recording. For this reason we choose here to use $\Lambda$. We will keep this notation for the remainder of this PhD thesis.

### 4.2.2 Rotation formalisms

Rotation matrices are one of the most widely used formalisms for representing rotations. Some other representations are possible, each one with its own advances and disadvantages. Another convenient parametrization for the rotation is to use unit quaternions [113]: a quaternion is a 4-tuple, providing a more efficient and less memory intensive method of representing rotations compared to rotation matrices. It can be easily used to compose any arbitrary sequence of rotations. For example, a rotation of angle $\theta$ about rotation axis $\widetilde{\boldsymbol{r}}$ is represented by a quaternion $q$ satisfying:

$$q(\theta, \widetilde{\boldsymbol{r}}) = \cos\left(\frac{\theta}{2}\right) + \widetilde{\boldsymbol{r}} \sin\left(\frac{\theta}{2}\right), \tag{4.2}$$

where $\widetilde{\boldsymbol{r}}$ is a unit vector. In this chapter we will use the quaternion parametrization for rotations.

When trying to visualize rotations, we will also use the axis-angle representation, defined as the rotation vector $\boldsymbol{r} = \theta\widetilde{\boldsymbol{r}}$.

### 4.2.3 2D edge selection

For each incoming event, the first step of the algorithm is to determine whether the event has been generated by the tracked object and, if so, which of its points has more likely generated it. To that end, we first perform a 2D matching step explained below.

The model of the object and its initial pose are assumed to be known. This allows us to virtually project the model onto the focal plane as a set of edges. For each incoming event $\mathbf{e}_k$ occurring at position $\boldsymbol{u}_k = [x_k, y_k]^T$ on the focal plane, we look for the closest visible edge. Thus, for every visible edge $\varepsilon^{(ij)}$, projected on the focal plane as the segment $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$, we compute $d_k^{(ij)}$, the euclidean distance from $\boldsymbol{u}_k$ to $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$ (see Fig. 4.2). To compute this distance, $\boldsymbol{u}_k$ is projected onto the line defined by $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$. If this projection falls inside of the segment $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$, then the distance is given by the generic expression:

$$d_k^{(ij)} = \frac{\|(\boldsymbol{u}_k - \mathbf{v}^{(i)}) \times (\mathbf{v}^{(j)} - \mathbf{v}^{(i)})\|}{\|\mathbf{v}^{(j)} - \mathbf{v}^{(i)}\|}, \tag{4.3}$$

where $\times$ is the cross product. If the projection is not inside $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$, then $d_k^{(ij)}$ is set to be equal to the distance between $\boldsymbol{u}_k$ and the closest endpoint.

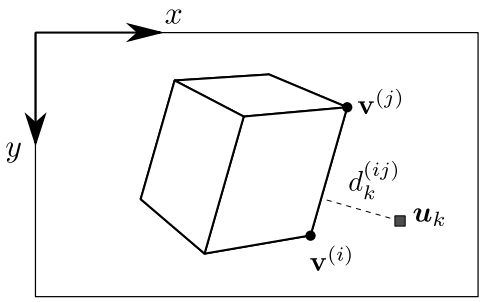



Figure 4.2: Edge selection for an event $\mathbf{e}_k$ occurring at $\boldsymbol{u}_k$. The distance between $\boldsymbol{u}_k$ and each visible edge $\varepsilon^{(ij)}$ is computed as $d_k^{(ij)}$, the euclidean distance between $\boldsymbol{u}_k$ and the segment defined by the projected edge $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$.

We set a maximum allowed distance for the event to be assigned to an edge as $d^{(max)}$. The edge to which the event is assigned to is $\varepsilon_k^{(nm)}$ such that:

$$d_k^{(nm)} = \min_{i,j} d_k^{(ij)}, \tag{4.4}$$

assuming $d_k^{(nm)} \leq d^{(max)}$, otherwise the event is considered as noise and discarded.

**Remark 1**: in complex scenarios, the 2D matching step can be further strengthened by applying more refined criteria. We implement a 2D matching based on Gabor events, which are oriented events generated by events lying on a line [117]. When the 2D matching is performed using this technique, a Gabor event will only be assigned to a visible edge if the angle of the event and the angle formed by the edge are close enough. An example of application of this method will be shown in the experiments, where pose estimation is performed even with partial occlusions and egomotion of the camera.

**Remark 2**: this section assumes that the visibility of the edges is known. This is done via a *hidden line removal* algorithm [118] applied for each new pose of the model.

### 4.2.4 3D matching

Once $\varepsilon_k^{(nm)}$ has been determined, we look for the point on the edge that has generated the event. The high temporal resolution of the sensor allows us to set this point as the closest to the line of sight of the incoming event. Performing this matching between an incoming event in the focal plane and a physical point on the object allows to overcome issues that appear when computation is performed directly in the focal plane. The perspective projection on the focal plane is neither preserving distances nor angles, i.e. the closest point on the edge in the focal plane is not necessarily the closest 3D point of the object.

The camera calibration parameters allow us to map each event at pixel $\boldsymbol{u}_k$ to a line of sight passing through the camera's center. The 3D matching problem is then equivalent to a search for the smallest distance between any two points lying on the object's edge and the line of sight.

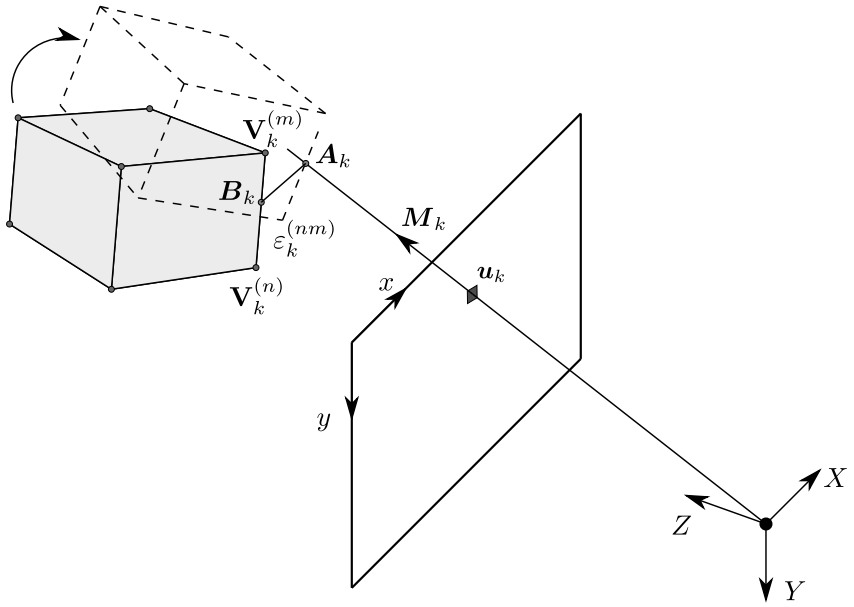



Figure 4.3: Geometry of the 3D matching problem: an event $\mathbf{e}_k$ at position $\boldsymbol{u}_k = [x_k, y_k]^T$ is generated by a change of luminosity in the line of sight passing through the event, defined by the vector $\boldsymbol{M}_k$. $\boldsymbol{A}_k$ is a point of the line of sight and $\boldsymbol{B}_k$ a point of the edge $\varepsilon_k^{(nm)}$, such that the minimum distance between these two lines is reached. Finding $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ is the objective of the 3D matching step.

As shown in Fig. 4.3, let $\boldsymbol{A}_k$ be a point on the line of sight of an incoming event $\mathbf{e}_k$ located at $\boldsymbol{u}_k$ in the focal plane. Let $\boldsymbol{B}_k$ be a point on the edge $\varepsilon_k^{(nm)}$ that has been computed as being at a minimal distance from the line of sight passing through $\boldsymbol{u}_k$. We can assume $\mathbf{e}_k$ to be generated by a 3D point on the moving object at the location $\boldsymbol{A}_k$, that was at $\boldsymbol{B}_k$ before $\mathbf{e}_k$ occurred. This hypothesis is reasonable as due to the high temporal resolution events are generated by small motions. Finding $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ is the scope of the 3D matching step.

Let $\boldsymbol{M}_k$ be the vector defining the line of sight of $\mathbf{e}_k$, it can be obtained as:

$$\boldsymbol{M}_k = \Lambda^{-1} \begin{bmatrix} \boldsymbol{u}_k \\ 1 \end{bmatrix}, \tag{4.5}$$

where $\Lambda$ denotes the camera's intrinsic parameters matrix.

$\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ can therefore be expressed as:

$$\boldsymbol{A}_k = \alpha \boldsymbol{M}_k \tag{4.6}$$

$$\boldsymbol{B}_k = \mathbf{V}_k^{(n)} + \beta(\mathbf{V}_k^{(m)} - \mathbf{V}_k^{(n)}), \tag{4.7}$$

where $\alpha$ and $\beta$ are two real valued parameters. $\mathbf{V}_k^{(m)}$ and $\mathbf{V}_k^{(n)}$ denote the 3D vertices defining edge $\varepsilon_k^{(nm)}$.

Let $\boldsymbol{\varepsilon}_k^{(nm)}$ denote $\mathbf{V}_k^{(m)} - \mathbf{V}_k^{(n)}$, we are looking for solutions such that $(\boldsymbol{A}_k - \boldsymbol{B}_k)$ is perpendicular to both $\boldsymbol{\varepsilon}_k^{(nm)}$ and $\boldsymbol{M}_k$. Hence, we obtain the following equation:

$$\begin{bmatrix} -\boldsymbol{M}_k^T \boldsymbol{M}_k & \boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)} \\ -\boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)} & \left(\boldsymbol{\varepsilon}_k^{(nm)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -\left(\boldsymbol{V}_k^{(n)}\right)^T \boldsymbol{M}_k \\ -\left(\boldsymbol{V}_k^{(n)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)} \end{bmatrix}. \tag{4.8}$$

Solving this equation for $\alpha$ and $\beta$ provides both $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$. The solution to this system is discussed in the Appendix G.

We also set a maximum 3D distance between $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$, denoted $D^{(max)}$. If the distance between $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ is larger than this value we discard the event.

### 4.2.5 Rigid motion estimation

Knowing $\boldsymbol{B}_k$ and $\boldsymbol{A}_k$ allows us to estimate the rigid motion that transforms $\boldsymbol{B}_k$ into $\boldsymbol{A}_k$. The rigid motion is composed of a translation $\boldsymbol{\Delta T}_k$ and a rotation $\Delta q_k$ around $\boldsymbol{V}^{(0)}$, the origin of the object-centered reference frame.

Let us define the scaling factor $\lambda$ such that $\boldsymbol{\Delta T}_k$ is related to the vector $\boldsymbol{A}_k - \boldsymbol{B}_k$ as:

$$\boldsymbol{\Delta T}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & m \end{bmatrix} \lambda(\boldsymbol{A}_k - \boldsymbol{B}_k), \tag{4.9}$$

where $(\boldsymbol{A}_k - \boldsymbol{B}_k)$ is the translation that makes $\boldsymbol{B}_k$ coincide with $\boldsymbol{A}_k$. Here, $m$ is a multiplier that allows us to set the scaling factor independently for the $Z$ axis. The need for this extra degree of freedom can be justified because changes in the depth of the object will only become apparent through changes in the $x$ or $y$ position of the events on the focal plane. Consequently, the system does not react in the same way to changes in depth as it does to changes in the $X$ or $Y$ position, resulting in a different latency for the $Z$ axis. $m$ is then a tuning factor that will be set experimentally.

The rotation around $\boldsymbol{V}^{(0)}$ is given by a unit quaternion $\Delta q_k$ of the form:

$$\Delta q_k(\phi\theta_k, \widetilde{\boldsymbol{r}}_k) = \cos\left(\frac{\phi\theta_k}{2}\right) + \widetilde{\boldsymbol{r}}_k \sin\left(\frac{\phi\theta_k}{2}\right), \tag{4.10}$$

where $\widetilde{\boldsymbol{r}}_k$ is a unit vector collinear to the axis of rotation and $\phi\theta_k$ is equal to the rotation angle, that we conveniently define as a product between a scaling factor $\phi$ and the angle $\theta_k$ defined below.

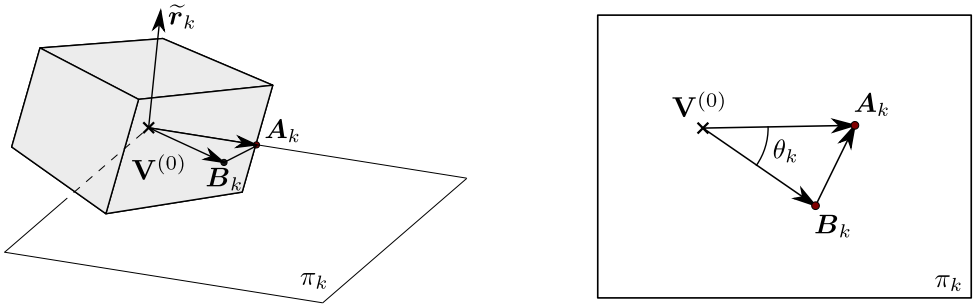
If $\pi_k$ is the plane passing through $\boldsymbol{B}_k$, $\boldsymbol{A}_k$ and $\boldsymbol{V}^{(0)}$ (see Fig. 4.4(a)) such that $\widetilde{\boldsymbol{r}}_k$ is its normal, then $\widetilde{\boldsymbol{r}}_k$ can be computed as:

$$\widetilde{\boldsymbol{r}}_k = \frac{(\boldsymbol{B}_k - \boldsymbol{V}^{(0)}) \times (\boldsymbol{A}_k - \boldsymbol{V}^{(0)})}{\|(\boldsymbol{B}_k - \boldsymbol{V}^{(0)}) \times (\boldsymbol{A}_k - \boldsymbol{V}^{(0)})\|}. \tag{4.11}$$

We define $\theta_k$ as the angle between $(\boldsymbol{B}_k - \boldsymbol{V}^{(0)})$ and $(\boldsymbol{A}_k - \boldsymbol{V}^{(0)})$ (see Fig. 4.4(b)).

$$\theta_k = \tan^{-1}\left(\frac{\left\|\left(\boldsymbol{B}_k - \boldsymbol{V}^{(0)}\right) \times \left(\boldsymbol{A}_k - \boldsymbol{V}^{(0)}\right)\right\|}{\left(\boldsymbol{B}_k - \boldsymbol{V}^{(0)}\right)^T \left(\boldsymbol{A}_k - \boldsymbol{V}^{(0)}\right)}\right). \tag{4.12}$$

If $\boldsymbol{A}_k$, $\boldsymbol{B}_k$ and $\boldsymbol{V}^{(0)}$ are aligned, $\widetilde{\boldsymbol{r}}_k$ is undefined. This happens when no rotation is applied or when the rotation angle is equal to $\pi$. This last case is unlikely to occur because of the small motion assumption.




$(a)$ $(b)$

Figure 4.4: (a) $\pi_k$ represents the plane defined by $\boldsymbol{A}_k$, $\boldsymbol{B}_k$ and the origin of the object centered reference frame $\boldsymbol{V_0}$. The desired rotation is contained in this plane, and thus the rotation axis $\widetilde{\boldsymbol{r}}_k$ is normal to it. (b) Normal view to $\pi_k$. Both $(\boldsymbol{B}_k - \boldsymbol{V_0})$ and $(\boldsymbol{A}_k - \boldsymbol{V_0})$ are contained in this plane, and thus their cross product gives us the axis of rotation. The angle $\theta_k$ between these two vectors is equal to the rotation angle that makes $\boldsymbol{B}_k$ and $\boldsymbol{A}_k$ coincide.

Finally, the pose of the model is updated incrementally according to:

$$\boldsymbol{T}_k = \boldsymbol{T}_{k-1} + \boldsymbol{\Delta T}_k \tag{4.13}$$

$$q_k = \Delta q_k q_{k-1}. \tag{4.14}$$

For the rest of this chapter, this described procedure will be referred to as the *direct transformation* strategy.

**Remark 1:** Once the pose is updated, the next step is to update the transformation between the object and the camera. This is a computationally expensive process that requires transforming the 3D points, projecting them onto the image plane and applying the hidden-line removal algorithm. Consequently, in order to increase the performance of the system, we do not apply the transformation for every incoming event, but every $N$ events. $N$ is experimentally chosen, and its effect on the algorithm discussed in the experiments section.

**Remark 2:** $\lambda$ and $\phi$ are set experimentally, and they should always be equal or smaller than one. When they are smaller than one, we do not fully transform the model so that $\boldsymbol{B}_k$ matches $\boldsymbol{A}_k$ for every event. Instead, we apply a small displacement for each incoming event. Here, it is important to keep in mind that a moving edge generates more than one event. This number and the frequency of events are proportional to the local contrast.

**Remark 2:** since we are assuming that the are always close to the true position, the angle $\theta_k$ will always be small. It is then possible to make the usual assumption for small

angles: $\sin(\theta_k) \approx \theta_k$, and compute $\theta_k$ as:

$$\theta_k = \frac{\|(\boldsymbol{B}_k - \boldsymbol{V}^{(0)}) \times (\boldsymbol{A}_k - \boldsymbol{V}^{(0)})\|}{\|\boldsymbol{B}_k - \boldsymbol{V}^{(0)}\| \, \|\boldsymbol{A}_k - \boldsymbol{V}^{(0)}\|} \tag{4.15}$$

### 4.2.6 Global algorithm

The global algorithm for 3D pose estimation is given below (Algorithm 4).

**Algorithm 4** Event-Based 3D pose estimation algorithm

**Require:** $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T \; \forall k \geq 0$
**Ensure:** $\boldsymbol{T}$, $q$
  Initialize the parameters
  **for** every incoming event $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T$ **do**
    **for** every visible edge $\varepsilon^{(ij)}$ **do**
      Compute the distance $d_k^{(ij)}$ between $\boldsymbol{u}_k$ and $\{\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\}$
    **end for**
    $d_k^{(nm)} \leftarrow \min(d_k^{(ij)})$
    **if** $d_k^{(nm)} \leq d^{(max)}$ **then**
      Solve (4.8) in $\alpha$ and $\beta$
      Compute $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ using (5.9) and (5.10)
      **if** $||\boldsymbol{A}_k - \boldsymbol{B}_k|| \leq D^{(max)}$ **then**
        Compute $\boldsymbol{\Delta T}_k$ and $\Delta q_k$ using (5.23) and (4.10)
        Update $\boldsymbol{T}$ and $q$ using (4.13) and (4.14)
      **end if**
    **end if**
    **for** each $N$ consecutive events **do**
      Apply the transformation to the model
    **end for**
  **end for**

## 4.3 Experiments

In this section we present experiments to test 3D pose estimation on real data . The first two experiments estimate the pose of a moving icosahedron and house model while viewed by a static event-based sensor. In Section 4.3.3 we estimate the pose of the icosahedron from the view of a moving event-based sensor in a scene containing multiple moving objects. In Section 4.3.4 we estimate the pose of the icosahedron under high rotational velocity (mounted on a motor). Finally, in Section 4.3.5 and Section 4.3.6 we investigate how temporal resolution affects pose estimation accuracy, and how implementation parameters affect the time required for computation.

In what follows, we will denote the ground truth as $\{\boldsymbol{T}, q\}$ and the estimated pose as $\{\widehat{\boldsymbol{T}}, \widehat{q}\}$.

All recordings and the corresponding ground truth data are publicly available at https://drive.google.com/folderview?id=0B5gzfP0R1VEFNS1PZ0xKU3F5dG8&usp=sharing

The algorithm is implemented in C++ and tested in recordings of an icosahedron—shown in Fig. 4.5(a)—and the model of a house—Fig. 4.5(b)—freely evolving in the 3D space. We set the following metrics on $\mathbb{R}^3$ and $SO(3)$:

- The absolute error in linear translation is given by the norm of the difference between $\widehat{\boldsymbol{T}}$ and $\boldsymbol{T}$. For a given recording, let $\overline{\boldsymbol{T}} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{T}_k$ be the mean displacement of the object, where $K$ is the total number of events in the recording. We define $\xi_k^{(T)}$ the relative translation error at time $t_k$ as:

$$\xi_k^{(T)} = \frac{\|\widehat{\boldsymbol{T}}_k - \boldsymbol{T}_k\|}{\|\overline{\boldsymbol{T}}\|}. \tag{4.16}$$

- For the rotation, the error is defined with the distance $d$ between two unit quaternions $q$ and $\widehat{q}$:

$$d(q, \widehat{q}) = min\{\|q - \widehat{q}\|, \|q + \widehat{q}\|\}, \tag{4.17}$$

  which is proven to be a suitable metric for $SO(3)$, the space spanned by 3D rotations [119]. It takes values in the range $[0, \sqrt{2}]$. Thus, let $\xi_k^{(q)}$ be the relative rotation error at time $t_k$:

$$\xi_k^{(q)} = \frac{d(q_k, \widehat{q}_k)}{\sqrt{2}}. \tag{4.18}$$

The algorithm provides an instantaneous value of the errors for each incoming event. In order to characterize its accuracy, we will consider $\overline{\xi^{(T)}}$ and $\overline{\xi^{(q)}}$, the mean error for the whole duration of a given recording:

$$\overline{\xi^{(T)}} = \frac{1}{K}\sum_{k=1}^{K}\xi_k^{(T)}, \tag{4.19}$$

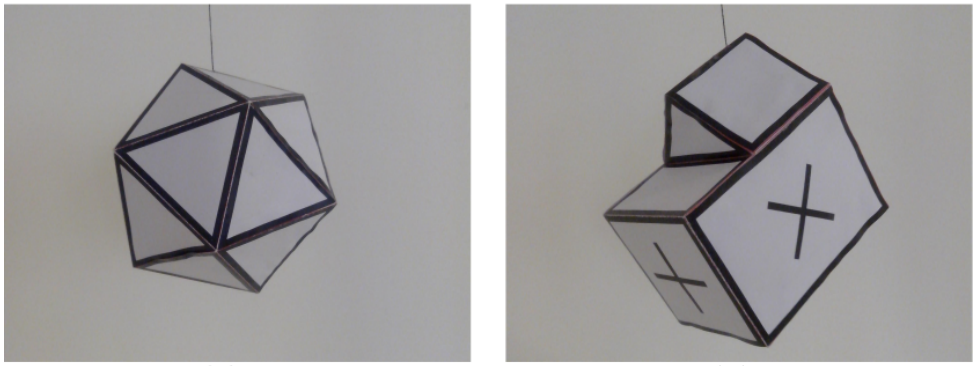
$$\overline{\xi^{(q)}} = \frac{1}{K}\sum_{k=1}^{K}\xi_k^{(q)}. \tag{4.20}$$

### 4.3.1 Icosahedron

The icosahedron shown in Fig. 4.5(a) is recorded by an ATIS sensor for 25 s while freely rotating and moving. The 3D model is a mesh of 12 vertices and 20 triangular faces.

The ground truth is built from frames output from the event-based camera. We have manually selected the image position of the visible vertices every 100 ms and applied the OpenCV implementation of the EP$n$P (Efficient Perspective-$n$-Point) algorithm [109] to estimate the pose. In [109], the authors test the robustness of their algorithm to gaussian noise perturbations on the focal plane. It is important to outline that this is a theoretical disturbance model. They are not assessing their algorithm's performance with real noisy data. Based on their noise model results, we can give an order of magnitude of the ground truth accuracy. Assuming that the manual annotation of the vertices of the icosahedron has at least 2 pixels precision, we can read the pose error from the error curves (Fig. 5 in [109]), that is at most 2%.

The intermediate positions are obtained by linear interpolation, and the intermediate rotations using Slerp (Spherical Linear intERPolation [120]). From the ground truth we compute the model's linear velocity $\boldsymbol{v}$ and the angular velocity $\boldsymbol{w}$. In this recording, the



(a) (b)

Figure 4.5: Real objects used in the experiments. (a) White icosahedron with black edges, used in the first experiment. (b) Non-convex model of a house with cross markers on its faces, used in the second experiment.

linear speed $\|\boldsymbol{v}\|$ reaches a maximum of 644.5 mm/s, while the angular speed $\|\boldsymbol{w}\|$ starts with a maximum of 2.18 revolutions per second (rev/s) at the beginning of the recording and then continuously decreases (see Fig. 4.6).

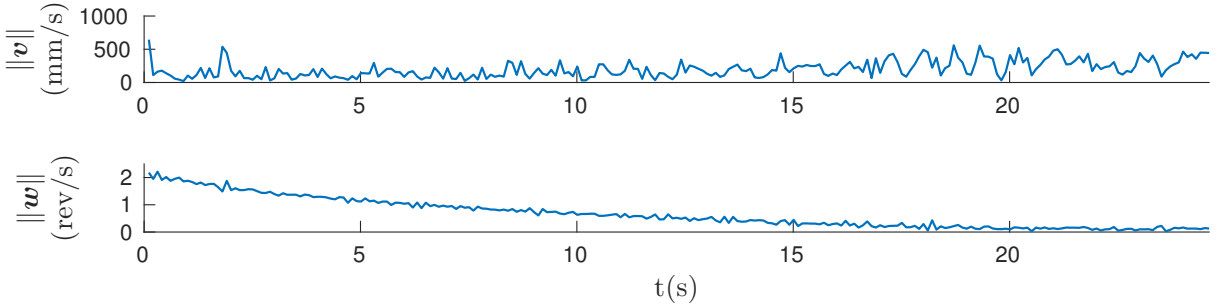



Figure 4.6: Evolution of the linear and angular speeds of the icosahedron for the whole length of the recording.

After several trials, the thresholds are set experimentally to values giving stable and repeatable pose estimations. These are: $d^{(max)} = 20$ pixels and $D^{(max)} = 10$ mm. The remaining tuning parameters are experimentally chosen for each experiment as the ones giving the smallest sum of the mean relative estimation errors $\overline{\xi^{(T)}}$ and $\overline{\xi^{(q)}}$. The update factors $\lambda$, $\phi$ are always taken between 0.001 and 0.4, a large range in which the algorithm has proven to yield stable results.

Fig. 4.7 shows the results when applying our algorithm with $\lambda = 0.4$, $\phi = 0.2$, $N = 1$ and $m = 2$. We show the translation vector $\boldsymbol{T}$ as well as the rotation vector $\boldsymbol{r}$. Plain curves, representing estimation results, are superimposed with dashed lines indicating the ground truth. Snapshots, showing the state of the system at interesting instants are shown. They provide the projection of the shape on the focal plane using the estimated pose.

We verify that plain and dashed lines (representing estimated and ground truth poses respectively) coincide most of the time, showing that the pose estimation is in general correctly performed. Experiments provide the following mean estimation errors: $\overline{\xi^{(T)}} = 1.48\%$ for the translation, and $\overline{\xi^{(q)}} = 1.96\%$ for the rotation. Instantaneous errors reach a local maximum, as a consequence of the large values chosen for $\lambda$ and $\phi$. These parameters

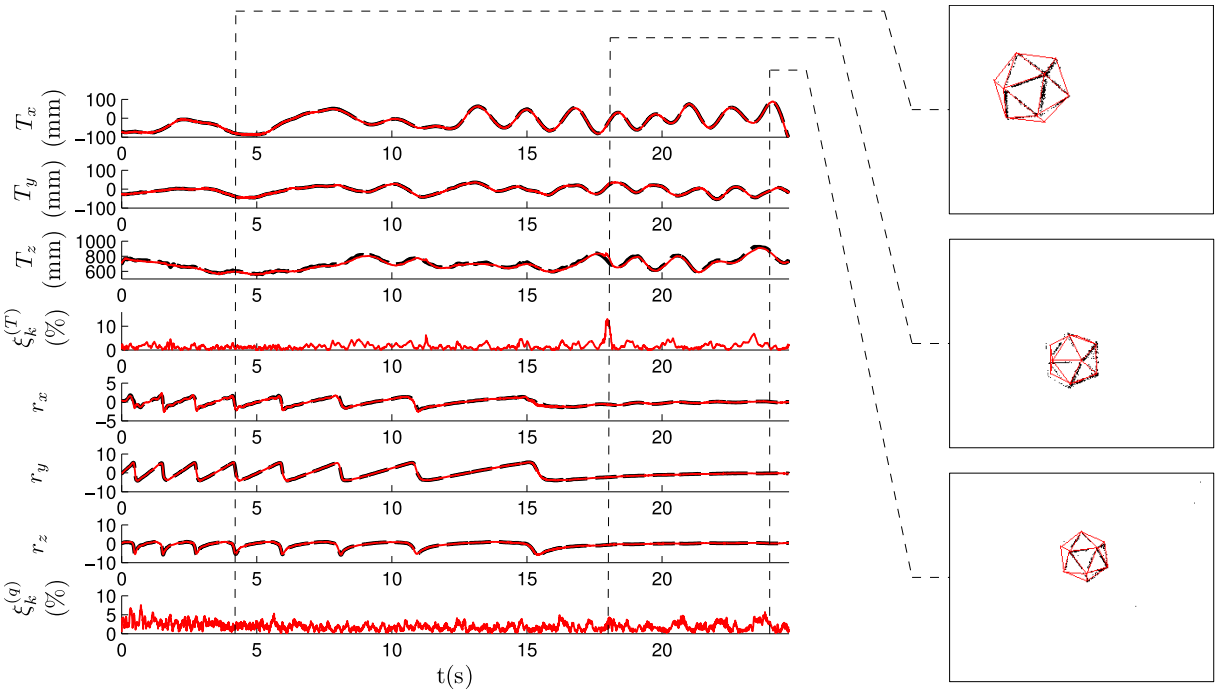



Figure 4.7: Results for the first experiment, where we recorded an icosahedron freely evolving in the 3D space. By abuse of notation, we denote $T_x$, $T_y$ and $T_z$ the components of the translation vector $\boldsymbol{T}$, and $r_x$, $r_y$, $r_z$ the components of the axis-angle representation of the rotation $\boldsymbol{r}$. The dashed lines represent ground truth, while the solid curves represent estimated pose. The snapshots on the top show the state of the system in some characteristic moments, with the estimation made by the algorithm printed over the events.

being gains, large values imply an oscillatory behavior around the correct pose parameters. We include in Fig. 4.7 a snapshot showing the state of the system at this instant, where we observe that the estimation is slightly displaced from the true pose. However, even when considering this local maximum, the estimation errors remain below 15%. The system is always capable of recovering the correct pose.

### 4.3.2 House

This experiment tests the accuracy of the algorithm using a more complex model of a house shown in Fig. 4.5(b). The object is recorded for 20 s while freely rotating and moving in front of the camera. The 3D model is composed of 12 vertices and 20 triangular faces. We compute velocities from the ground truth obtained from generated frames as was done with the icosahedron. In this case, the linear speed reaches a maximum of 537.4 mm/s, while the angular speed starts with a maximum of 1.24 rev/s at the beginning of the experiment and then continuously decreases (see Fig. 4.8).

As in the previous case, we experimentally choose the set of parameters that produces the minimum sum of errors. Fig 4.9 shows the results when applying our algorithm with $\lambda = 0.2$, $\phi = 0.05$, $m = 1$ and $N = 10$. We verify that there is a coherence between the ground truth and the estimated pose showing that the pose estimation is in general correctly estimated. However, in this case we observe a larger local maxima reaching values as high as 20%. These local maxima degrade the overall performance, they provide

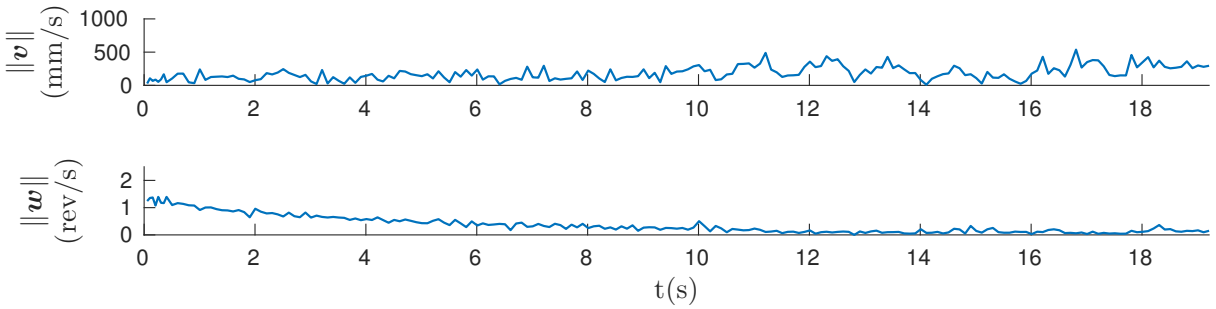



Figure 4.8: Evolution of the linear and angular speeds of the house for the whole length of the recording.

the following values for the mean estimation errors: $\overline{\xi^{(T)}} = 3.12\%$ for the translation and $\overline{\xi^{(q)}} = 2.62\%$ for the rotation, higher than in the previous case. Nevertheless, the system is always capable of recovering the correct pose after these maxima, and the mean estimation errors remain acceptable.

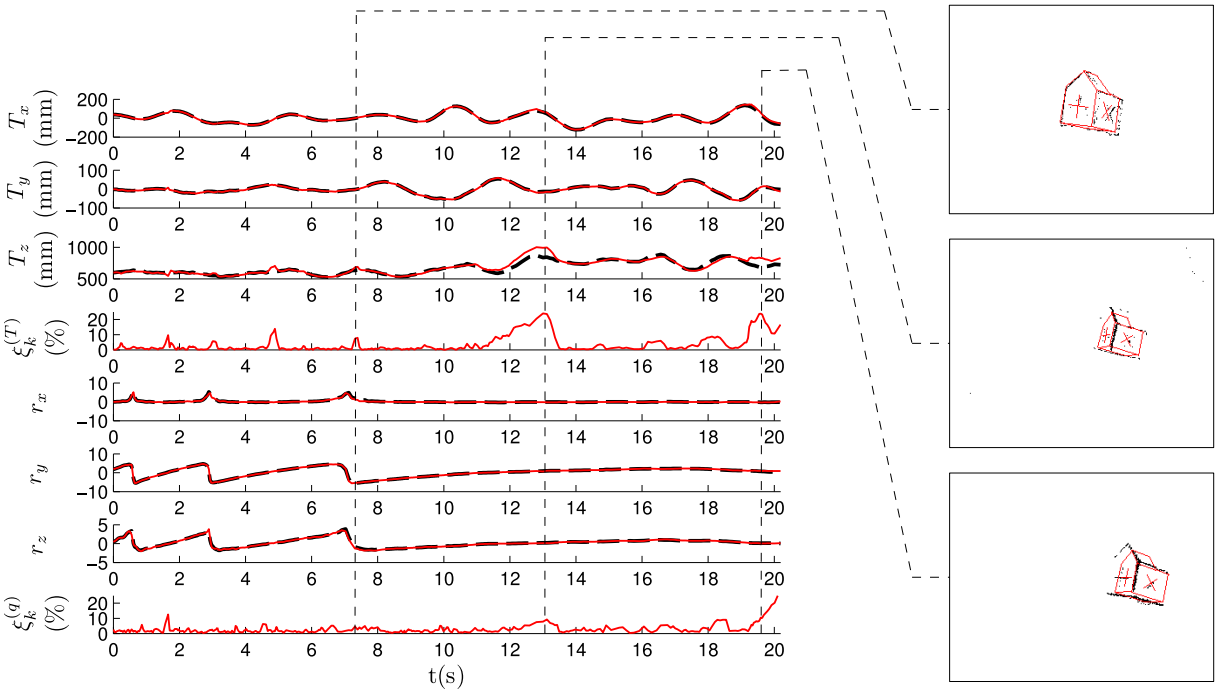



Figure 4.9: Results for the second experiment, where we recorded a non-convex model of a house freely evolving in the 3D space.

In this recording, local maxima mostly occur because of the algorithm mistakenly interpreting the cross markers as edges or viceversa. This usually happens when a given face is almost lateral with respect to the camera. In that case, it provides the projection of these lines very close to each other.

### 4.3.3 2D matching using Gabor events

In this experiment we test pose estimation in a more complex scenario, with egomotion of the camera and partial occlusions of the object, using Gabor events for the 2D matching step. A hand-held icosahedron is recorded for 20s while the camera moves. Ground truth is obtained from reconstructed frames as in the previous experiments.

The parameters for the Gabor events' generation process are set as in [117], and the maximum angular distance for an event to be assigned to an edge is set as 0.174 rad (obtained as $\frac{\pi}{1.5\times 12}$, where 12 is the number of different orientations that the Gabor events can take). The tuning parameters are experimentally chosen as in previous experiments.

Fig. 4.10 shows the evolution of the errors when applying our algorithm with $\lambda = 0.4$, $\phi = 0.2$, $N = 5$ and $m = 4$ (we do not show $\boldsymbol{T}$ or $\boldsymbol{r}$ in order to lighten the figure). We verify that the estimation errors remain low for the whole recording, always below 10%.

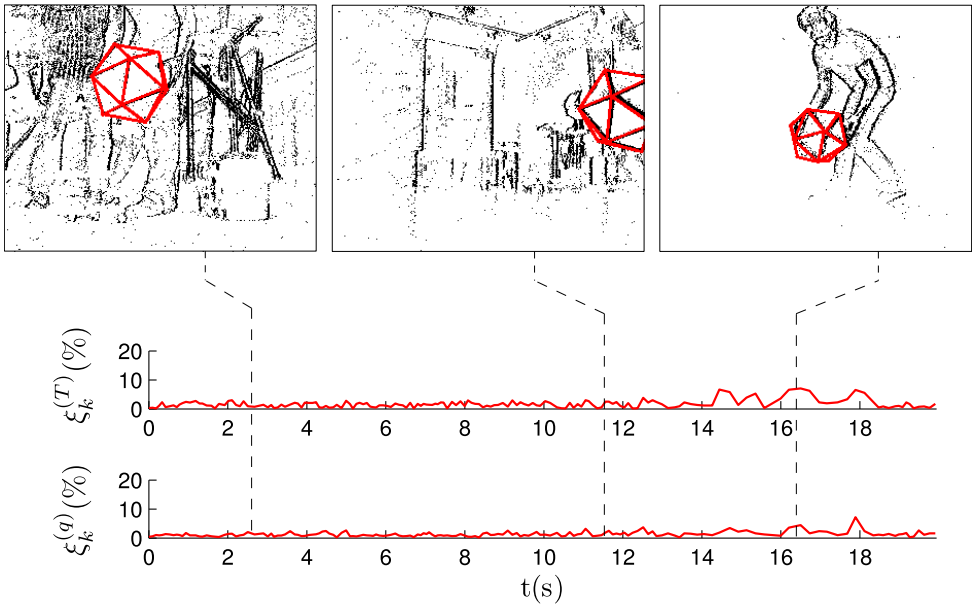



Figure 4.10: Results for the third experiment, where we recorded a hand-held icosahedron while the camera moved to follow it. Snapshots show the state of the system at some characteristic moments. The errors remain low, always below 10%.

The leftmost snapshot in Fig. 4.10 shows the state of the system while the camera is moving: as we can see, the number of events is much higher in this case than in the previous recordings, as a result of the camera not being static. Consequently, most of these events are not generated by the tracked object, but rather by other visible edges in the scene. However, we verify that pose estimation is correctly performed, since the errors remain low and the projection of the estimation is coincidental with the position of the events. In the central snapshot we can see how pose estimation is performed even when a fraction of the icosahedron has left the field of view of the camera. Finally, the rightmost snapshot shows one of the instants in which the errors reach their highest values. This happens when the object is at its furthest position from the camera, and thus when we are less precise (a pixel will represent a larger 3D distance when points are further away from the camera). However, even at this moment errors remain below 10% and the projection of the estimation is almost coincidental with the events. We conclude that pose estimation is correctly performed even in this complex scenario, providing the following mean values for the estimation errors: $\overline{\xi^{(T)}} = 1.65\%$ and $\overline{\xi^{(q)}} = 1.29\%$.

This experiment shows how the method can perform pose estimation even in complex scenarios, by simply adding some additional criteria for the matching of events.

### 4.3.4 Fast spinning object

In order to test the accuracy of the algorithm with fast moving objects, we attached the icosahedron to an electric brushless motor and recorded it at increasing angular speeds. As shown in Fig. 4.11, the icosahedron is mounted on a plane with four dots, used for ground truth. These four points are tracked using the part-based tracker described in Chapter 2.

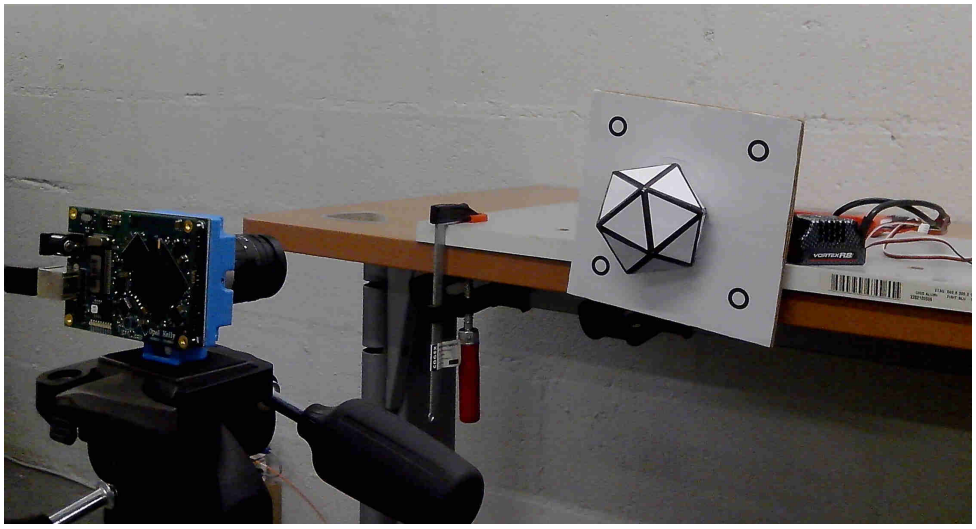


Figure 4.11: Experimental set-up for the fast spinning experiment. An icosahedron is attached to a brushless motor and recorded by the event-based camera. The four dots on the plane are used for ground truth.

Through electronic control of the motor, we created four sections during which the angular speed is approximately constant. From the obtained ground truth we can estimate the linear and angular velocities $\boldsymbol{v}$ and $\boldsymbol{w}$. The angular speed is displayed in Fig. 4.12: it reaches a maximum value of 26.4 rev/s in this case, much higher than in the previous recordings.

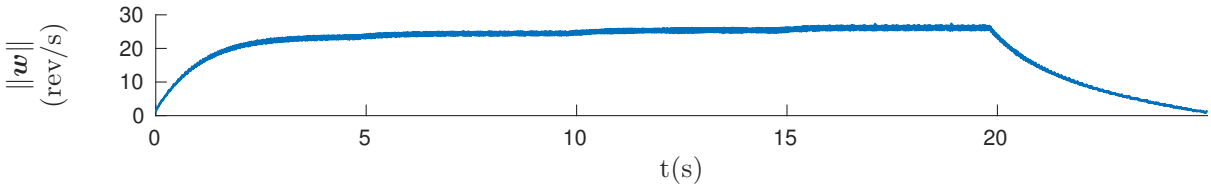



Figure 4.12: Evolution of the angular speed of the fast spinning icosahedron for the whole length of the recording.

The estimation errors are, for an experimentally selected optimal set of parameters: $\overline{\xi^{(T)}} = 1.06\%$, $\overline{\xi^{(q)}} = 3.95\%$. Errors are in this case slightly higher, notably for the estimation of the rotation. However, the mean values for the errors are low enough to conclude that, in general, the pose is correctly estimated even for objects moving at high velocity.

### 4.3.5 Degraded temporal resolution

In order to test the impact of the acquisition rate and to emphasize the importance of the high temporal resolution on the accuracy of our algorithm, we repeated the previous experiment progressively degrading the temporal resolution of recorded events. To degrade the temporal resolution, we select all the events occurring within a given time window of size $dt$ and assign the same timestamp to all of them. If several events occur at the same spatial location inside of this time window, we only keep a single one. We also shuffle the events randomly, since the order of the events contains implicit high temporal resolution information. Fig. 4.13 shows, in semi-logarithmic scale, the evolution of both the mean relative translation error and the mean relative rotation error with the size of the time window when tracking the fast spinning icosahedron with a fixed set of tuning parameters taken from the previous step. We only plot errors between 0% and 20%, since we consider the estimation to be unsuccessful for errors above 20%. The errors remain approximately stable until the time window reaches 1ms. This can be explained because the small motion assumption is experimentally satisfied for time windows of 1 ms for the typical velocity in this recording. From this point on the errors start growing, until the tracker gets completely lost for values above 10 ms.

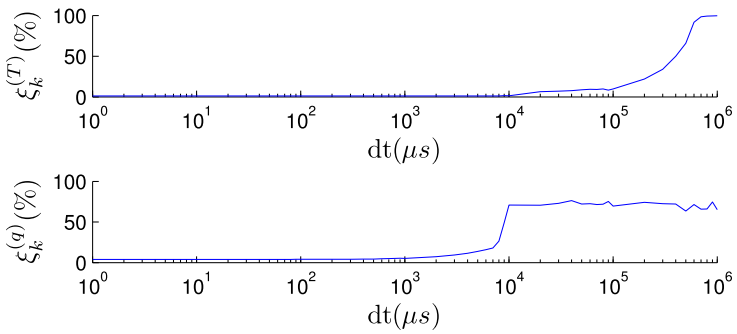



Figure 4.13: Evolution of the errors with the size of the binning window $dt$ (in µs), when tracking the fast spinning icosahedron. As the time resolution is degraded, the errors start growing, until the tracker gets completely lost for values above 10 ms.

We conclude from this experiment that the high temporal resolution of the neuromorphic camera output is a key feature to the successful performance of the 3D pose estimation algorithm. Beyond 10 ms pose estimation becomes a difficult problem. 10 ms is already smaller than the frame interval used by conventional computer vision algorithms.

### 4.3.6 Computational time

We conclude the present chapter by discussing the computational time required by the current C++ implementation of the algorithm. This analysis is performed in the same conditions as in the two previous chapters (see Sections 2.4.3 and 3.4.3 for details).

As in the previous chapters, let us consider the ratio of processing time to the length of the recording, measured every time period of 1 ms and averaged over 10 tests. We show in Fig. 4.14 results obtained when estimating the 3D pose of the icosahedron with $N = 10$. Let us remind the reader that the parameter $N$ controls how often we execute the hidden line removal algorithm, which is applied every $N$ events. Since the hidden line removal procedure is computationally demanding, the value of $N$ will have a strong impact on the computational time required by the algorithm. From Fig. 4.14 we verify

that the ratio of processing time to the length of the recording is always smaller than one when estimating the pose of the icosahedron, allowing us to conclude that we are processing this stream of events in real time.

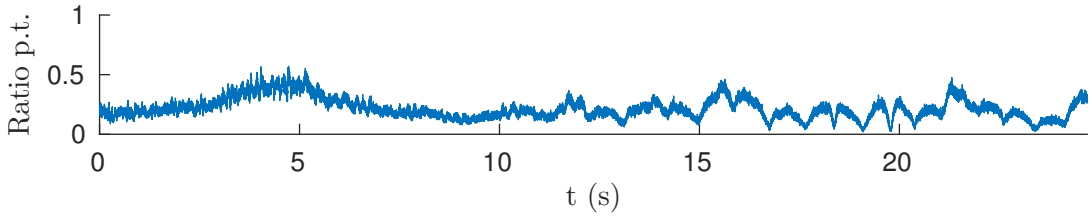



Figure 4.14: Computational time required to estimated the pose of the icosahedron with $N = 10$.

Let us next display in Fig. 4.15 the computational time as a function of the event rate for three different values of $N$. As in the previous chapters, we verify that the computational time grows with the event rate and can be approximated by a linear function. We then extrapolate and compute the maximum rate of events that can be processed in real time by our algorithm. We verify as well that the value of $N$ has a strong impact on the computational time required, resulting in different slopes for the regression lines.

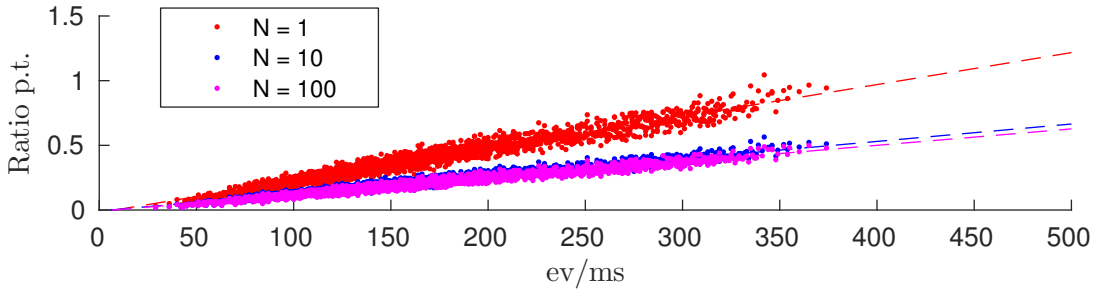



Figure 4.15: Computational time as a function of the event rate when tracking the icosahedron. We show the results obtained with three different values of the parameter $N$, which has a strong impact on the computational time..

We display in the top row of Fig. 4.16 the maximum event rate that can be processed in real time by the algorithm as a function of $N$, where $N$ takes values between 1 and 100. We verify that the maximum event rate is largely increased for the first values of $N$, but then it is almost insensitive to its value (indeed, it stabilizes after approximately $N = 25$). This can be explained because, for small values of $N$, the relative importance of the time required for applying the hidden line removal algorithm is large. Consequently, increasing the value of $N$ will have a strong impact on the computational time. As $N$ gets larger, the relative importance of this process is smaller, and increasing $N$ will have a weaker effect.

In order to properly evaluate the effect of the parameter $N$, we study its effect on the tracking errors as well: we show in the Fig 4.16 the evolution of the mean translation error $\overline{\xi^{(T)}}$ (middle row) and the mean rotation error $\overline{\xi^{(q)}}$ (bottom row) with the value of $N$, when estimating the 3D pose of the icosahedron. We verify that the tracking errors grow with $N$: this occurs because for large values of $N$ the small motion assumption is not true anymore, and thus the algorithm fails to yield correct results. In other words, when we accumulate too many events we are losing the high temporal resolution of the data, and the accuracy of the pose estimation will therefore degrade.

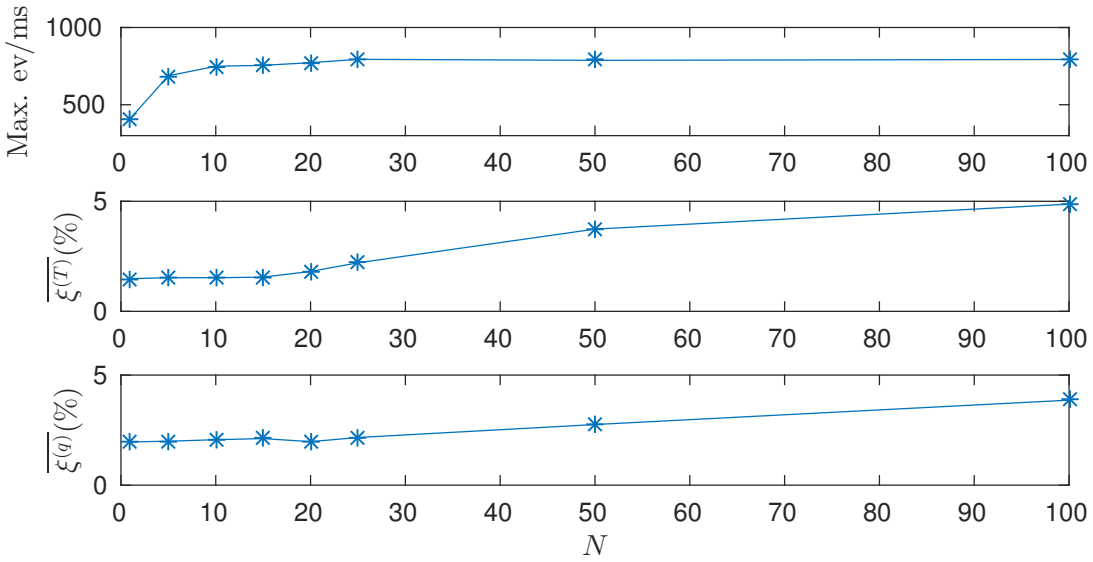



Figure 4.16: Top row: maximum event rate that can be processed in real time as a function of the parameter $N$. Middle row: evolution of the mean translation error $\overline{\xi^{(T)}}$ with the value of $N$. Bottom row: evolution of the mean rotation error $\overline{\xi^{(q)}}$ with the value of $N$

Let us next repeat the same analysis for the second recording, containing the papercraft model of a house. We show in Fig. 4.17 the ratio of processing time to the length of the recording obtained when estimating the pose of the house with $N = 10$. We verify that the required computational time is higher than in the case of the icosahedron, and the ratio sometimes exceeds the value 1. This is mainly due to the higher complexity of the hidden line removal algorithm (let us note that the model of the house is non-convex, which greatly increases the complexity of the hidden line removal step [116]).

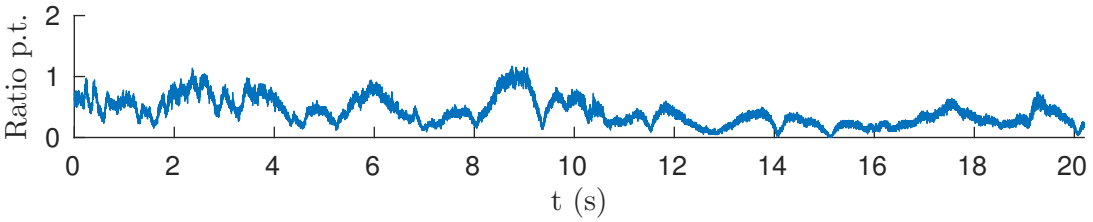



Figure 4.17: Computational time required to estimated the pose of the house with $N = 10$.

Let us also display, in Fig. 4.18 (top), the maximum event rate that can be processed by the algorithm faster than it is acquired, as a function of the value of $N$. Once again, we verify that the maximum event rate grows as $N$ increases, but it always remains smaller than in the case of the icosahedron. As previously stated, this is due to the higher complexity of the object.

We display, in the middle and bottom rows of Fig. 4.18, the evolution of the tracking errors with the value of $N$ when estimating the 3D pose of the house. We verify as well that the errors grow with the value of $N$. Nevertheless, for small values of $N$ there is a plateau in which they are very slightly affected by its value. For example, for $N = 25$ we get $\overline{\xi^{(T)}} = 3.129\%$ and $\overline{\xi^{(q)}} = 2.776\%$. Therefore, we get low values for the estimation errors while greatly reducing the required computational time. We conclude that it is possible to guarantee real-time performance even for this more complex object by slightly

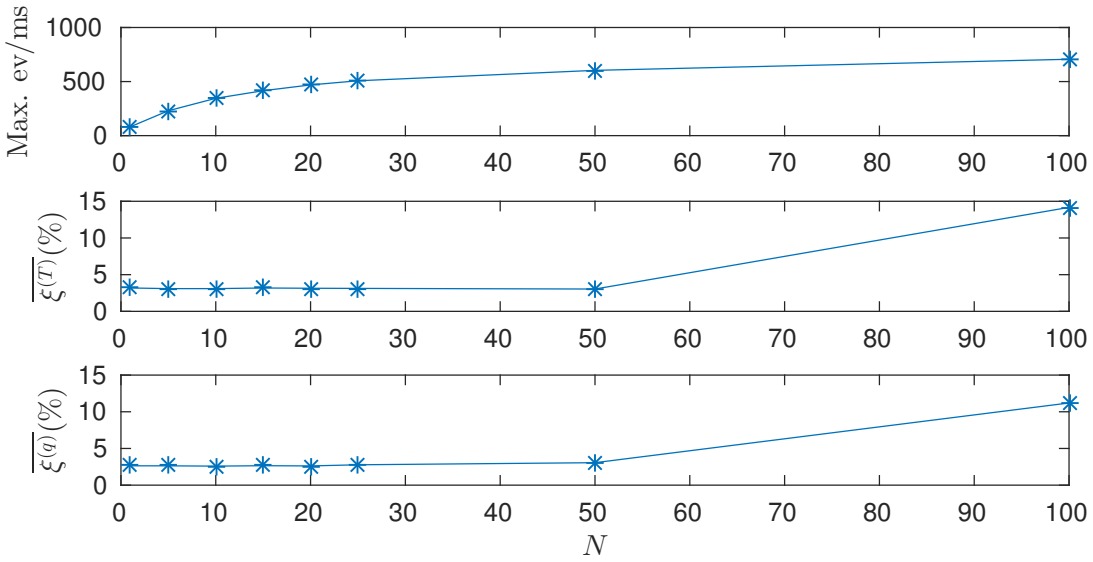



Figure 4.18: Top row: maximum event rate than can be processed in real time as a function of $N$, when considering a complex object. Middle row: evolution of the mean translation error with the value of $N$. Bottom row: evolution of the mean rotation error with the value of $N$.

increasing the value of $N$, with small effect on the accuracy.

We then conclude that the value of $N$ should be chosen according to the desired application. If real-time constraints apply, its value can be increased in order to reduce the computational time required. For the typical application, we recommend values between 10 and 20.

## 4.4 Discussion

This chapter introduces a new method for event-based 3D pose estimation. The transformation applied with each event is intuitively simple and uses the distance to the line of sight of events. Depending on the recording, we get translation errors ranging from 1.06% to 3.12% (relative to the norm of the mean translation for the considered recording) and rotation errors from 1.29% to 4.71% (relative to the maximum possible distance between two unit quaternions). These values are reasonably low for us to conclude that pose estimation is correctly performed.

We have also shown that when the temporal resolution of the events is degraded to simulate frame based conditions, a point is reached after which the pose cannot be accurately estimated. In the studied recording, this happens when the temporal resolution is 10 ms. We conclude that the high temporal resolution of the neuromorphic camera is a key feature to the accuracy of our algorithm.

The method can also be used in mobile scenarios by applying more robust matching algorithms relying on additional matching criteria, such as the local orientation of edges. The method is robust to partial occlusions and does not impose any limitation on the type of model that can be used. The only constraint is given by the increase in computational time associated with the complexity of the object, especially in computing hidden surfaces. Other models, including parametric curves or point clouds, could be used with very small modifications to the algorithm. In the case of real-time requirements, we show that the

tuning of the parameter $N$ provides lower computational times with little impact on the accuracy of the pose estimation.

The method is based on the assumption that the current estimation of the pose is close to its true value. Thanks to the high temporal resolution of the neuromorphic camera this is usually a reasonable assumption once a first estimation is available. However, it implies that a manual initialization step is usually needed. Moreover, the required 3D structure for the model of the tracked object is a complex one, including the knowledge of its vertices and edges. This motivates the research for the solution to the P$n$P problem presented in the next chapter.

# Chapter 5

# An Event-Based Solution to the Perspective-$n$-Point Problem

## 5.1 Introduction

The Perspective-$n$-Point problem—usually referred to as P$n$P—is the problem of finding the relative pose between an object and a camera from a set of $n$ pairings between 3D points of the object and their corresponding 2D projections on the focal plane, assuming that a model of the object is available. Since it was formally introduced in 1981 [41], the P$n$P problem has found numerous applications in photogrammetry and computer vision, such as tracking [121], visual servoing [122] or augmented reality [123].

For three or four points with non-collinear projections on the focal plane, the P$n$P problem can be solved up to some ambiguity in the camera pose [124]. When more points are to be considered, the standard method is to minimize the sum of some squared error, usually defined on the focal plane. Existing methods differ in the way this error function is minimized, and can be classified as iterative [101] and non iterative [109]. Other techniques consider an object-space error function instead [114], [125]. The advantage of such an approach is that a correct matching of the 3D points leads to a correct 2D projection on the focal plane, while the reverse is not necessarily true.

This chapter introduces a new approach to the problem designed to work on the output of an asynchronous event-based vision sensor. To our knowledge, this is the first P$n$P algorithm designed to work on the asynchronous output of neuromorphic cameras. In the previous chapter we presented an event-based 3D pose estimation algorithm. However, that method is based on the assumption that the estimation is always close to the true pose of the object, and thus requires a manual initialization step. The technique described in the present chapter, greatly inspired by the work of Lu *et al.* in [114], is designed to overcome this limitation.

The problem is formulated here as a least-squares minimization problem, where the error function is the object-space collinearity error [114], which is updated with every incoming event. The optimal translation is then computed in closed form. The optimal rotation, however, cannot be computed in closed form. In order to overcome this problem, we build a virtual mechanical system whose energy is proven to be equal to the error function: since mechanical systems evolve in the sense of minimizing their energy, the desired rotation will then be given by the evolution of this virtual mechanical system. This allows for a simple yet robust solution of the problem, which takes full advantage of the high temporal resolution of the sensor, as the estimated pose is incrementally

updated with every incoming event. Two approaches are proposed: the Full and the Efficient methods. These two methods are compared against a state of the art P$n$P algorithm both on synthetic and on real data, producing similar accuracy in addition of being faster.

The outline of the current chapter is as follows: we first describe the method in Section 5.2. The experimental results are then presented in Section 5.3, and the obtained results discussed in Section 5.4.

## 5.2 Event-based solution to the P$n$P problem

In this section we develop our event-based solution to the Perspective-$n$-Point problem. The task is mathematically described in 5.2.1, while the rotation formalisms used in this chapter are explained in 5.2.2. The object-space collinearity error (which we wish to minimize in order to obtain the pose), is given in 5.2.3. The optimal translation is then obtained in closed form as explained in 5.2.4, while the rotation is given by the evolution of a virtual mechanical system, as shown in 5.2.5.

### 5.2.1 Problem Description

Let us imagine a scene with a moving rigid object observed from a calibrated silicon retina, as shown in Fig. 5.1. Let $\{\mathbf{V}^{(i)}\}$ be a model of the object, described as a collection of 3D points $\mathbf{V}^{(i)} = \left[X^{(i)}, Y^{(i)}, Z^{(i)}\right]^T$. Attached to this object there is a frame of reference, whose origin we denote as $\mathbf{V}^{(0)}$.

We will keep the same notation and formalisms for the representation of 3D pose used in the previous chapter, that we will briefly remind here. According to this, the pinhole projection maps 3D points $\mathbf{V}^{(i)}$ expressed in the object's frame of reference into $\mathbf{v}^{(i)}$ on the camera's focal plane, according to the relation:

$$\begin{bmatrix} \mathbf{v}^{(i)} \\ 1 \end{bmatrix} \sim \Lambda \begin{bmatrix} R & \boldsymbol{T} \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(i)} \\ 1 \end{bmatrix}, \tag{5.1}$$

where $\Lambda$ is the $3 \times 3$ matrix defining the camera's intrinsic parameters, while $R \in SO(3)$, $\boldsymbol{T} \in \mathbb{R}^3$ are the extrinsic ones, also referred to as the relative pose between the object and the camera [113]. The intrinsic parameters matrix $\Lambda$ is obtained through a prior calibration procedure, while estimating $R$, $\boldsymbol{T}$ is the objective of our method. The sign $\sim$ in (5.1) indicates that the equality is defined up to a scale [112]. In the following, we will denote $\widehat{R}$, $\widehat{\boldsymbol{T}}$ the estimated pose, that we update with the incoming events. The corresponding points of the estimation are denoted $\widehat{\mathbf{V}}^{(i)}$ and computed with the expression:

$$\widehat{\mathbf{V}}^{(i)} = \widehat{R}\mathbf{V}^{(i)} + \widehat{\boldsymbol{T}}. \tag{5.2}$$

Analogously, the origin of the frame of reference attached to the estimation is denoted $\widehat{\mathbf{V}}^{(0)}$.

### 5.2.2 Rotation formalisms

The usual rotation formalisms most frequently employed have been presented in section 4.2.2. The method introduced in the current chapter will be developed using both rotation matrices and the axis-angle representation, where the rotation is expressed as the rotation

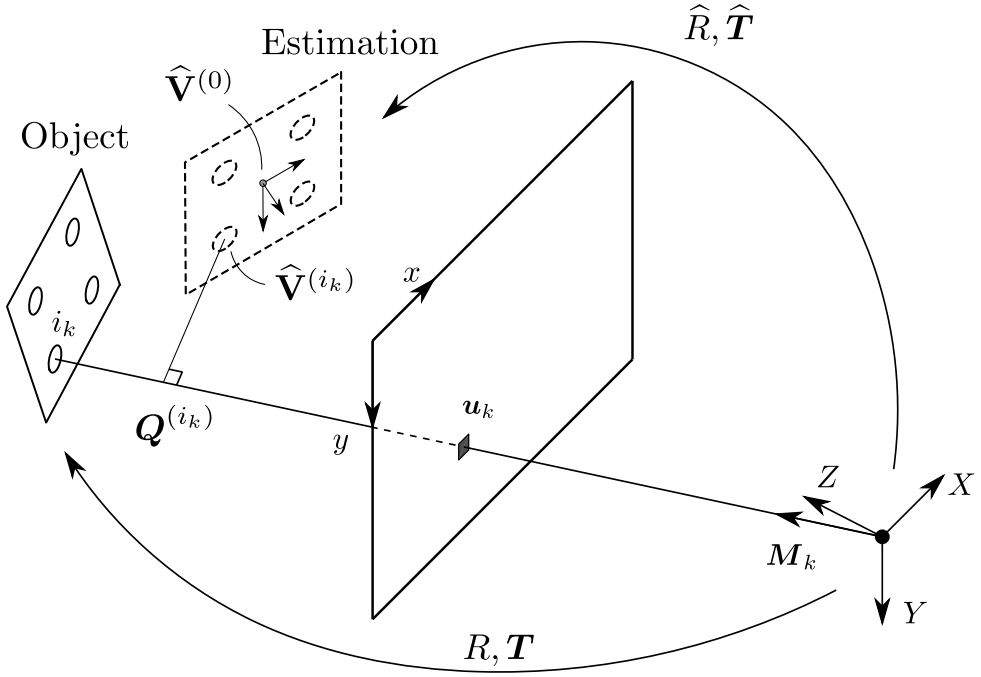



Figure 5.1: An object, given as a collection of 3D points $\{\mathbf{V}^{(i)}\}$, is observed by a calibrated silicon retina. The true pose of the object is given by $R$, $\boldsymbol{T}$, while the estimated pose is denoted $\widehat{R}$, $\widehat{\boldsymbol{T}}$. Attached to the estimation there is a frame of reference, whose origin we denote by $\widehat{\mathbf{V}}^{(0)}$. An event $\mathbf{e}_k$ has to be generated by a point lying on the *line of sight* of the event, whose direction is given by the vector $\boldsymbol{M}_k$. The point of the object generating event $\mathbf{e}_k$ is denoted by the index $i_k$. $\boldsymbol{Q}^{(i_k)}$ is then the projection of $\mathbf{V}^{(i_k)}$ on its corresponding line of sight. When the estimation is aligned with the true position of the object, then $\boldsymbol{Q}^{(i_k)}$ and $\widehat{\mathbf{V}}^{(i_k)}$ are the same.

vector $\boldsymbol{r} = \theta\tilde{\boldsymbol{r}}$, with $\tilde{\boldsymbol{r}}$ a unit vector in the direction of the axis of rotation and $\theta$ the rotation angle.

### 5.2.3 Object-space collinearity error

Let us consider an event $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T$ occurring at time $t_k$ at location $\boldsymbol{u}_k = [x_k, y_k]^T$. According to the pinhole camera model, we know that this event has to be generated by a point lying on the *line of sight* of event $\mathbf{e}_k$—the line defined by the optical center and the spatial position of the event on the focal plane—as shown in Fig. 5.1. Assuming that we can identify which point of the object has generated the event, we try to estimate the pose that minimizes the orthogonal projection errors on the line of sight for the last $n$ events.

Let $\boldsymbol{M}_k$ be a vector defining the line of sight of event $\mathbf{e}_k$, whose coordinates can be easily obtained as:

$$\boldsymbol{M}_k = \Lambda^{-1} \begin{bmatrix} \boldsymbol{u}_k \\ 1 \end{bmatrix}, \tag{5.3}$$

where $\Lambda$ represents the intrinsic parameters matrix of the camera.

Next, let us assume that we can identify the point of the object that has generated event $\mathbf{e}_k$, that we denote by the index $i_k$. Hence, if the true pose of the object and the estimation were perfectly aligned, $\widehat{\mathbf{V}}^{(i_k)}$ would necessarily lie on the line of sight of event $\mathbf{e}_k$.

Let $\boldsymbol{Q}_k$ be the projection of $\widehat{\mathbf{V}}^{(i_k)}$ on the line of sight of event $\mathbf{e}_k$, that can be computed as:

$$\boldsymbol{Q}_k = \frac{\left(\widehat{\mathbf{V}}^{(i_k)}\right)^T \boldsymbol{M}_k}{\|\boldsymbol{M}_k\|^2} \boldsymbol{M}_k = L_k \widehat{\mathbf{V}}^{(i_k)}, \tag{5.4}$$

where $L_k$ is the *Line-of-Sight* projection matrix of event $\mathbf{e}_k$. It takes the value:

$$L_k = \frac{\boldsymbol{M}_k \boldsymbol{M}_k^T}{\boldsymbol{M}_k^T \boldsymbol{M}_k}. \tag{5.5}$$

For a given event $\mathbf{e}_k$, we define the object-space collinearity error $\boldsymbol{\xi_k}(\widehat{R}, \widehat{\boldsymbol{T}})$ as:

$$\boldsymbol{\xi}_k(\widehat{R}, \widehat{\boldsymbol{T}}) = \boldsymbol{Q}_k - \widehat{\mathbf{V}}^{(i_k)} = (L_k - I)\widehat{\mathbf{V}}^{(i_k)} = (L_k - I)(\widehat{R}\mathbf{V}^{(i_k)} + \widehat{\boldsymbol{T}}), \tag{5.6}$$

where $I$ denotes the $3 \times 3$ identity matrix. We will take into account the last $n$ events and minimize the weighted mean of their squared collinearity errors. The goal of our algorithm will therefore be to minimize the following error function $E_k(\widehat{R}, \widehat{\boldsymbol{T}})$:

$$E_k(\widehat{R}, \widehat{\boldsymbol{T}}) = \frac{1}{\Omega_k} \sum_{j=0}^{n-1} \omega_{k,j} \|\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\|^2, \tag{5.7}$$

where we are considering the event-based weighted mean introduced in Section 2.3.1 and detailed in the Appendix A.

We will then be looking for the pose $\widehat{R}, \widehat{\boldsymbol{T}}$ that minimizes the error function given by (5.7), which is computed as the weighted mean of the collinearity errors for the last $n$ events. These collinearity errors depend on the estimated position of the point generating the event (where we assume that we can identify the point) and the position of the corresponding event on the focal plane (or, equivalently, the projection of the point generating the event). Consequently, our approach can be classified as a solution to the P$n$P problem, since we are estimating the pose of an object from a set of $n$ pairings between 3D points of the object and their projections on the focal plane. Unlike classical frame-based techniques, our approach allows us to consider several events generated by the same point of the object, and thus $n$ can be chosen to be bigger than the number of points conforming the object.

**Remark:** P$n$P methods always require matching 3D points with their corresponding 2D projections. In the case of our method, this equals to identifying which point of the object has generated an event. Consequently, our algorithm relies on an event-based tracking technique. For the rest of this chapter, the term "tracking" will always refer to this previous method. As we will show in the experiments, the overall performance of the system is strongly dependent on the accuracy of this tracking.

### 5.2.4 Optimal Translation

For a given rotation $\widehat{R}$, the optimal translation that minimizes the sum of the squared collinearity errors can be computed in closed form [114]. Equivalently, if the previous estimation of the pose is given by $\widehat{R}_{k-1}$, $\widehat{\boldsymbol{T}}_{k-1}$, the optimal displacement $\Delta\boldsymbol{T}_k$ can be computed as:

$$\Delta\boldsymbol{T}_k(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}) = A_k^{-1} \boldsymbol{B}_k, \tag{5.8}$$

where $A_k$ is a $3 \times 3$ matrix and $\boldsymbol{B}_k$ a 3D vector given by:

$$A_k = \sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j}), \tag{5.9}$$

$$\boldsymbol{B}_k = \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}_{k-1}^{(i_{k-j})}, \tag{5.10}$$

where $\widehat{\mathbf{V}}_{k-1}^{(i_{k-j})}$ denotes the position of $\mathbf{V}^{(i_{k-j})}$ estimated with $\widehat{R}_{k-1}$, $\widehat{\boldsymbol{T}}_{k-1}$:

$$\widehat{\mathbf{V}}_{k-1}^{(i_{k-j})} = \widehat{R}_{k-1}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}}_{k-1}. \tag{5.11}$$

The mathematical development yielding this result is included in the Appendix H.

We will refer to this way of computing $A_k$ and $\boldsymbol{B}_k$ as the *full method*. As shown in [114], $A_k$ can be proven to be non-singular, guaranteeing that (5.8) can always be solved. We then update the estimation of the position making:

$$\widehat{\boldsymbol{T}}_k = \widehat{\boldsymbol{T}}_{k-1} + \lambda\Delta\boldsymbol{T}_k, \tag{5.12}$$

where $\widehat{\boldsymbol{T}}_k$ denotes the estimated translation at time $t_k$, and $\lambda$ is a tuning factor. Let us note that $\lambda$ is a dimensionless quantity, and it should always be chosen smaller or equal to one. Its effect will be more carefully studied in the experiments.

As shown in the Appendix I, if we choose the standard set of weights presented in the Appendix A.1 and under some reasonable assumptions, $A_k$ and $\boldsymbol{B}_k$ can be iteratively updated making:

$$A_k \approx \omega_0(I - L_k) + (1 - \omega_0)A_{k-1}, \tag{5.13}$$

$$\boldsymbol{B}_k \approx \omega_0(L_k - I)\widehat{\mathbf{V}}_{k-1}^{(i_k)} + (1 - \omega_0)\boldsymbol{B}_{k-1}. \tag{5.14}$$

This allows us to update $A_k$ and $\boldsymbol{B}_k$ for each event in an iterative manner, saving memory and computational time. We will refer to this way of updating $A_k$ and $\boldsymbol{B}_k$ as the *efficient method*, and test its effect on the experiments.

### 5.2.5 Rotation

As shown in [114], for a given translation $\widehat{\boldsymbol{T}}$ the optimal rotation $\widehat{R}$ cannot be computed in closed form. In [114], the rotation is obtained via an absolute orientation problem between the points of the estimation and their projections onto the corresponding line of sight, which is then solved using Singular Value Decomposition. This is, however, a computationally expensive process, not well-suited to the output of the neuromorphic camera: in order to fully exploit the high dynamics of the sensor, we wish to update the estimated pose with every incoming event. Given the high frequency of the arrival of events, the computations carried out with each one of them should be kept to a minimum, in order to achieve real-time performance.

In our approach, instead of trying to find the optimal rotation for each event, we will simply apply a rotation such that our error function is reduced at each step. Since events happen with such a high temporal resolution, this will very fast lead to a correct estimation. To that end, we will define a virtual mechanical system whose energy is equal

to the error function. Since mechanical systems evolve in the sense of minimizing their energy, simulating the behavior of this system will be equivalent to minimizing the error function, approaching the estimation towards its true value.

Consequently, let us picture the following virtual mechanical system: since rotations happen around the origin of the estimation $\widehat{\mathbf{V}}^{(0)}$, let us imagine $\widehat{\mathbf{V}}^{(0)}$ to be attached to the world by a spherical joint, as shown in Fig. 5.2. This allows the object to freely rotate around this point, but prevents any translation. Next, for every event $\mathbf{e}_{k-j}$ with $j = 0, 1, ..., n-1$ (that is to say, for the last $n$ events) we wish to attract the corresponding point of the estimation $\widehat{\mathbf{V}}^{(i_{k-j})}$ towards the line of sight of the event. To that end, let us imagine $\widehat{\mathbf{V}}^{(i_{k-j})}$ and the line of sight to be linked by a linear spring, whose direction is always perpendicular to the line of sight. In other words, we link $\widehat{\mathbf{V}}^{(i_{k-j})}$ and $\boldsymbol{Q}_{k-j}$ by a linear spring. In a real mechanical system, this would be achieved by linking the spring and the line of sight with a cylindrical joint, as shown in Fig. 5.2.

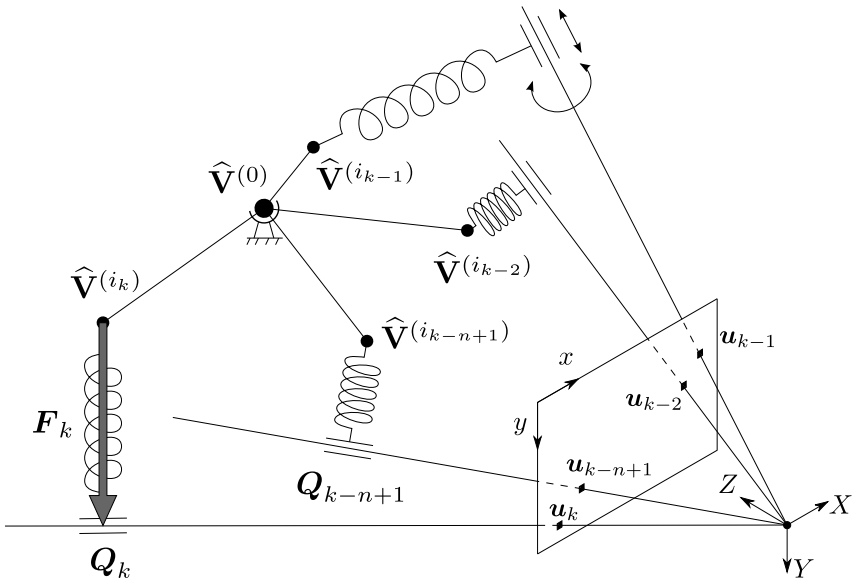



Figure 5.2: In order to solve the rotation we build the following virtual mechanical system: the origin of the estimation $\widehat{\mathbf{V}}^{(0)}$ is linked to the world by a spherical joint, and every point of the estimation $\widehat{\mathbf{V}}^{(i_{k-j})}$ generating an event $\mathbf{e}_{k-j}$ is linked to its corresponding line of sight by a linear spring. Simulating the behavior of this mechanical system is equivalent to minimizing the collinearity error.

Let $\boldsymbol{F}_{k-j}$ denote the force exerted by the virtual spring attached to some point of the estimation $\widehat{\mathbf{V}}^{(i_{k-j})}$, which has been assigned to the event $\mathbf{e}_{k-j}$. This force will be dependent on the estimated pose: $\boldsymbol{F}_{k-j} = \boldsymbol{F}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})$.

As previously explained in section 2.3.2, the force exerted by a linear spring is given by Hooke's law, which states that the direction of the force is that of the axis of the spring, while its magnitude is given by the expression:

$$\|\boldsymbol{F}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\| = C_{k-j}\Delta l_{k-j} = C_{k-j}\big(l_{k-j} - l_0\big), \tag{5.15}$$

where $C_{k-j}$ is the stiffness of the spring and $\Delta l_{k-j} = l_{k-j} - l_0$ its elongation. Since the axis of the spring is aligned with $\boldsymbol{Q}_{k-j} - \widehat{\mathbf{V}}^{(i_{k-j})}$, and considering (5.6), $\boldsymbol{F}_{k-j}$ takes the value:

$$\boldsymbol{F}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) = \frac{\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})}{\|\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\|} C_{k-j}\Delta l_{k-j}. \tag{5.16}$$

Next, let us make $l_0 = 0$ for all virtual springs. This implies that the elongation at rest is zero: in other words, the virtual spring will not produce any force when $\widehat{\mathbf{V}}^{(i_{k-j})}$ lies on its corresponding line of sight, that is to say when it is correctly aligned with the corresponding event. The elongation $\Delta l_{k-j}$ then takes the value:

$$\Delta l_{k-j} = l_{k-j} = \|\boldsymbol{Q}_{k-j} - \widehat{\mathbf{V}}^{(i_{k-j})}\| = \|\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\|. \tag{5.17}$$

Finally, let us make the magnitude of the stiffness equal to the weight of the corresponding event:

$$C_{k-j} = \beta\omega_{k,j}, \tag{5.18}$$

where $\beta$ is just a unit adjustment constant, that compensates for the fact that weights are dimensionless but not the stiffness. For the rest of this chapter all distances will be given in mm, and thus $\beta = 1$ Nmm$^{-1}$. $\boldsymbol{F}_{k-j}$ becomes:

$$\boldsymbol{F}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) = \beta\omega_{k,j}\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}). \tag{5.19}$$

Let us remind the reader that the energy $g_{k-j}$ of a linear spring is given by the expression:

$$g_{k-j} = \frac{1}{2}C_{k-j}(\Delta l_{k-j})^2, \tag{5.20}$$

which, considering (5.18) and (5.17) becomes:

$$g_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) = \beta\omega_{k,j}\|\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\|^2. \tag{5.21}$$

The energy $G_k$ of the whole virtual when considering the last $n$ events is then computed by applying the principle of superposition:

$$G_k(\widehat{R}, \widehat{\boldsymbol{T}}) = \sum_{j=0}^{n-1} g_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) = \frac{1}{2}\sum_{j=0}^{n-1} \beta\omega_{k,j}\|\boldsymbol{\xi}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}})\|^2 = \frac{\beta\Omega_k}{2}E_k(\widehat{R}, \widehat{\boldsymbol{T}}). \tag{5.22}$$

According to this equation, we have built a virtual mechanical system whose energy is equal to the error function, up to some normalization factor. Since mechanical systems evolve in the sense of minimizing their energy, simulating the behavior of this system will then be equivalent to minimizing the error function.

Since the translation is prevented in this case, we only wish to compute the moments of the forces and their effect. Thus, let $\boldsymbol{\tau}_{k-j}$ be the torque generated by force $\boldsymbol{F}_{k-j}$ with respect to the origin of the estimation $\widehat{\mathbf{V}}^{(0)}$:

$$\begin{aligned} \boldsymbol{\tau}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) &= (\widehat{\mathbf{V}}^{(i_{k-j})} - \widehat{\mathbf{V}}^{(0)}) \times \boldsymbol{F}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) \\ &= \widehat{R}\mathbf{V}^{(i_{k-j})} \times \beta\omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}^{(i_{k-j})}, \end{aligned} \tag{5.23}$$

where $\times$ denotes the cross product. The resulting torque $\boldsymbol{\Gamma}_k$ when we take into account the last $n$ events takes the value:

$$\boldsymbol{\Gamma}_k(\widehat{R}, \widehat{\boldsymbol{T}}) = \sum_{j=0}^{n-1} = \boldsymbol{\tau}_{k-j}(\widehat{R}, \widehat{\boldsymbol{T}}) = \sum_{j=0}^{n-1} \widehat{R}\mathbf{V}^{(i_{k-j})} \times \beta\omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}^{(i_{k-j})}, \tag{5.24}$$

We will compute the resulting torque using this expression when applying the *full method*. As in the case of the translation, we will compute its value using the previous

estimated position $\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}$. We then approximate its effect by a rotation given, in its axis-angle representation, by the vector $\boldsymbol{r}_k$ computed as:

$$\boldsymbol{r}_k = \phi \boldsymbol{\Gamma}_k(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}), \tag{5.25}$$

where $\phi$ is a tuning factor. A complete justification of this choice is given in the Appendix J.

In the Appendix K we give some more insight on how to pick a value for $\phi$, and derive the following expression for its theoretical optimum $\phi^{(opt)}$:

$$\phi^{(opt)} = \frac{3\pi}{2(1+\sqrt{2})} \frac{1}{\beta \Omega_k (\rho^{(max)})^2}, \tag{5.26}$$

where $\rho^{(max)}$ is equal to the maximum distance in the object $\rho^{(max)} = \max_i \{\|\mathbf{V}^{(i)}\|\}$. From this expression it is evident that $\phi$ is not dimensionless in this case, and its optimal value will therefore depend on the dimensions of the object whose pose we want to estimate (and the units in which they are expressed). For the rest of this chapter, all values of $\phi$ will be expressed in $\mathrm{N}^{-1}\mathrm{mm}^{-1}$.

Let $\Delta R_k$ be the rotation matrix corresponding to the rotation represented by $\boldsymbol{r}_k$. We update the estimation with the following expression:

$$\widehat{R}_k = \Delta R_k \widehat{R}_{k-1}. \tag{5.27}$$

As in the case of $A_k$ and $\boldsymbol{B}_k$, the resulting torque can be approximated with the iterative expression:

$$\boldsymbol{\Gamma}_k \approx \beta \omega_0 \widehat{R}_{k-1} \mathbf{V}^{(i_k)} \times (L_k - I) \widehat{\mathbf{V}}_{k-1}^{(i_k)} + (1-\omega_0) \boldsymbol{\Gamma}_{k-1}, \tag{5.28}$$

where, as previously stated, $\widehat{\mathbf{V}}_{k-1}^{(i_k)}$ denotes the position of $\mathbf{V}^{(i_{k-j})}$ estimated with $\widehat{R}_{k-1}$, $\widehat{\boldsymbol{T}}_{k-1}$, as given by (5.11). The value of the resulting torque will be updated in this way when applying the *efficient method*.

### 5.2.6 Global algorithm

The P$n$P problem is solved by the global algorithm described below.

## 5.3 Results

In this section, two experiments showing the accuracy of our method are presented. The algorithm is implemented in Matlab and C++ and tested in a synthetic scene for the first experiment. Next, another experiment is produced from a real recording.

In order to characterize the accuracy of our method we will consider the sum of the squared collinearity errors $E_k$. Additionally, we adopt the following metrics in the space of rigid motions:

- The absolute estimation error in linear translation at time $t_k$ is given by the norm of the difference between the estimated translation $\widehat{\boldsymbol{T}}_k$ and its true value $\boldsymbol{T}_k$. We define the relative translation error $\xi_k^{(T)}$ as:

$$\xi_k^{(T)}(\%) = 100 \, \frac{\|\widehat{\boldsymbol{T}}_k - \boldsymbol{T}_k\|}{\|\overline{\boldsymbol{T}}\|}, \tag{5.29}$$

**Algorithm 5** Event-Based P$n$P algorithm

**Require:** $\mathbf{e}_k = (\boldsymbol{u}_k^T, t_k, p_k)^T \ \forall k > 0$
**Ensure:** $\widehat{R}$, $\widehat{\boldsymbol{T}}$
  Initialize the parameters
  **for** every incoming event $\mathbf{e}_k$ **do**
    Identify the point generating the event $i(k)$
    Compute the *Line-of-Sight* projection matrix $L_k$ using (5.5)
    **if** *full method* **then**
      Compute the resulting torque $\boldsymbol{\Gamma}_k$ using (5.24)
      Compute $A_k$ and $\boldsymbol{B}_k$ using (5.9) and (5.10)
    **else if** *efficient method* **then**
      Update the resulting torque $\boldsymbol{\Gamma}_k$ using (5.28)
      Update $A_k$ and $\boldsymbol{B}_k$ using (5.13) and (5.14)
    **end if**
    Compute the resulting rotation $\Delta R_k$ using (5.25)
    Compute the resulting displacement $\Delta \boldsymbol{T}_k$ using (5.8)
    Update $\widehat{R}$ using (5.27)
    Update $\widehat{\boldsymbol{T}}$ using (5.12)
  **end for**

where $\|\overline{\boldsymbol{T}}\|$ is the norm of the mean translation of the object for the whole experiment.

- The distance $d$ between two rotations, given by the corresponding rotation matrices $R_1$ and $R_2$ can be computed as:

$$d(R_1, R_2) = \|I - R_1 R_2^T\|_F, \tag{5.30}$$

where $I$ is the $3 \times 3$ identity matrix and $\|\cdot\|_F$ denotes the Frobenius norm of the matrix. This can be proven to be a metric in the space of 3D rotations [119] and takes values in the range $[0, 2\sqrt{2}]$. Thus, let $\xi_k^{(R)}$ be the relative rotation error computed as:

$$\xi_k^{(R)}(\%) = 100 \ \frac{d(\widehat{R}_k, R_k)}{2\sqrt{2}}. \tag{5.31}$$

When the *full method* is employed, the weights of the past events are chosen to be linearly decaying. Imposing $\Omega_k = 1$ yields:

$$\omega_{k,j} = \omega_j = \frac{2(n-j)}{n(n+1)}, \ \forall j = 0, ..., n-1. \tag{5.32}$$

### 5.3.1 Synthetic scene

The algorithm is first tested in a synthetic scene containing a virtual object. This object is composed by 10 points whose 3D coordinates were randomly initialized following a normal distribution with zero mean and standard deviation equal to 10 mm. Both the object and the camera are assumed to be static, and the pose of the object relative to

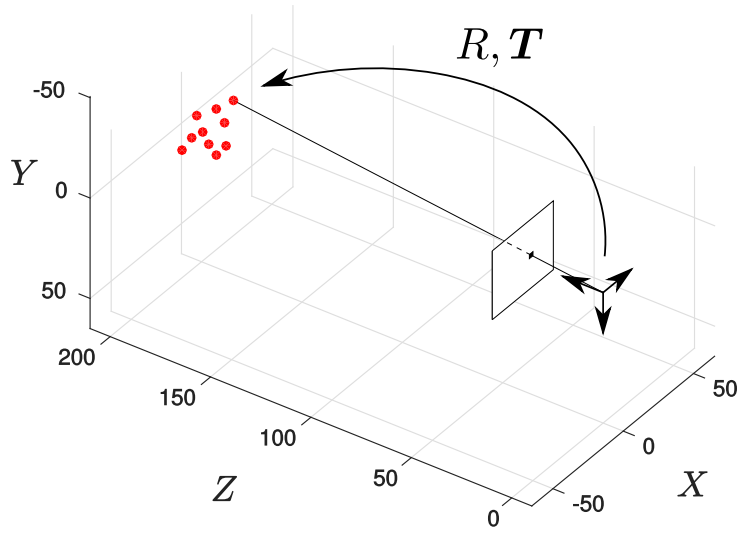



Figure 5.3: Synthetic scene: the solid dots represent the pose of the object, static in this experiment. We randomly select a point of the object and generate an event located on its projection on the focal plane.

the camera is given by the translation vector $\boldsymbol{T} = (0, 0, 200)^T$ (in mm) and the rotation vector $\boldsymbol{r} = (2/3, 2/3, 1/3)^T$. Fig. 5.3 shows the resulting geometry.

The virtual camera has the following intrinsic parameters matrix:

$$\Lambda = \begin{bmatrix} fm_x & 0 & c_x \\ 0 & fm_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \text{ with } \begin{cases} f = 20 \text{ mm} \\ m_x = m_y = 30 \text{ px/mm} \\ c_x = 152 \text{ px} \\ c_y = 120 \text{ px} \end{cases}$$

which corresponds to an ideal pinhole camera model. The precise geometric parameters are those of an ATIS device equipped with an objective with focal distance 20 mm.

A stream of events is generated from this synthetic scene, sequentially selecting a random point of the object and generating an event on its corresponding projection on the focal plane. The inter-event times are random integers following a normal distribution with mean 5 µs and standard deviation 2 µs (corresponding to some characteristic values observed in ATIS recordings of real moving objects). Let us note, however, that we are not trying to accurately simulate the event generation mechanism of neuromorphic image sensors. For this first experiment we are just trying to evaluate the algorithm when the object is static and assuming perfect tracking. Even if the events are not generated in a realistic fashion, this will allow us to characterize different aspects of the algorithm in the simplest possible situation. We will evaluate our method on real recordings in the next experiment.

In a first step we test the accuracy of the rotation and the translation estimation strategies separately. This will allow us to explore the space of parameters and give some guidance on how to set them.

### Translation

In order to exclusively check how the algorithm estimates translation, we make the initial estimation of the rotation $\widehat{R}(0)$ equal to its true value $R$. Additionally, we set the tuning

factor for the rotation $\phi$ as being zero. Let us remind the reader that the object is in this case static, and thus the estimated rotation will remain equal to its true value at every instant. We make the initial estimation of the translation $\widehat{\boldsymbol{T}}(0) = (0, 0, 0)^T$.

Let us first apply the *full method* to the synthetic stream of events making $n = 20$. Fig. 5.4(a) shows the evolution of both the sum of the squared collinearity errors $E_k$ and the relative translation error $\xi_k^{(T)}$ with the incoming events for four different values of $\lambda$. We do not plot the relative rotation error $\xi_k^{(R)}$, since it will always be zero in this case. Let us note that, for the first $n$ events, we cumulate the information (updating $A_k$, $\boldsymbol{B}_k$ and $\boldsymbol{\Gamma}_k$) but we do not update the estimation of the pose. Consequently, the relative translation error remains stable. $E_k$ is not stable because, in general, different points of the object will yield different collinearity errors. After $n$ events we start updating the estimation, and we can see how both errors decay towards zero.

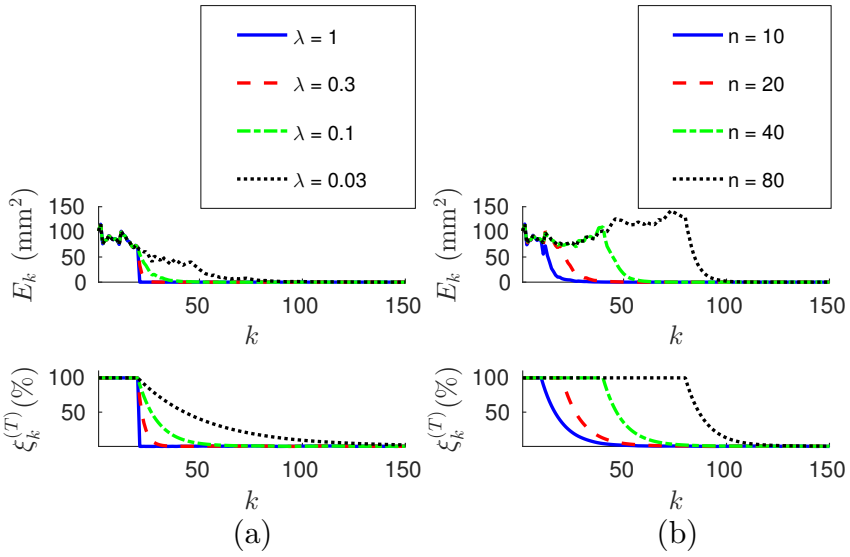



Figure 5.4: (a) Evolution of the sum of the squared collinearity errors $E_k$ and the relative translation error $\xi_k^{(T)}$ with the number of iterations for four different values of $\lambda$, when applying the *full method* with $n = 20$. The bigger $\lambda$, the faster the convergence. (b) Evolution of $E_k$ and $\xi_k^{(T)}$ for four values of $n$ when $\lambda = 0.1$. We do not start updating the estimation until we have accumulated $n$ events. After that point, the behavior of the system is very similar for every value of $n$.

Comparing the results obtained with different values of $\lambda$, we verify that the bigger $\lambda$ the faster the convergence. As a matter of fact, we could consider making $\lambda = 1$. This value of $\lambda$ makes the estimated translation equal to the optimal one for the last $n$ events at every iteration. If we are confident enough in the accuracy of our tracking this constitutes an acceptable strategy.

Fig. 5.4(b) shows the evolution of $E_k$ and $\xi_k^{(T)}$ for four different values of $n$, with $\lambda = 0.1$. We can see how the estimation is not updated until $n$ events have elapsed, but once it does the behavior of the system is very similar for all values of $n$. This can be explained because the object is in this case static and the tracking is perfect. In real world scenarios, choosing $n$ will require a tradeoff between acceptable velocities of the object—a smaller $n$ results in a shorter reaction time—and stability in the presence of tracking errors—if $n$ is big enough we can expect tracking errors to be canceled out. In order to illustrate this point, let us simulate some inaccuracy in the tracking and evaluate

the algorithm again.

To that end, we will assume that events are correctly assigned to the points generating them, but their position is noisy. This would correspond to a real case in which we are estimating the position of some markers, but there is some inaccuracy in this estimation. We will model the tracking errors as a gaussian noise with zero mean and standard deviation equal to $\sigma$. We plot in Fig. 5.5 the evolution of the relative translation error $\xi_k^{(T)}$ for four different values of $\sigma$ (in pixels) and $n$. We verify that an inaccurate tracking strongly degrades the performance of our algorithm, resulting in increasing values for the final error. Additionally, for small values of $n$ the estimated pose has a greater variance as the inaccuracy grows.

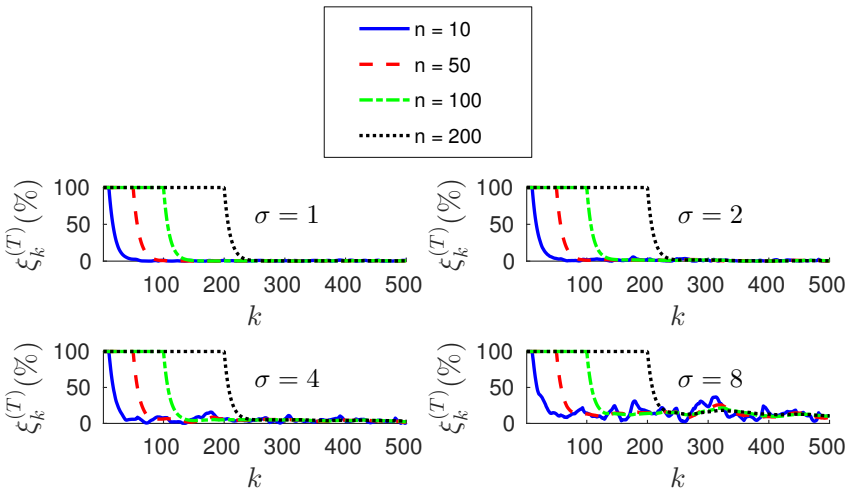



Figure 5.5: Evolution of the relative translation error $\xi_k^{(T)}$ when the tracking is not perfectly accurate. As the inaccuracy $\sigma$ grows, the results of the pose estimation are strongly degraded. Additionally, when $n$ is small the estimated pose has a greater variance.

We next evaluate the effect of matching errors, i.e. when events are not correctly assigned to the point of the object generating them. Thus, we take a percentage of the events and randomly assign them to some other point of the object. We show in Fig. 5.5 the evolution of the relative translation error $\xi_k^{(T)}$ for four different percentages of wrongly assigned events. Again, we observe that matching errors degrade the performance of the algorithm.

These results allow us to conclude that the overall performance of the system is strongly dependent on the accuracy of the tracking. Additionally, we conclude that $n$ should be chosen to be big enough to assure stability in the presence of tracking errors. The effect of this parameter will be more deeply analyzed in the next experiment, when treating real recordings. Let us note that increasing $n$ will also result in a greater computation time.

Next, let us estimate the translation by applying the *efficient method* to the synthetic stream of events. Fig. 5.7(a) shows the evolution of both $E_k$ and $\xi_k^{(T)}$ for four different values of $\lambda$, when $\omega_0 = 0.1$. Let us note that when applying this strategy we do not set the value of $n$, and thus we will start updating the estimation from the first incoming event. As we can see in Fig. 5.7(a), both errors decay towards zero in this case as well, and the convergence is faster for bigger values of $\lambda$. When applying the *efficient method*, however, big values of $\lambda$ will cause the system to oscillate. Choosing $\lambda$ will consequently require a tradeoff between speed of convergence and stability of the system.

Fig. 5.7(b) shows the evolution of $E_k$ and $\xi_k^{(T)}$ for four different values of $\omega_0$, when

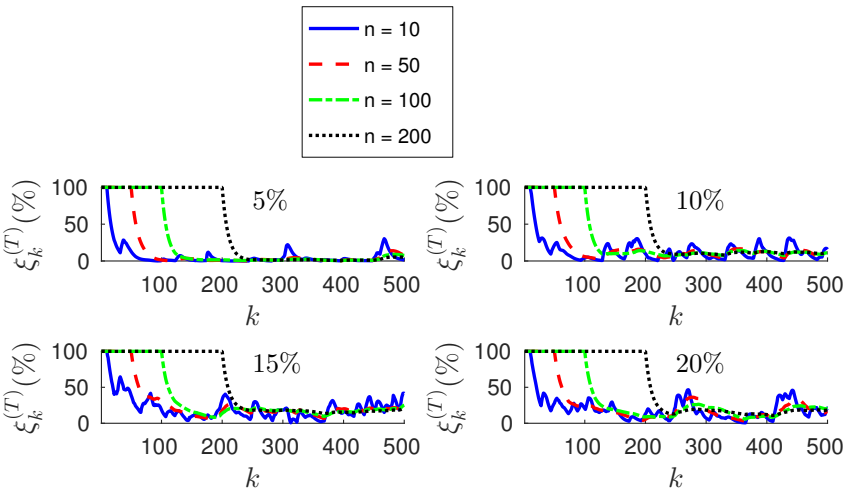



Figure 5.6: Evolution of $\xi_k^{(T)}$ with different percentages of wrongly assigned events.

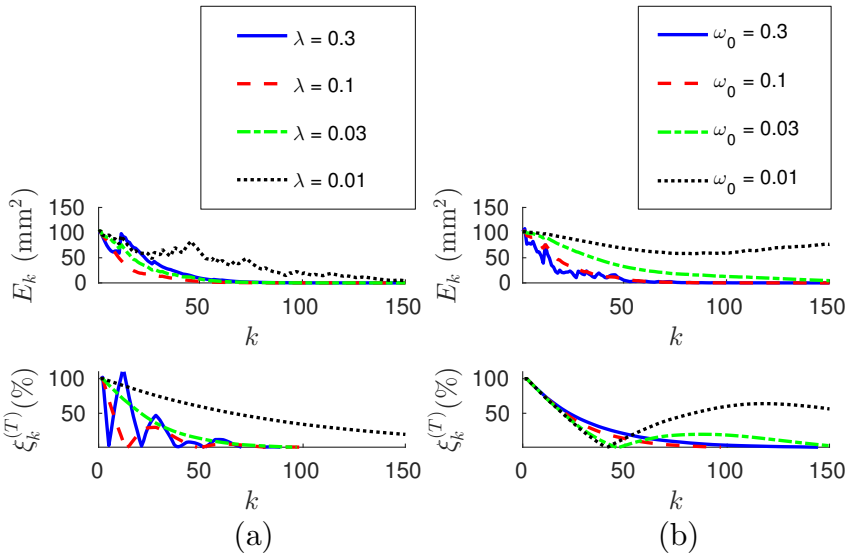



Figure 5.7: (a) Evolution of $E_k$ and $\xi_k^{(T)}$ with the number of iterations for four different values of $\lambda$, with $\omega_0 = 0.1$. The bigger $\lambda$, the faster the convergence. However, too big a value of $\lambda$ will cause the system to oscillate. (b) Evolution of $E_k$ and $\xi_k^{(T)}$ for four different values of $\omega_0$, with $\lambda = 0.03$. Bigger values of $\omega_0$ make the system more stable.

applying the *efficient method* with $\lambda = 0.03$. We observe that the system has a tendency to oscillate for small values of $\omega_0$. This can be explained because small values of $\omega_0$ result in a big "inertia" of the system. Let us remind the reader that we are iteratively updating the desired displacement $\Delta\boldsymbol{T}$ at each step. The parameter $\omega_0$ controls how much we update $\Delta\boldsymbol{T}$ with every incoming event. Consequently, if $\omega_0$ is too small, the desired displacement will continue to be big even when the collinearity errors are already small.

In order to further clarify this point, let us plot in Fig. 5.8 the evolution of the relative translation error $\xi_k^{(T)}$ for different values of $\omega_0$ and $\lambda$. As we can see, for small values of $\omega_0$ the system tends to oscillate even when $\lambda$ is small. This result suggests that one should assign big values to $\omega_0$ (close to one) and to $\lambda$, allowing to achieve fast convergence while the system remains stable. However, if the value of $\omega_0$ is too big we will be assigning a great importance to the most recent events. This is actually equivalent to setting a small

value for $n$, which will cause the system to be less stable in the presence of tracking errors.

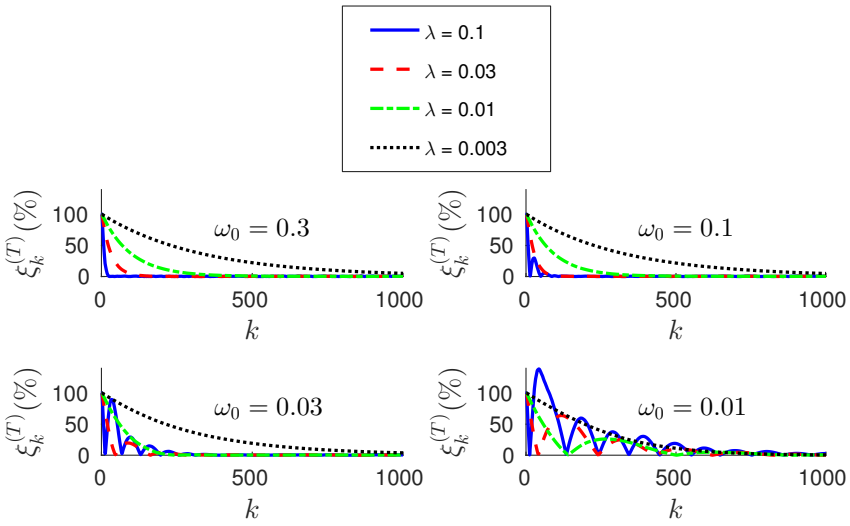



Figure 5.8: Evolution of the relative translation error $\xi_k^{(T)}$ with the incoming events for different values of $\omega_0$ and $\lambda$. We verify that bigger values of $\omega_0$ allow us to set a bigger $\lambda$ without loosing stability. However, the system will become more sensible to tracking errors.

We conclude that when applying the *efficient method* $\omega_0$ and $\lambda$ should be chosen together. We thus recommend values of $\omega_0$ between 0.03 and 0.3, for $\lambda$ between 0.01 and 0.1. In this case, the value of $\omega_0$ has no effect on the computation time required.

### Rotation

We next test how the algorithm estimates only rotation. To that end, we make the initial estimation of the translation $\widehat{\boldsymbol{T}}(0)$ equal to its true value $\boldsymbol{T}$, and set $\lambda = 0$. The initial estimation of the rotation is made $\widehat{R}_0 = I$, the $3 \times 3$ identity matrix.

The maximum distance in the synthetic object is $\rho^{(max)} = 19.95$ mm. Applying (5.26) yields:

$$\phi^{(opt)} = 0.0049\text{N}^{-1}\text{mm}^{-1} \approx 0.005\ \text{N}^{-1}\text{mm}^{-1}. \tag{5.33}$$

We will test different values of $\phi$ around $\phi^{(opt)}$. Fig. 5.9(a) shows the evolution of $E_k$ and $\xi_k^{(R)}$ with the incoming events for four different values of $\phi$, when applying the *full method* with $n = 20$. We verify that the results are very similar to the ones obtained in the case of the translation, with both errors decaying towards zero after $n$ events. Analogously, the convergence is faster for bigger values of $\phi$. We verify that the system still yields stable results for $\phi > \phi^{(opt)}$. In this case, we experimentally determine that for values of $\phi$ greater than 0.01 the rotation fails to converge. We verify that (5.26) provides good theoretical guidance for setting the order of magnitude of $\phi$. We thus recommend to simply set the value of $\phi$ as $\phi^{(opt)}$.

Fig. 5.9(b) shows the evolution of $E_k$ and $\xi_k^{(R)}$ with the incoming events for four different values of $n$, when $\phi = \phi^{(opt)} = 0.005\ \text{N}^{-1}\text{mm}^{-1}$. As in the case of the translation, after $n$ events have elapsed both errors decay towards zero. Again, since the object is static and the tracking is perfect the behavior of the system is very similar for all values of $n$. For inaccurate tracking the same tradeoff applies as in the case of the translation, and we recommend values of $n$ between 20 and 200.

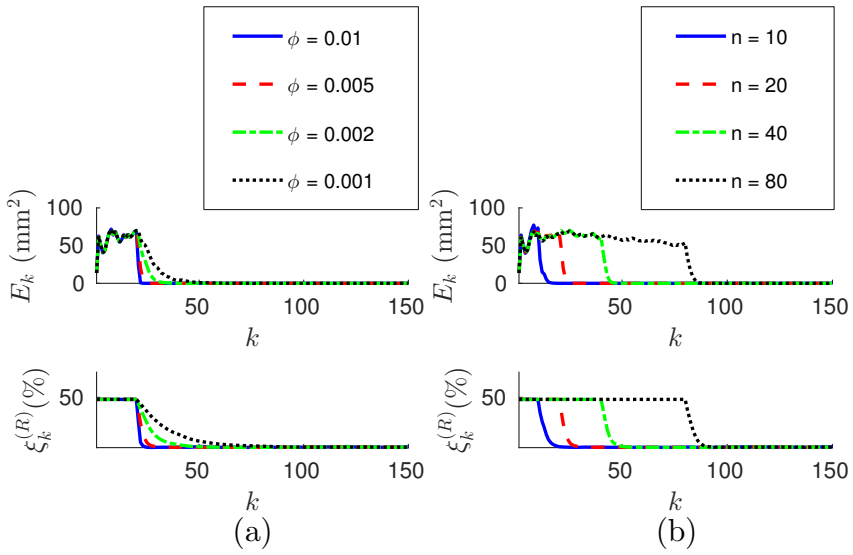



Figure 5.9: (a) Evolution of the errors for four different values of $\phi$ (in N$^{-1}$mm$^{-1}$), when $n = 20$. After $n$ events both errors decay towards zero, the convergence being faster for bigger values of $\phi$. (b) Evolution of the errors for four different values of $n$, with $\phi = 0.005$ (in N$^{-1}$mm$^{-1}$). After $n$ events have elapsed, the behavior of the system is very similar for all values of $n$.

Finally, let us apply the *efficient method* to estimate the rotation. Fig. 5.10(a) shows the evolution of both $E_k$ and $\xi_k^{(T)}$ for four different values of $\phi$, when $\omega_0 = 0.1$. We verify that when applying the *efficient method* the system has a bigger tendency to oscillate.

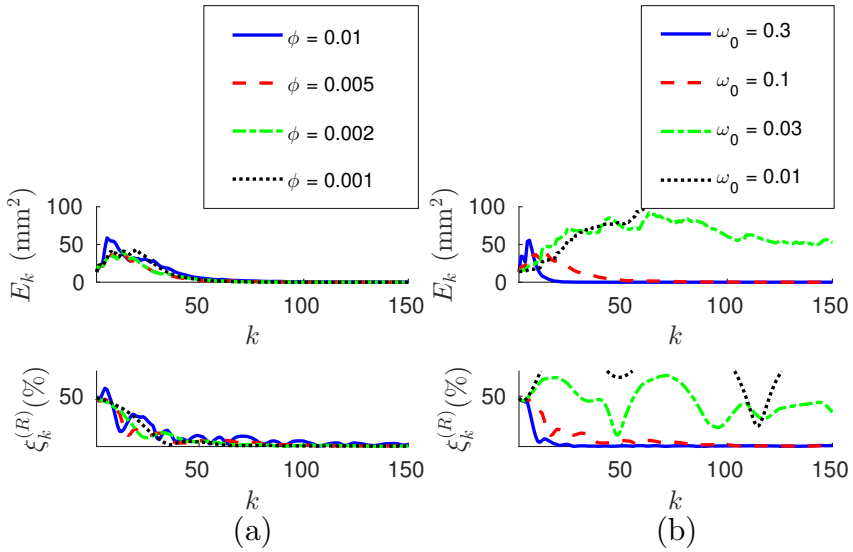



Figure 5.10: (a) Evolution of $E_k$ and $\xi_k^{(R)}$ for four different values of $\phi$ , when $\omega_0 = 0.1$. For big values of $\phi$ the system has a tendency to oscillate, and it might even fail to converge (b) Evolution of the errors when $\phi = 0.001$ N$^{-1}$mm$^{-1}$, for four different values of $\omega_0$. When $\omega_0$ is small, the system has a bigger tendency to oscillate.

Fig. 5.10(b) shows the evolution of the errors for four different values of $\omega_0$, when $\phi = 0.002$ N$^{-1}$mm$^{-1}$. As in the case of the translation, we verify that small values of $\omega_0$ cause the system to oscillate. Analogously, too big a value of $\omega_0$ will cause the system to

be sensible to tracking errors. Consequently, we recommend the same fork of values for $\omega_0$ between 0.03 and 0.3.

### 5.3.2 Real recordings

Next, our algorithm is tested on real data obtained from an ATIS sensor, where an object moves and rotates in front the camera. As an object, we use a white piece of paper in which we printed some logo and a set of black dots (see Fig. 5.11(a)). These dots constitute the model of the object.

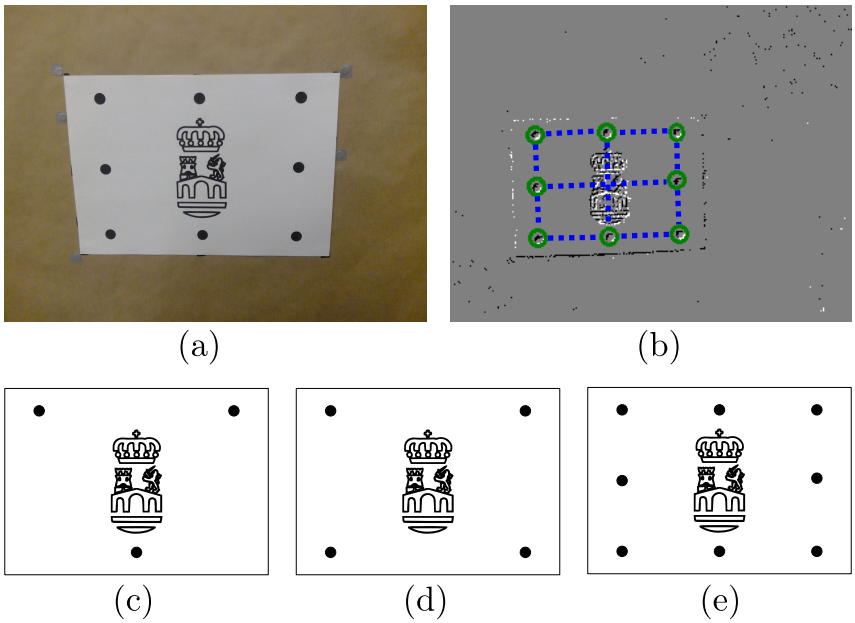



Figure 5.11: (a) Real object used in the experiments: black dots constitute the model of the object, while the logo in the center is used just for visual verification. (b) Output of the *Spring-Linked Tracker Set*: circles show the position of the simple trackers, while the dashed lines represent the springs linking them. (c) 3-point object (d) 4-point object (e) 8-point object.

In order to determine which point of the object has generated an event, we track these dots using the *Spring-Linked Tracker Set* introduced in Chapter 2.

Fig. 5.11(b) shows the output of the *Spring-Linked Tracker Set* when it is applied to one of our recordings. In the image, circles represent the position of each one of the simple trackers, while dashed lines depict the springs linking them. As we can see, we associate a simple tracker with each one of the black dots. Thus, when an event is assigned to one of these trackers, we consider that the event has been generated by the corresponding point of the object. In order to increase the accuracy of the method, we make the location of the event equal to the current position of the tracker, and then feed the resulting stream of clustered events to our P$n$P algorithm.

Ground truth values for numerical evaluation are obtained from an OptiTrack system. OptiTrack is a motion capture system that outputs reliable values for the 3D pose of rigid bodies, provided that they are equipped with a number of infrared markers. Fixing markers on the object and the camera allows us to obtain their poses in the 3D space, from

http://www.optitrack.com/

which we retrieve the pose of the object relative to the camera. Comparing this value with the estimation of our algorithm we compute the relative translation and rotation errors. Accuracy is characterized by the mean value of these errors computed for a whole recording.

We consider three different objects: the ones composed by three, four or eight points (see Figures 5.11(c), 5.11(d) and 5.11(e) respectively). This will allow us to evaluate the effect of the number of points on the accuracy of the algorithm. For all three objects the maximum distance is $\rho^{(max)} = 136.01$ mm. Applying (5.26) yields $\phi^{(opt)} \approx 0.0001$ $\text{N}^{-1}\text{mm}^{-1}$.

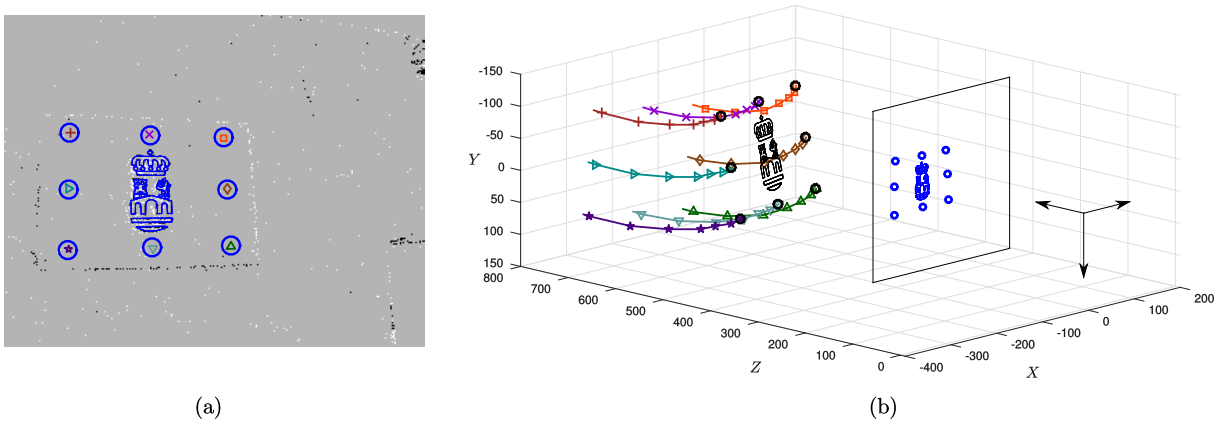
We make three different recordings for each one of the objects, producing a total of nine recordings. We identify them by their index, going from one to nine, where recordings #01 to #03 correspond to scenes containing the 3-point object. Recordings #04 to #06 contain the 4-point object, and #07 to #09 the 8-point object. In all of them the corresponding object is displaced and rotated in every direction. All recordings are cropped to have the same duration of 25 s and solved using the same set of parameters. The initial estimation of the pose is always made $\widehat{\boldsymbol{T}} = (0, 0, 0)^T$, $\widehat{R}(0) = I$.

We test the accuracy of both the *full method* and the *efficient method* on these real data. Additionally, we implement Lu's method [114] and apply it to our recordings as well. This allows us to evaluate our approach against a state of the art P$n$P algorithm.

Let us first apply the *full method* to the 9 recordings. Parameters are selected in the range giving stable results in the previous experiment. After several trials, they are experimentally set to $n = 50$, $\lambda = 0.1$ and $\phi = 0.0001$ $\text{N}^{-1}\text{mm}^{-1}$. Fig. 5.12(a) depicts the characteristic output of the P$n$P algorithm at a given instant, corresponding to recording #09. Here, the background of the image shows a snapshot of the ATIS output. Additionally, events assigned to different points of the object are indicated by different shapes (crosses, triangles and so on), while circles represent the reprojection of the object at the pose estimated by the algorithm. We can see that, in general, circles surround the events generated by the corresponding points, showing that our method is yielding good results on the focal plane. We also reproject the logo, that as we can see matches the corresponding events. Fig. 5.12(b) shows the state of the system at the same instant represented in the 3D space, where the camera's optical center has been placed at the origin. Recent trajectories of the points of the object have been plotted too, represented with the same set of symbols as in the 2D image. We show in Video 1 the output of the algorithm for this recordings: to the left we show results obtained on the focal plane, while 3D results are shown on the right side.

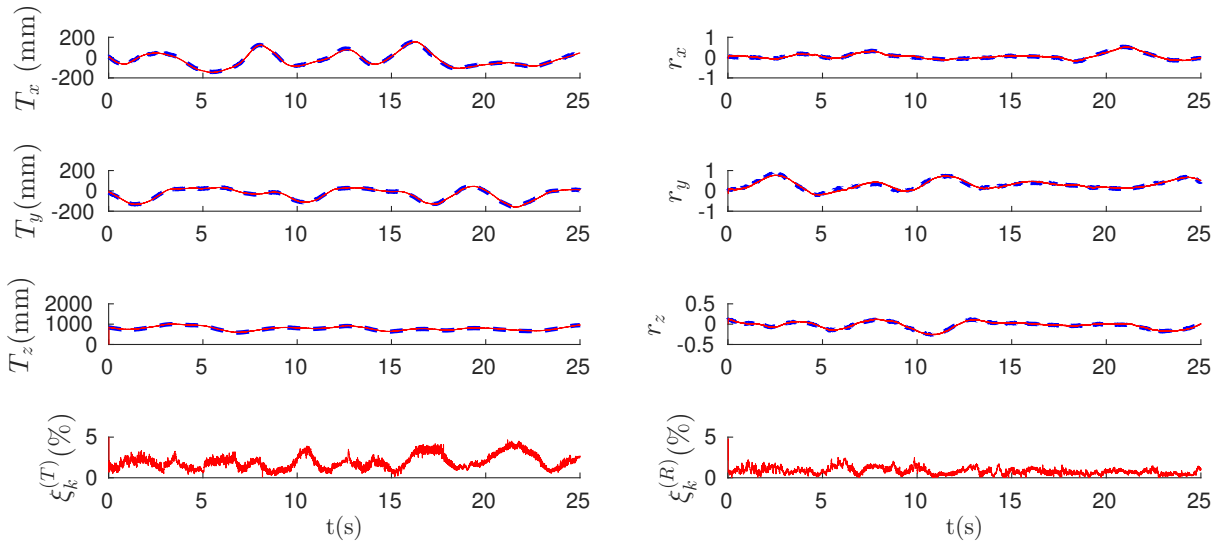
In order to illustrate pose estimation results produced by the algorithm, let us plot in Fig. 5.13(a) the evolution of the three components of the translation vector $\boldsymbol{T}$ (in mm) for recording #09. In the figure, ground truth values are represented by dashed lines, and estimated values by continuous lines. We verify that these curves are coincidental, showing that the algorithm is correctly estimating translation. The relative translation error $\xi_k^{(T)}$ is shown at the bottom of the figure: as we can see, after a short initial transient its value stabilizes to be always lower than 5%. This results in a mean value for the whole recording of just 1.79%. We denote this mean error $\overline{\xi^{(T)}}$, and use it to characterize the accuracy of a given approach.

Analogously, Fig. 5.13(b) shows the three components of the rotation vector $\boldsymbol{r}$. As in the case of the translation, estimated values are coincidental with the ground truth references provided by the OptiTrack system. Consequently, the relative rotation error $\xi_k^{(R)}$ is always below 5%, resulting in a mean value (denoted $\overline{\xi^{(R)}}$) of only 0.79%. These




(a) (b)

Figure 5.12: (a) Characteristic output of the P$n$P algorithm, corresponding to recording #09. The background of the image shows a snapshot of the ATIS recording, while events assigned to different points of the object are represented by different shapes (crosses, triangles, etc.). Circles indicate the reprojection of the object at its estimated pose: as we can see, the reprojection matches the corresponding events, showing that the algorithm is yielding good results on the focal plane. We reproject the logo as well, that matches the corresponding events. (b) 3D representation of the same instant, where the recent trajectories of the points of the object have been plotted using the same set of symbols.




(a) (b)

Figure 5.13: (a) Evolution of the three components of the translation vector $\boldsymbol{T}$ (in mm) for recording #09. Ground truth values are indicated by dashed lines, while the results produced by the *full method* are represented by solid lines. We verify that these two curves are coincidental. The resulting value of the relative translation error $\xi_k^{(T)}$ is shown at the bottom: after a short initial transient its value remains always below 5%, yielding a mean value for the whole recording of just 1.79%. (b) Evolution of the rotation vector $\boldsymbol{r}$: estimated values are coincidental with ground truth values. This results in a mean value for the relative rotation error of 0.79%.

results allow us to conclude that the *full method* is correctly estimating the pose of the object for this recording.

Let us next apply the *efficient method* to all the recordings with the following set of parameters: $\omega_0 = 0.1$, $\lambda = 0.1$ and $\phi = 0.0001$ N$^{-1}$mm$^{-1}$ (note that $\lambda$ and $\phi$ take the same values as for the *full method*). Considering Lu's algorithm, the only parameter is $n$, that we experimentally set to $n = 50$. We show in Fig. 5.14 the mean errors obtained for every recording with all three methods, where the relative translation error is shown on top and the relative rotation error at the bottom. Recordings of the same object are grouped together. From the results displayed in Fig. 5.14 we can extract the following conclusions:

- The *full method* and the *efficient method* yield statistically equivalent results: for every recording, results obtained with both methods are almost identical. This proves the *efficient method* to be a valuable approximation.
- We cannot uniquely estimate the pose of an object with less than four points: when the 3-point object is considered, every method fails to produce accurate pose estimations. This is a known limitation of the P$n$P technique [124], not specific to the event-based approach.
- When four or eight points are considered, pose is correctly estimated by our algorithm. Results obtained by our method are as reliable as the ones provided by Lu's, showing the accuracy of our approach. Both the *efficient method* and the *full method* produce errors in the same range of values, from 1.9% to 2.8% for the translation, and from 0.8% to 1.2% in the case of the rotation.
- Increasing the number of points to more than four does not improve the accuracy of the algorithm in the experiments. More than four points produce an overdetermined system, which is not always a guarantee of a better accuracy in the pose estimation. We hypothesize that, in this particular case, we have reached the limit of the algorithm because of the sensor's spatial resolution. A more thorough study with different stimuli, experimental conditions and tracking techniques is necessary to reach a firm conclusion.

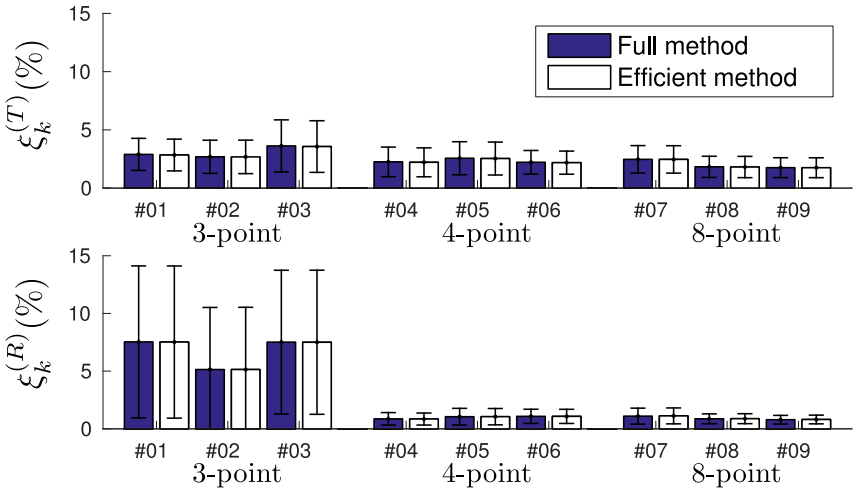



Figure 5.14: Statistics of the errors obtained for each one of the nine recordings. The *full method* and the *efficient method* yield equivalent results. When four or eight points are considered, the accuracy of our algorithm is equivalent to Lu's.

In summary, when the object is composed by at least four points all three methods provide comparable accuracy. In order to establish a complete comparison between them we next evaluate their computation time.

### 5.3.3 Computational time

The presented experiments were carried out using a conventional laptop equipped with an Intel Core i7 processor and running Debian Linux. The algorithm was implemented both in Matlab and C++. Only the computational time of the C++ implementation is discussed. As in the previous chapters, the code is not parallelized and a single core was used. We discuss here the total time it takes to process each recording (let us remind the reader that all recordings have a same length of 25 s), where each recording was processed ten times and the average result is considered.

When applying the *full method* or Lu's method, the computational time depends on the value of $n$ (the number of past events taken into account for updating the pose). We thus evaluate the evolution of both the computational time and the pose errors with the value of this parameter. We show in Fig. 5.15 the results obtained for recordings #01, #04 and #07, where $n$ takes values between 2 and 30. Only three recordings are shown for clarity reasons, results obtained for the remaining recordings are equivalent.

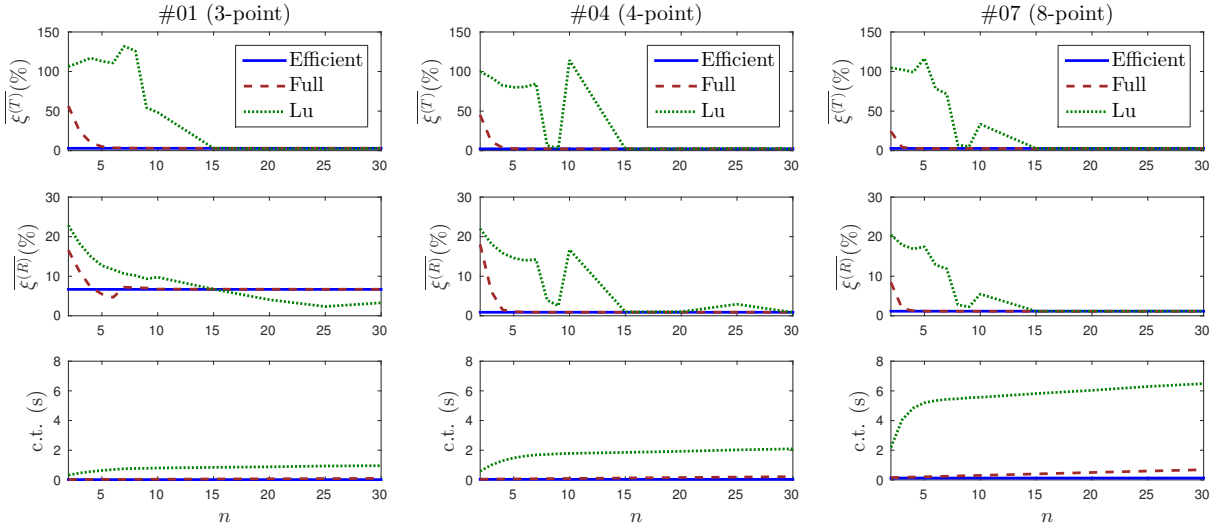



Figure 5.15: Evolution of the pose estimation errors and the computational time with the value of $n$. (a) Results for recording #01. Top: mean translation error. Lu's method is unstable for small values of $n$, and its behavior is not predictable until $n = 15$. Middle: mean rotation error. Since this recording contains the 3-point object, rotation cannot be correctly estimated and rotation errors take big values for all three methods. Bottom: computational time. The *efficient method* is the fastest one, and it does not depend on $n$. (b) Results for recording #02. Rotation can be correctly estimated in this case. The computational time is greater than for recording #01, because more points in the object implies more events to treat. (c) Results for recording #07.

Fig. 5.15(a) shows the results obtained for recording #01, which contains the 3-point object. On top, the evolution of the mean translation error $\overline{\xi^{(T)}}$ with the value of $n$ is shown. If we analyze the *efficient method*, we observe that a straight line is obtained. This is an expected result, since this method does not depend on $n$. The value of the error is $\overline{\xi^{(T)}} = 2.95\%$.

In the case of the *full method*, we observe that large errors are obtained for small values of $n$. This is also an expected result, as explained in Section 5.3.1. From $n = 6$ the error stabilizes at around 3%.

Finally, when applying Lu's method, we verify that the errors are even higher for small values of $n$, and they do not stabilize until $n = 15$. This can be explained because Lu's method is not incremental. Instead, it computes at each iteration the best solution for the last $n$ events. Due to the nature of the neuromorphic camera, it often occurs that several consecutive events are generated by the same point of the object. Consequently, when Lu's method is applied with a small value of $n$, pose is often estimated using just one or two points of the object. This causes the estimation to be flawed, leading to large oscillations which result in the large observed errors. This is an indication that Lu's algorithm is not well-suited to the output of the neuromorphic camera.

The middle row of Fig. 5.15(a) shows the mean rotation errors for recording #01. Since the 3-point object is considered in this case, rotation cannot be uniquely estimated. This results in large rotation errors for all three methods.

At the bottom of Fig. 5.15(a) we show the computational time required for applying the P$n$P algorithm to recording #01. Values are averaged over 10 trials for each set of parameters. For the *efficient method* we observe a straight line at the value 0.017 s. In the case of the *full method*, we verify that the computational time grows linearly with $n$. If we analyze Lu's method, we verify that it also grows with $n$, linearly from $n = 8$.

We consider the computational time at $n = 30$ as a reference, since this value ensures stable solutions for all recordings with every method. The *efficient method* is then 6.1 times faster than the *full method* and 57.5 times faster than Lu's.

Fig. 5.15(b) shows the results obtained for recording #04, that contains the 4-point object. Rotations can be correctly estimated in this case, the *efficient method* yields a mean rotation error $\overline{\xi^{(R)}} = 0.89\%$. The *full method* and Lu's method stabilize around this value from $n = 6$ and $n = 30$ respectively. As observed before, small values of $n$ cause the solution to oscillate, specially in the case of Lu's method.

At the bottom of Fig. 5.15(b) the computational time required to solve recording #04 is shown. We verify that it follows the same previously observed pattern. However, the computational time is larger for every method, because considering more points implies more events to process. The computational time for the *efficient method* is 0.035 s. This is 6.0 times faster than the *full method* and 58.4 times faster than Lu's method (with $n = 30$).

Figure 5.15(c) shows the results for recording #07, that contains the 8-point object. Pose can be correctly estimated from $n = 4$ when applying the *full method*, and from $n = 15$ when Lu's method is chosen. When $n = 30$, the *efficient method* is 5.5 times faster than the *full method* and 52.4 times faster than Lu's.

It is important to emphasize that these results are implementation-dependent. Several optimization techniques can be applied, providing faster computational times. However, our implementation shows the *efficient method* to be around 50 times faster than Lu's method.

## 5.4 Discussion

This chapter introduces an event-based solution to the P$n$P problem. When computing the optimal translation of the object, we adapt a preexisting closed-form solution to our incremental approach. Rotation, however, cannot be solved in such a simple manner.

Previous solutions employed complicated techniques to compute the optimal rotation for each frame, applying SVD to the solution of complex systems of equations. In our work, the rotation is estimated by simulating the evolution of a virtual mechanical system instead. This results in a simple yet robust algorithm, capable of accurately estimating the pose of the tracked object with microsecond precision. As an additional advantage, we consider our algorithm to be more intuitive and easier to implement than Lu's one. As a drawback, it requires the tuning of 3 parameters, while Lu's method has only one. However, we prove that there is a big range of values for which our method provides stable and accurate results.

When applying the *efficient method*, the resulting equations are very simple. Nevertheless, we prove that this approximation yields equivalent results to the *full method* when dealing with real recordings of moving objects. For the chosen set of parameters, when the object is composed of at least 4 points both the *efficient method* and the *full method* produce errors in the same range of values: from 1.9% to 2.8% for the translation (relative to the norm of the mean translation for the considered recording), and between 0.8% and 1.2% for the rotation (relative to the maximum possible distance between two rotation matrices). These values, very similar to the ones produced by Lu's algorithm, are sufficiently low to conclude that our method can correctly estimate 3D pose at a lower cost.

When the computational time of the different approaches is analyzed, we show that the *efficient method* is faster than the *full method*, and much faster than Lu's algorithm [114], while being equally accurate. For our precise implementation, the *efficient method* is around 5 times faster than the *full method* and 50 times faster than Lu's algorithm. Even if we are aware that these results are implementation-dependent, we consider the difference to be significant enough to conclude that the *efficient method* is faster, and thus recommend it as the standard choice. We claim that this gain in efficiency comes from the fact that our method is specifically adapted to handle the output of neuromorphic cameras.

As every P$n$P technique, our method requires matching 3D points with their 2D projections on the focal plane. The matching accuracy has consequently a strong impact on the overall performance of the system and the event-based P$n$P algorithm will benefit from any advances in event-based tracking or marker detection.

# Chapter 6

# Conclusion and Future Perspectives

## 6.1 Conclusion

This PhD thesis introduces a series of novel techniques for visual detection and tracking, specifically designed to operate on the asynchronous output of neuromorphic event-based cameras. Its main contribution is the development of purely event-based algorithms, which have been applied to a number of different tasks. Chapter 2 introduces a 2D tracking technique that allows the tracking of complex objects, while a line and segment detection algorithm is presented in Chapter 3. The 3D pose estimation algorithm introduced in Chapter 4 and the solution to the P$n$P problem presented in Chapter 5 are the first purely event-based algorithms available in the field.

All the techniques presented in this thesis are truly event-driven, as the position (2D or 3D) of the tracked object is updated with every incoming event. Due to the techniques' event-based processing fully exploiting the neuromorphic cameras' quasi-continuous acquisition of visual information, the proposed methods are particularly suitable for high-speed applications. As a canonical example, in Chapter 4 the pose of an object is accurately estimated, though the object is spinning at angular speeds of up to 26.4 revolutions per second. To achieve equivalent accuracy with a frame-based camera, high frame rates would be required, and consequently the number of frames to process would increase. Compared to conventional frame-based acquisition, the results presented in this thesis show that neuromorphic event-driven high temporal resolution cameras allow the tracking problem to be tackled by introducing true dynamics into the system.

All the presented methods have been implemented in C++ and run on a standard computer. The computational requirements of each method are studied at the end of the corresponding experimental section, showing that the computational time grows linearly with the event rate (i.e. the time it takes to process one event remains, on average, approximately constant). This allows to extrapolate and compute the maximum event rate that can be processed in real time by the algorithms. Of course, computational requirements measured in this way are dependent on the precise implementation of the algorithm, as well as the computational power of the considered machine. However, I consider that this measure provides a good idea of the computational load of a given algorithm, and it is particularly useful for means of comparison. The current implementation of the introduced algorithms is thus proven to be capable of processing event streams in real time using a conventional computer, up to some event rate. This maximum event rate is different for each of the studied methods, but its value is, in all cases, high (relative to the typical event rates produced by the ATIS camera). These characteristics make these

algorithms suitable for real-time applications.

According to the results provided in this thesis, I think it is fair to claim that neuromorphic event-based computer vision makes it possible to reformulate a broad set of computer vision tasks. The high temporal resolution of events usually allows to apply simplifying assumptions, which result in simpler algorithms requiring a smaller number of operations per event, without degrading the obtained results. Very often, one does not need to find an optimal solution to the problem for each incoming event. Instead, it is enough to take a small step in the right direction. Consider, for example, the solution to the P$n$P problem presented in Chapter 5: the optimal rotation is not guaranteed for each event. However, we know that the system is evolving towards the minimum of the error function. Since events are numerous in the parts of the scene containing rich dynamics (where the "important things" are happening), this approach quickly leads to an accurate estimate of the pose. Compared to frame-based methods, I believe this approach to be conceptually simpler: instead of redundantly processing all pixels, as it is usually done in the frame based approach, the event-based philosophy is to minimize the computational resources applied to each event. I thus consider that neuromorphic vision often results in simpler and more intuitive solutions to computer vision problems.

Taking into account the achieved performances of the techniques developed in this thesis, I consider the contribution of my work to be somehow significant to the field of event-based computer vision.

## 6.2 Future Perspectives

One of the main motivations for this work has been its application to visual servoing. The techniques presented in this thesis constitute some of the founding blocks for this challenging undertaking. However, the complete realization of the task remains unfulfilled and it is the most natural continuation for this research. The final goal would be to build a complete event-based perception-action loop, fast and efficient enough to match the performance of biological sensory-motor systems. This implies a number of challenges that should be tackled. In the first place, most control techniques (from the basic PID controller to the most advanced algorithms) are designed to operate on periodic sampling. Event-based control is an emerging and challenging field, that should be explored in order to completely exploit the potential of event-based cameras when applied to visual servoing tasks.

Additionally, if the camera is mounted on a robot (which is usually known as the *eye-in-hand* configuration in the visual servoing literature), its motion tends to generate an important number of events, due to the relative motion of the visual scene with respect to the neuromorphic camera. As shown in Chapter 4, the tracking problem can still be solved by applying more refined matching criteria. However, this usually implies an increase in computational requirements. Thus, the movement of the camera typically results in an increased event rate and a greater computational load per event. Consequently, one needs to be especially careful if real-time performance needs to be guaranteed (which is always the case for visual servoing tasks).

Another challenge (common to most tracking algorithms) is the problem of initialization. If an algorithm requires a manual initialization step, its application to real-time tasks is hampered. The part-based shape tracker described in Chapter 2 and the 3D pose estimation algorithm from Chapter 4 both suffer from this drawback. The solution to the P$n$P problem presented in Chapter 5 was in part motivated by this problem, and it offers

a partial solution, as the 3D pose of the tracked object does not need to be manually initialized. Nevertheless, this issue has not been solved for our part-based shape tracker. Further research is required for more specialized event-based object (or marker) detectors capable of automatic initialization.

Finally, the four techniques developed in this thesis share a common philosophy, as well as a number of tools. Future research also includes the application of this same approach to other computer vision problems. Particularly, the problem of visual SLAM (Simultaneous Localization and Mapping) has not yet been solved in a purely event-driven fashion, and it is currently the subject of research for numerous scientists. This problem is related to the 3D Pose Estimation problem explored in Chapters 4 and 5, and it thus constitutes a natural extension to this work.

Moreover, some of these techniques can be easily adapted to operate on the output of other event-based sensors. Although I have only worked with the visual events produced by an event-based vision sensor, the underlying philosophy and some of the introduced tools can be applied to generic event-based signals produced by any event-based sensing device. Consider, for example, the event-based line detection algorithm presented in Chapter 3: this method can be used for event-based signal smoothing with very little modifications, locally approximating some generic event-based signal by line segments. This approach is currently being explored, along with the extension of the method to higher order polynomials (preliminary tests show promising results for visual tracking and velocity estimation).

I thus consider that this thesis takes some small first steps in the direction of establishing some fundamental low level processing tools for event-based signals, that can be further developed and explored. In my opinion, this is one of the most enticing and fascinating lines of research as a continuation to this work: abstracting the common characteristics to the different event streams produced by neuromorphic event-based sensors, and exploring these tools with the aim of advancing in the field of event-based signal processing.

# Bibliography


[1] G. Indiveri and T. K. Horiuchi, "Frontiers in neuromorphic engineering," *Frontiers in Neuroscience*, vol. 5, p. 118, 2011.

[2] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[3] ——, "Analog VLSI and neural systems," *Reading: Addison-Wesley, 1989*, vol. 1, 1989.

[4] R. P. Feynman, J. Hey, and R. W. Allen, *Feynman lectures on computation*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[5] B. Sengupta and M. B. Stemmler, "Power consumption during neuronal computation," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 738–750, 2014.

[6] C. Bartolozzi, R. Benosman, K. Boahen, G. Cauwenberghs, T. Delbrück, G. Indiveri, S.-C. Liu, S. Furber, N. Imam, B. Linares-Barranco, T. Serrano-Gotarredona, K. Meier, C. Posch, and M. Valle, *Neuromorphic Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016. [Online]. Available: http://dx.doi.org/10.1002/047134608X.W8328

[7] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[8] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[9] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[10] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2010, pp. 1947–1950.

[11] S. Scholze, H. Eisenreich, S. Höppner, G. Ellguth, S. Henker, M. Ander, S. Hänzsche, J. Partzsch, C. Mayr, and R. Schüffny, "A 32GBit/s communication SoC for a waferscale neuromorphic system," *INTEGRATION, the VLSI journal*, vol. 45, no. 1, pp. 61–75, 2012.

[12] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, 2016.

[13] S. Caviglia, M. Valle, and C. Bartolozzi, "Asynchronous, event-driven readout of POSFET devices for tactile sensing," in *Proceedings of 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2014, pp. 2648–2651.

[14] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "A kilohertz kilotaxel tactile sensor array for investigating spatiotemporal features in neuromorphic touch," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2015, pp. 1–4.

[15] T. J. Koickal, A. Hamilton, S. L. Tan, J. A. Covington, J. W. Gardner, and T. C. Pearce, "Analog VLSI circuit implementation of an adaptive neuromorphic olfaction chip," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 60–73, 2007.

[16] M. Mahowald and R. Douglas, "A silicon neuron," *Nature*, vol. 354, no. 6354, p. 515, 1991.

[17] M. Mahowald, "VLSI analogs of neuronal visual processing: a synthesis of form and function," Ph.D. dissertation, California Institute of Technology, 1992.

[18] E. Culurciello, R. Etienne-Cummings, and K. A. Boahen, "A biomorphic digital image sensor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 281–294, 2003.

[19] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128×128 120 dB 15 $\mu$s latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[20] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2008, pp. 2130–2133.

[21] ——, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011.

[22] R. F. Lyon and C. Mead, "An analog electronic cochlea," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 7, pp. 1119–1134, 1988.

[23] E. Fragnière, "A 100-channel analog CMOS auditory filter bank for speech recognition," in *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference.* IEEE, 2005, pp. 140–589.

[24] V. Chan, S.-C. Liu, and A. van Schaik, "AER EAR: A matched silicon cochlea pair with address event representation interface," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48–59, 2007.

[25] T. Delbrück, "Silicon retina with correlation-based, velocity-tuned pixels," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 529–541, 1993.

[26] R. Etienne-Cummings, J. Van der Spiegel, and P. Mueller, "A focal plane visual motion measurement sensor," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 44, no. 1, pp. 55–66, 1997.

[27] J. Krammer and C. Koch, "Pulse-based analog VLSI velocity sensors," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 2, pp. 86–101, 1997.

[28] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[29] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbrück, "Retinomorphic event-based vision sensors: bioinspired cameras with spiking output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014.

[30] R. Benosman, S.-H. Ieng, P. Rogister, and C. Posch, "Asynchronous event-based hebbian epipolar geometry," *IEEE Transactions on Neural Networks*, vol. 22, no. 11, pp. 1723–1734, 2011.

[31] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, pp. 32–37, 2012.

[32] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, 2014.

[33] S. Agarwal, A. Awan, and D. Roth, "Learning to detect objects in images via a sparse, part-based representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1475–1490, 2004.

[34] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Transactions. on Computers*, vol. 100, pp. 67–92, Jan. 1973.

[35] Y. Lu and D. Song, "Robust RGB-D odometry using point and line features," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3934–3942.

[36] G. Schindler, P. Krishnamurthy, and F. Dellaert, "Line-based structure from motion for urban environments," in *Third International Symposium on 3D Data Processing, Visualization, and Transmission.* IEEE, 2006, pp. 846–853.

[37] T. Lemaire and S. Lacroix, "Monocular-vision based SLAM using line segments," in *Proceedings of 2007 IEEE International Conference on Robotics and Automation.* IEEE, 2007, pp. 2791–2796.

[38] F. Chaumette and S. Hutchinson, "Visual servo control. I. basic approaches [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[39] ——, "Visual servo control. II. advanced approaches [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 109–118, 2007.

[40] V. Lepetit and P. Fua, "Monocular model-based 3D tracking of rigid objects: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 1, no. 1, pp. 1–89, 2005.

[41] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[42] M. Litzenberger, B. Kohn, A. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml, "Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor," in *2006 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2006, pp. 653–658.

[43] M. Litzenberger, C. Posch, D. Bauer, A. Belbachir, P. Schon, B. Kohn, and H. Garn, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *2006 IEEE 12th Digital Signal Processing Workshop 4th IEEE Signal Processing Education Workshop*. IEEE, 2006, pp. 173–178.

[44] G. L. Foresti, "Object recognition and tracking for remote video surveillance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 7, pp. 1045–1062, 1999.

[45] I. Cohen and G. Medioni, "Detecting and tracking moving objects for video surveillance," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, Ft. Collins, CO, USA, Jun. 1999, p. 325.

[46] R. J. Jacob and K. S. Karn, "Eye tracking in human-computer interaction and usability research: Ready to deliver the promises," *Mind*, vol. 2, no. 3, p. 4, 2003.

[47] U. Neumann and S. You, "Natural feature tracking for augmented reality," *IEEE Transactions on Multimedia*, vol. 1, pp. 53–64, Mar. 1999.

[48] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik, "A real-time computer vision system for vehicle tracking and traffic surveillance," *Transportation Research Part C: Emerging Technologies*, vol. 6, no. 4, pp. 271–288, 1998.

[49] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 34–58, Jan. 2002.

[50] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, Kauai, HI, USA, Dec. 2001, pp. 511–518.

[51] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proceedings of the IEEE International Conference on Image Processing*, Rochester, NY, USA, Sep. 2002, pp. 900–903.

[52] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[53] B. Wu, H. Ai, C. Huang, and S. Lao, "Fast rotation invariant multi-view face detection based on real Adaboost," in *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, Seoul, Korea, May 2004, pp. 79–84.

[54] T. Mita, T. Kaneko, and O. Hori, "Joint Haar-like features for face detection," in *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05)*, vol. 2. IEEE, 2005, pp. 1619–1626.

[55] M. Jones and P. Viola, "Fast multi-view face detection," Mitsubishi Electric Research Lab, Cambridge, MA, USA, Tech. Rep. TR-20003-96, 2003.

[56] B. Froba and A. Ernst, "Face detection with the modified census transform," in *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition.*, Seoul, Korea, May 2004, pp. 91–96.

[57] P. Menezes, J. Barreto, and J. Dias, "Face tracking based on Haar-like features and eigenfaces," in *Proceedings of the IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, Lisbon, Portugal, Jul. 2004.

[58] S. Zafeiriou, C. Zhang, and Z. Zhang, "A survey on face detection in the wild: past, present and future," *Computer Vision and Image Understanding*, vol. 138, pp. 1–24, 2015.

[59] X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, "HOTS: A Hierarchy Of event-based Time-Surfaces for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[60] M. C. Burl, M. Weber, and P. Perona, "A probabilistic approach to object recognition using local photometry and global geometry," in *European Conference on Computer Vision*, Freiburg, Germany, 1998, pp. 628–641.

[61] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, Madison, WI, USA, Jun. 2003, pp. 264–271.

[62] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

[63] M. Andriluka, S. Roth, and B. Schiele, "Pictorial structures revisited: People detection and articulated pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, Miami, FL, USA, Jun. 2009, pp. 1014–1021.

[64] Z. Ni, C. Pacoret, R. Benosman, S. Ieng *et al.*, "Asynchronous event-based high speed vision for microparticle tracking," *Journal of Microscopy*, vol. 245, no. 3, pp. 236–244, 2012.

[65] D. Drazen, P. Lichtsteiner, P. Häfliger, T. Delbrück, and A. Jensen, "Toward real-time particle tracking using an event-based dynamic vision sensor," *Experiments in Fluids*, vol. 51, no. 5, p. 1465, 2011.

[66] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Régnier, "Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1081–1089, 2012.

[67] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, "Asynchronous event-based multikernel algorithm for high-speed visual features tracking," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 8, pp. 1710–1720, 2015.

[68] W. Mendenhall, R. J. Beaver, and B. M. Beaver, *Introduction to probability and statistics*. Cengage Learning, 2012.

[69] J. Taylor, *Classical Mechanics*. Sausalito, CA, USA: University Science Books, 2005.

[70] V. I. Arnold, *Mathematical methods of classical mechanics*. Berlin, Germany: Springer Science & Business Media, 1989, vol. 60.

[71] C. S. Chane, S.-H. Ieng, C. Posch, and R. B. Benosman, "Event-based tone mapping for asynchronous time-based image sensor," *Frontiers in Neuroscience*, vol. 10, 2016.

[72] L. Zhang and R. Koch, "Structure and motion from line correspondences: representation, projection, initialization and sparse bundle adjustment," *Journal of Visual Communication and Image Representation*, vol. 25, no. 5, pp. 904–915, 2014.

[73] W. Y. Jeong and K. M. Lee, "Visual SLAM with line and corner features," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2570–2575.

[74] J. Lv, Y. Kobayashi, A. A. Ravankar, and T. Emaru, "Straight line segments extraction and EKF-SLAM in indoor environment," *Journal of Automation and Control Engineering*, vol. 2, no. 3, 2014.

[75] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.

[76] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[77] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of Hough transform," *Pattern Recognition*, vol. 48, no. 3, pp. 993–1010, 2015.

[78] J. B. Burns, A. R. Hanson, and E. M. Riseman, "Extracting straight lines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 4, pp. 425–455, 1986.

[79] A.-R. Mansouri, A. S. Malowany, and M. D. Levine, "Line detection in digital pictures: A hypothesis prediction/verification paradigm," *Computer Vision, Graphics, and Image Processing*, vol. 40, no. 1, pp. 95–114, 1987.

[80] v. G. R. Grompone, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: a fast line segment detector with a false detection control." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 4, pp. 722–732, 2010.

[81] X. Clady, S.-H. Ieng, and R. Benosman, "Asynchronous event-based corner detection and matching," *Neural Networks*, vol. 66, pp. 91–106, 2015.

[82] V. Vasco, A. Glover, and C. Bartolozzi, "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)*. IEEE, 2016, pp. 4144–4149.

[83] X. Clady, J.-M. Maro, S. Barré, and R. B. Benosman, "A motion-based feature for event-based pattern recognition," *Frontiers in Neuroscience*, vol. 10, p. 594, 2016.

[84] S. Seifozzakerini, W.-Y. Yau, B. Zhao, and K. Mao, "Event-based Hough transform in a spiking neural network for multiple line detection and tracking using a dynamic vision sensor," in *Proceedings of the 2016 British Machine Vision Conference (BMVC)*, 2016.

[85] C. Brändli, J. Strubel, S. Keller, D. Scaramuzza, and T. Delbruck, "ELiSeD—An event-based line segment detector," in *IEEE Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2016, pp. 1–7.

[86] D. R. Valeiras, S. Kime, S.-H. Ieng, and R. B. Benosman, "An event-based solution to the perspective-n-point problem," *Frontiers in neuroscience*, vol. 10, 2016.

[87] E. H. Adelson and J. A. Movshon, "Phenomenal coherence of moving visual patterns," *Nature*, vol. 300, no. 5892, pp. 523–525, 1982.

[88] Å. Björck, *Numerical methods in matrix computations*. Berlin, Germany: Springer, 2015.

[89] S. Chatterjee and A. S. Hadi, *Regression analysis by example*. John Wiley & Sons, 2015.

[90] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[91] R. Larson and B. H. Edwards, *Calculus of a single variable*. Cengage Learning, 2013.

[92] R. Brunelli, *Template matching techniques in computer vision: theory and practice*. Hoboken, NJ, USA: John Wiley & Sons, 2009.

[93] G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1858–1865, 2008.

[94] N. Snavely, S. M. Seitz, and R. Szeliski, "Modeling the world from internet photo collections," *International Journal of Computer Vision*, vol. 80, no. 2, pp. 189–210, 2008.

[95] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building Rome in a day," *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[96] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 932–946, Jul. 2002.

[97] D. Van Krevelen and R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International Journal of Virtual Reality*, vol. 9, no. 2, p. 1, 2010.

[98] F. Janabi-Sharifi, "Visual servoing: theory and applications," *Opto-Mechatronic Systems Handbook*, pp. 15–1–15–24, 2002.

[99] F. Janabi-Sharifi and M. Marey, "A kalman-filter-based method for pose estimation in visual servoing," *IEEE Transactions on Robotics*, vol. 26, pp. 939–947, Oct. 2010.

[100] Y. Abbel-Aziz, "Direct linear transformation from comparator coordinates in close-range photogrammetry," in *Proceedings of the American Society of Photogrammetry Symposium on Close-range Photogrammetry.* American Society of Photogrammetry, 1971.

[101] D. F. Dementhon and L. S. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, vol. 15, no. 1, pp. 123–141, 1995.

[102] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99).* IEEE, 1999, pp. 85–94.

[103] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate non-iterative o (n) solution to the pnp problem," in *IEEE 11th International Conference on Computer Vision.* IEEE, 2007, pp. 1–8.

[104] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, "Iterative pose estimation using coplanar feature points," *Computer Vision and Image Understanding*, vol. 63, no. 3, pp. 495–511, 1996.

[105] MIP, CAU Kiel, Germany, "BIAS: Basic Image AlgorithmS Library," http://www.mip.informatik.uni-kiel.de/BIAS, 2008.

[106] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization.* John Wiley and Sons, 2001.

[107] C. Harris, "Tracking with rigid models," in *Active vision.* MIT Press, 1993, pp. 59–73.

[108] H. Kollnig and H.-H. Nagel, "3D pose estimation by directly matching polyhedral models to gray value gradients," *International Journal of Computer Vision*, vol. 23, no. 3, pp. 283–302, 1997.

[109] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2009.

[110] Z. Ni, S.-H. Ieng, C. Posch, S. Régnier, and R. Benosman, "Visual tracking using neuromorphic asynchronous event-based cameras," *Neural Computation*, 2015.

[111] H. S. M. Coxeter, *Introduction to geometry*. New York, London, 1961.

[112] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University press, 2003.

[113] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[114] C.-P. Lu, G. Hager, and E. Mjolsness, "Fast and globally convergent pose estimation from video images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 610–622, Jun. 2000.

[115] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon mesh processing*. CRC press, 2010.

[116] J. O'Rourke, *Computational geometry in C*. Cambridge university press, 1998.

[117] G. Orchard, J. G. Martin, R. J. Vogelstein, and R. Etienne-Cummings, "Fast neuromimetic object recognition using fpga outperforms gpu implementations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 8, pp. 1239–1252, 2013.

[118] G. Glaeser, "Hidden-line removal," in *Fast Algorithms for 3D-Graphics*. Springer, 1994, pp. 185–200.

[119] D. Q. Huynh, "Metrics for 3D rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009.

[120] K. Shoemake, "Animating rotation with quaternion curves," in *ACM SIGGRAPH computer graphics*, vol. 19, no. 3. ACM, 1985, pp. 245–254.

[121] V. Lepetit and P. Fua, "Keypoint recognition using randomized trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 9, pp. 1465–1479, 2006.

[122] M. M. Campos and L. de Souza Coelho, "Autonomous dirigible navigation using visual tracking and pose estimation," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, vol. 4. IEEE, 1999, pp. 2584–2589.

[123] I. Skrypnyk and D. G. Lowe, "Scene modelling, recognition and tracking with invariant image features," in *Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*. IEEE, 2004, pp. 110–119.

[124] B. M. Haralick, C.-N. Lee, K. Ottenberg, and M. Nölle, "Review and analysis of solutions of the three point perspective pose estimation problem," *International Journal of Computer Vision*, vol. 13, no. 3, pp. 331–356, 1994.

[125] G. Schweighofer and A. Pinz, "Robust pose estimation from a planar target," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2024–2030, 2006.

[126] M. Hazewinkel, *Encyclopaedia of Mathematics: Volume 6*, ser. Encyclopaedia of Mathematics. Springer Netherlands, 2013.

# Appendices

# Appendix A

# Event-based Weighted Mean

Let $z$ denote some generic variable, whose value is accessible in an asynchronous event-based fashion. Thus, we denote $z_k$ the value of $z$ at time $t_k$. Let us clarify that the variable $z_k$ can represent a number of different quantities: for example, we could be considering the mean position of some cluster, and $z_k$ would represent the $x$ or $y$ position of the events assigned to this cluster [67]. In a more complex example, $z_k$ could denote the value of some error function which we intend to minimize, computed for every incoming event [86].

Typically, one event carries little information, and we usually want to integrate the information from some events in the recent past. Then, if we are measuring some event-based variable $z_k$, the most common method is to consider its weighted mean $\overline{z}_k$ over the past events, computed as:

$$\overline{z}_k = \frac{1}{\Omega_k} \sum_{j=0}^{k} \omega_{k,j} z_{k-j}, \tag{A.1}$$

where $z_{k-j}$ is the value of the variable measured $j$ steps before the current one, with $j = 0, 1, ..., k$. The weight of the corresponding value is then denoted $\omega_{k,j}$, verifying:

$$\omega_{k,j} \geq 0, \quad \forall j = 0, 1, ..., k, \tag{A.2}$$

$$\Omega_k = \sum_{j=0}^{k} \omega_{k,j}. \tag{A.3}$$

We will next discuss the weighting strategy most commonly applied in this thesis, that we denote the *standard weighting strategy*.

## A.1 Standard Weighting Strategy

Let us define the following set of weights, that we denote the *standard weighting strategy*:

$$\omega_{k,j} = \omega_j = \omega_0 (1 - \omega_0)^j, \tag{A.4}$$

which is a geometric progression [126] with $(1 - \omega_0)$ the *common ratio* and $\omega_0$ the *scale factor*. Imposing $\omega_0 < 1$, we obtain a set of weights that decay with $j$ (i.e. with the number of events into the past), verifying:

$$\lim_{j \to \infty} \omega_{k,j} = 0 \tag{A.5}$$

$$\lim_{k \to \infty} \sum_{j=0}^{k} \omega_{k,j} = \lim_{k \to \infty} \Omega_k = 1. \tag{A.6}$$

As an example, let us display in Fig. A.1 the set of weights obtained with $\omega_0 = 0.1$. As we can see, they asymptotically decay towards 0 as $j$ grows (let us note that this strategy does not take into account the time elapsed between events in order to assign them some weight).

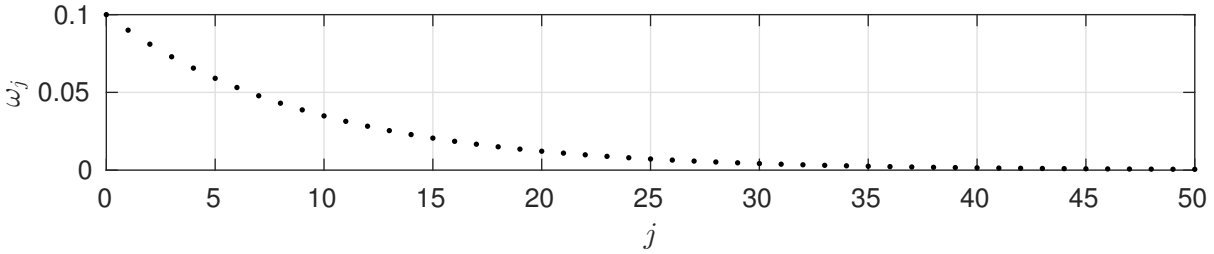



Figure A.1: Evolution of the standard set of weights with $j$ (i.e. the number of events into the past) for $\omega_0 = 0.1$. We obtain a set of weights that asymptotically decay towards 0.

Next, if we introduce this set of weights into (A.1), $\overline{z}_k$ becomes:

$$\overline{z}_k = \frac{1}{\Omega_k}\Big(\omega_0 z_k + \sum_{j=1}^{k} \omega_j z_{k-j}\Big) = \frac{1}{\Omega_k}\Big(\omega_0 z_k + \sum_{j=0}^{k-1} \omega_{j+1} z_{k-j+1}\Big). \tag{A.7}$$

Let us also take into account that this weighting strategy verifies:

$$\omega_{j+1} = \omega_0(1-\omega_0)^{j+1} = (1-\omega_0)\omega_j, \tag{A.8}$$

which, introduced into (A.7) yields:

$$\overline{z}_k = \frac{1}{\Omega_k}\Big(\omega_0 z_k + (1-\omega_0)\sum_{j=0}^{k-1} \omega_j z_{(k-1)-j}\Big) = \frac{1}{\Omega_k}\Big(\omega_0 z_k + (1-\omega_0)\overline{z}_{k-1}\Big). \tag{A.9}$$

The value of $\Omega_k$ can be exactly computed as the sum of the first $k+1$ terms of a geometric series, yielding $\Omega_k = 1 - (1-\omega_0)^{k+1}$. However, the most common solution is to take into account (A.6) and assume $\Omega_k \approx 1$. This is usually a reasonable hypothesis, due to the great number of events that are typically available. This yields the following iterative expression for $\overline{z}_k$:

$$\overline{z}_k = \omega_0 z_k + (1-\omega_0)\overline{z}_{k-1}, \tag{A.10}$$

which is the standard expression for computing the event-based weighted mean of some event-based variable $z_k$.

**Remark:** let us note that this expression is also valid if $z_k$ represents a matrix variable, because of the properties of the scalar multiplication of matrices. Taking into account the compatibility of the product of scalars with scalar multiplication, the same development can be carried out to obtain A.10.

# Appendix B

# Solution to the damped harmonic oscillator equation

Equation (2.12) is a very typical equation in classical mechanics, known as the differential equation of a *damped harmonic oscillator*. It is a second-order differential equation, and it is usually rewritten as:

$$\frac{d^2\Delta l}{dt^2} + 2\zeta w_0 \frac{d\Delta l}{dt} + w_0^2 = 0, \tag{B.1}$$

where $w_0$ is the undamped angular frequency of the oscillator and $\zeta$ is a constant parameter known as the *damping ratio* of the system, given by:

$$\zeta = \frac{B}{2\sqrt{mC}}. \tag{B.2}$$

The damping ratio $\zeta$ is a dimensionless quantity that will determine the behavior of the system. According to its value, the system can be:

- Overdamped ($\zeta > 1$): the system returns to the equilibrium state without oscillating.
- Critically damped ($\zeta = 1$): the system returns to the equilibrium state as quickly as possible without oscillating.
- Underdamped ($\zeta < 1$): the system oscillates.

As we do not wish our system to oscillate, we can impose it to be always overdamped. In that case, the solution to (B.1) is given by:

$$\Delta l(t) = A^{(1)} \exp\left(\lambda^{(1)} t\right) + A^{(2)} \exp\left(\lambda^{(2)} t\right), \tag{B.3}$$

where $A^{(1)}$ and $A^{(2)}$ depend on the initial state of the system, while $\lambda^{(1)}$, $\lambda^{(2)}$ are real negative coefficients given by:

$$\begin{aligned} \lambda^{(1)} &= \frac{-B - \sqrt{B^2 - 4Cm}}{2m}, \\ \lambda^{(2)} &= \frac{-B + \sqrt{B^2 - 4Cm}}{2m}. \end{aligned} \tag{B.4}$$

The elongation of the system is then given by the addition of two declining exponential functions with different time constants $\frac{1}{\lambda^{(1)}}$ and $\frac{1}{\lambda^{(2)}}$. The first of them is much smaller and corresponds to the rapid cancellation of the effect of the initial speed. The second one is bigger and describes the slow decay towards the equilibrium position.

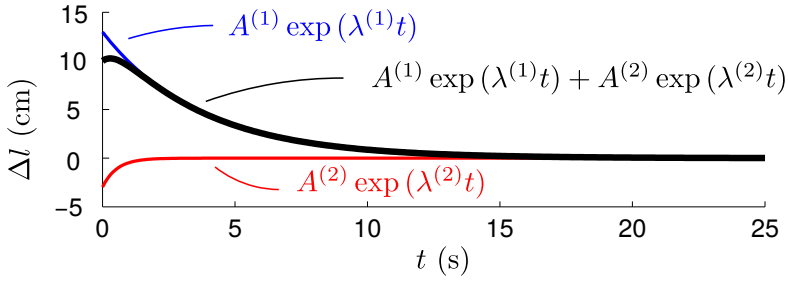



Figure B.1: Different dynamics of the two declining exponential functions for a harmonic oscillator with $C = 10$ N/cm, $m = 20$ kg, $B = 42$ Ns/cm, and initial conditions $\Delta l(t = 0) = 10$ cm and $\frac{d\Delta l}{dt}|_{t=0} = 2$ cm/s.

Fig. B.1 shows the different dynamics of the two declining exponential functions for an harmonic oscillator with some typical values: $C = 10$ N/cm, $m = 20$ kg, $B = 42$ Ns/cm, and initial conditions $\Delta l(t = 0) = 10$ cm and $\frac{d\Delta l}{dt}|_{t=0} = 2$ cm/s. As we can see, the first of them is much faster than the other one and it rapidly tends to zero. This allows us to approximate the position $\Delta l(t)$ of the system just by the second exponential:

$$\Delta l(t) \approx A^{(2)} \exp(\lambda^{(2)} t), \tag{B.5}$$

from where we can isolate t, that takes the value:

$$t = \frac{1}{\lambda^{(2)}} \log\left(\frac{\Delta l}{A^{(2)}}\right). \tag{B.6}$$

Deriving (B.5) we obtain an approximation for the speed:

$$\frac{d(\Delta l)}{dt}\Big|_t \approx \lambda^{(2)} A^{(2)} \exp(\lambda^{(2)} t). \tag{B.7}$$

Next, combining (B.7) and (B.6) we obtain the speed as a function of the position:

$$\frac{d(\Delta l)}{dt} \approx \lambda^{(2)} A^{(2)} \exp\left(\lambda^{(2)} \frac{1}{\lambda^{(2)}} \log\left(\frac{\Delta l}{A^{(2)}}\right)\right) = \lambda^{(2)} \Delta l. \tag{B.8}$$

If we now assume the speed to be constant during a certain period of time $\Delta t$, we can approximate it by the expression:

$$\frac{d(\Delta l)}{dt} \approx \frac{\Delta(\Delta l)}{\Delta t} \approx \lambda^{(2)} \Delta l, \tag{B.9}$$

where $\Delta(\Delta l)$ is the displacement experienced by the system during the time interval $\Delta t$. Isolating the displacement $\Delta(\Delta l)$ in (B.10) we obtain its approximate value, given by:

$$\Delta(\Delta l) \approx \lambda^{(2)} \Delta t \Delta l = \frac{-B + \sqrt{B^2 - 4Cm}}{2m} \Delta t \Delta l = \frac{\alpha}{m} \Delta l, \tag{B.10}$$

where

$$\alpha = \frac{-B + \sqrt{B^2 - 4Cm}}{2}\Delta t \tag{B.11}$$

would be the stiffness parameter of our spring-like connections.

As we can see in (B.11), $\alpha$ actually depends on the mass of the object for which we are solving the differential equation, and not only on the characteristics of the harmonic oscillator. According to this, if we were simulating a real mechanical system we could not assign a value of $\alpha$ to each connection, independently of the value of the masses. However, for simplicity, this is what we do in our method (see (2.15), (2.19) and (2.21)). Thus, the stiffness $\alpha$ and the masses of the trackers become nothing but a dimensionless scaling factor.

# Appendix C

# Mathematical development yielding the optimal $\rho$, $\theta$

Let us remind the reader that the error we are trying to minimize is given by (3.9):

$$E_k^{(i)}(\theta,\rho) = \frac{1}{\Omega_k}\sum_{j=0}^{k}\omega_{k,j}\big(x_{k-j}\cos(\theta)+y_{k-j}\sin(\theta)-\rho\big)^2. \tag{C.1}$$

Then, making $\frac{\partial E_k}{\partial \rho}=0$ we obtain:

$$\frac{2}{\Omega_k}\sum_{j=0}^{k}\omega_{k,j}\big(x_{k-j}\cos(\theta)+y_{k-j}\sin(\theta)-\rho\big)(-1)=0, \tag{C.2}$$

which, after simplification, yields the necessary condition for $\rho$ to produce a minimum (or maximum) of the error function:

$$\rho = \cos(\theta)\sum_{j=0}^{k}\omega_{k,j}x_{k-j}+\sin(\theta)\sum_{j=0}^{k}\omega_{k,j}y_{k-j}=0. \tag{C.3}$$

Introducing the value of the auxiliary parameters in (C.3) we obtain (3.10):

$$\rho_k^{(i)} = \frac{\widehat{x}_k^{(i)}\cos(\theta)+\widehat{y}_k^{(i)}\sin(\theta)}{\Omega_k}, \tag{C.4}$$

From $\frac{\partial E_k}{\partial \theta}=0$ we obtain:

$$\frac{2}{\Omega_k}\sum_{j=0}^{k}\omega_{k,j}\big(x_{k-j}\cos(\theta)+y_{k-j}\sin(\theta)-\rho\big)\big(-x_{k-j}\sin(\theta)+y_{k-j}\cos(\theta)\big)=0, \tag{C.5}$$

Simplifying and developing this equation we obtain the following:

$$\begin{aligned}0=\sum_{j=0}^{k}\omega_{k,j}\Big(&-x_{k-j}^2\cos(\theta)\sin(\theta)+x_{k-j}y_{k-j}\cos^2(\theta)\\&-x_{k-j}y_{k-j}\sin^2(\theta)-y_{k-j}^2\sin(\theta)\cos(\theta)\\&+\rho(x_{k-j}\sin(\theta)-y_{k-j}\cos(\theta))\Big)\end{aligned} \tag{C.6}$$

Introducing (3.10) and the value of the auxiliary parameters into the previous equation, and applying some trigonometry we get:

$$\begin{aligned} 0 &= \frac{1}{2}\sin(2\theta)(\widehat{yy}_k^{(i)} - \widehat{xx}_k^{(i)}) + \cos(2\theta)\widehat{xy}_k^{(i)} \\ &\quad + \frac{\widehat{x}_k^{(i)}\cos(\theta) + \widehat{y}_k^{(i)}\sin(\theta)}{\Omega_k}\left(\widehat{x}_k^{(i)}\sin(\theta) - \widehat{y}_k^{(i)}\cos(\theta)\right) \\ &= \frac{1}{2}\sin(2\theta)\left(\widehat{yy}_k^{(i)} - \widehat{xx}_k^{(i)}\right) + \cos(2\theta)\widehat{xy}_k^{(i)} \\ &\quad + \frac{1}{\Omega_k}\left(\frac{1}{2}\left(\left(\widehat{x}_k^{(i)}\right)^2 - \left(\widehat{y}_k^{(i)}\right)^2\right)\sin(2\theta) - \widehat{x}_k^{(i)}\widehat{y}_k^{(i)}\cos(2\theta)\right). \end{aligned} \tag{C.7}$$

And thus we obtain:

$$\sin(2\theta)\left(\Omega_k\left(\widehat{yy}_k^{(i)} - \widehat{xx}_k^{(i)}\right) + \left(\widehat{x}_k^{(i)}\right)^2 - \left(\widehat{y}_k^{(i)}\right)^2\right) + \cos(2\theta)\left(2\Omega_k\widehat{xy}_k^{(i)} - 2\widehat{x}_k^{(i)}\widehat{y}_k^{(i)}\right) = 0 \tag{C.8}$$

From which we directly obtain (3.11).

# Appendix D

# Optimization strategy for the computation of the line parameters

Let us rewrite (3.11) dropping the superindices, in order to lighten the notation:

$$a_k \sin(2\theta_k) + b_k \cos(2\theta_k) = 0. \tag{D.1}$$

Taking into account the fundamental trigonometric identity we obtain:

$$\pm a_k \sqrt{1 - \cos^2(2\theta_k)} = -b_k \cos(2\theta_k),$$

which yields:

$$a_k^2\big(1 - \cos^2(2\theta_k)\big) = b_k^2 \cos^2(2\theta_k) \quad \Rightarrow \quad \cos^2(2\theta_k) = \frac{a_k^2}{a_k^2 + b_k^2}.$$

From this we get:

$$\cos(2\theta_k) = \beta_k, \tag{D.2}$$

with:

$$\beta_k = \pm\sqrt{\frac{a_k^2}{a_k^2 + b_k^2}}. \tag{D.3}$$

Next, we insert into (D.2) the following trigonometric identities: $\cos(2\theta) = 2\cos^2(\theta) - 1$, and $\cos(2\theta) = 1 - 2\sin^2(\theta)$. We obtain:

$$\cos(\theta_k) = \sqrt{\frac{\beta_k + 1}{2}}, \quad \sin(\theta_k) = \pm\sqrt{\frac{1 - \beta_k}{2}}. \tag{D.4}$$

Where we only keep the positive sign for the cosinus, because $\theta_k \in [-\pi/2,\ \pi/2]$.

# Appendix E

# Iterative expression for the auxiliary parameters

For a given line $\mathcal{L}^{(i)}$, the auxiliary parameter $\widehat{x}_k^{(i)}$ is given by the weighted sum of the $x$ coordinates of the events previously assigned to the line. When considering the speed-tuned weighting strategy given by (3.15), this auxiliary parameter takes the value:

$$\begin{aligned}\widehat{x}_k^{(i)} &= \sum_{j=0}^{k} \omega_{k,j} x_{k-j} = \omega_{k,0} x_k + \sum_{j=1}^{k} \omega_{k,j} x_{k-j} \\ &= x_k + \sum_{j=0}^{k-1} \omega_{k,j+1} x_{k-(j+1)} \\ &= x_k + e^{-\|\boldsymbol{v}_k\|\Delta t_k} \sum_{j=0}^{k-1} \omega_{k-1,j} x_{(k-1)-j} \\ &= x_k + \delta_k \widehat{x}_{k-1}^{(i)},\end{aligned} \tag{E.1}$$

where $\delta_k = e^{-\|\boldsymbol{v}_k\|\Delta t_k}$. We can repeat the same development for the rest of the auxiliary parameters, obtaining the iterative expressions given in (3.16).

Analogously, we obtain the following value for $\Omega_k$:

$$\Omega_k = \sum_{j=0}^{k} \omega_{k,j} = \omega_{k,0} + e^{-\|\boldsymbol{v}_k\|\Delta t_k} \sum_{j=0}^{k-1} \omega_{k-1,j} = 1 + \delta_k \Omega_{k-1}, \tag{E.2}$$

which is equivalent to the expression of the activity given in (3.6).

# Appendix F

# Disambiguation of cos($\theta$), sin($\theta$)

Let us define $\gamma_1$, $\gamma_2$:

$$\gamma_1 = \sqrt{\frac{1-|\beta_k|}{2}}, \quad \gamma_2 = \sqrt{\frac{1+|\beta_k|}{2}}, \tag{F.1}$$

From (3.18) we obtain four possible combinations for $\cos\theta_k$, $\sin\theta_k$ (where we drop the superindex indicating the line in order to lighten the notation):

$$\begin{cases} \text{if } \beta_k = +|\beta_k| \quad \Rightarrow \quad \sin(\theta_k) = \pm\gamma_1, \quad \cos(\theta_k) = \gamma_2 \\ \text{if } \beta_k = -|\beta_k| \quad \Rightarrow \cos(\theta_k) = \gamma_1, \quad \sin(\theta_k) = \pm\gamma_2 \end{cases} \tag{F.2}$$

Next, we need to disambiguate for the sign of the sinus. Here, we take into account that $\tan 2\theta_k = \frac{-b_k}{a_k}$. Then, as one can see in Fig. F.1, the following rule applies:

$$\begin{cases} \text{if } \quad \dfrac{-b_k}{a_k} > 0 \begin{cases} \text{if } \cos(\theta_k) < \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) < 0 \\ \text{if } \cos(\theta_k) > \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) > 0 \end{cases} \\ \text{if } \quad \dfrac{-b_k}{a_k} < 0 \begin{cases} \text{if } \cos(\theta_k) < \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) > 0 \\ \text{if } \cos(\theta_k) > \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) < 0 \end{cases} \end{cases} \tag{F.3}$$

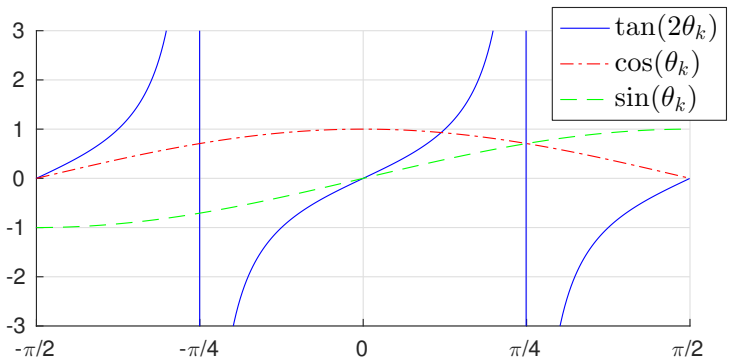



Figure F.1: The sign of the sinus can be disambiguated from the value of $\lambda$ and $\cos(\theta_k)$.

This leaves us with two possible combinations, which correspond to the two perpendicular lines yielding the maximum and the minimum error. We disambiguate between then by choosing the value of $\cos(\theta_k)$ closest to the previous one. Here, we choose to compare the cosinus because for almost horizontal lines we have $\theta_k \approx \pm\pi/2$. Values of $\theta_k$ close to $\pi/2$ or $-\pi/2$ will have a similar cosinus, allowing for the line to change from one to another.

# Appendix G

# Solutions to the system of equations of the 3D matching step

In this section, we discuss the solutions to the system of equations defined in (4.8). Let $S$ be the system matrix, that has the form:

$$S = \begin{pmatrix} -\boldsymbol{M}_k^T \boldsymbol{M}_k & \boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)} \\ -\boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)} & \left(\boldsymbol{\varepsilon}_k^{(nm)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)} \end{pmatrix}. \tag{G.1}$$

Next, we discuss the solutions to this system, both in the singular and in the general case.

## G.1 Singular case

The system matrix $S$ is singular when $\det(S) = 0$, where the determinant takes the following value:

$$\det(S) = -\left(\boldsymbol{M}_k^T \boldsymbol{M}_k\right)\left(\left(\boldsymbol{\varepsilon}_k^{(nm)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)}\right) + \left(\boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)}\right)^2. \tag{G.2}$$

Developing the dot products in this equation, we get:

$$\begin{aligned} \det(S) = 0 \quad &\Leftrightarrow \quad \|\boldsymbol{M}_k\|^2 \|\boldsymbol{\varepsilon}_k^{(nm)}\|^2 = (\|\boldsymbol{M}_k\| \|\boldsymbol{\varepsilon}_k^{(nm)}\| \cos(\gamma))^2, \\ \det(S) = 0 \quad &\Leftrightarrow \quad \cos(\gamma) = \pm 1. \end{aligned} \tag{G.3}$$

where $\gamma$ is the angle between $\boldsymbol{M}_k$ and $\boldsymbol{\varepsilon}_k^{(nm)}$.

Consequently, $S$ will be singular if $\gamma = 0$ or $\gamma = \pi$, i.e. if $\boldsymbol{M}_k$ and $\boldsymbol{\varepsilon}_k^{(nm)}$ are collinear. Fig. G.1 shows the geometry of the problem when this situation occurs. In this case, $\boldsymbol{B}_k$ is chosen between $\boldsymbol{V}_k^{(n)}$ and $\boldsymbol{V}_k^{(m)}$ by taking the one with smaller $Z$ coordinate. We then compute $\alpha$ from the perpendicularity constraint between $\boldsymbol{M}_k$ and $(\boldsymbol{A}_k - \boldsymbol{B}_k)$, getting the following result:

$$\alpha = \frac{\boldsymbol{B}_k^T \boldsymbol{M}_k}{\boldsymbol{M}_k^T \boldsymbol{M}_k}, \tag{G.4}$$

and insert this value in (5.9) to obtain $\boldsymbol{A}_k$.

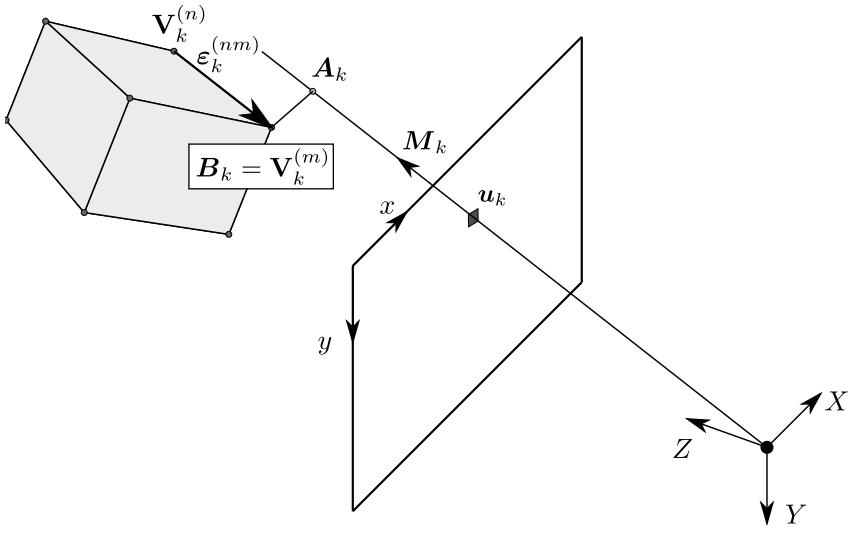



Figure G.1: If $\boldsymbol{M}_k$ and $\boldsymbol{\varepsilon}_k^{(nm)}$ are parallel, then the system matrix is singular. In this case the system does not have a unique solution, since every point in the line defined by $\boldsymbol{M}_k$ is at the same distance from $\varepsilon_k^{(nm)}$. For solving this indeterminacy, we simply choose $\boldsymbol{B}_k$ between $\boldsymbol{V}_k^{(n)}$ and $\boldsymbol{V}_k^{(m)}$, taking the one with smaller $Z$ coordinate.

## G.2 General case

In the general case, $S$ will be invertible. Since $S$ is a $2 \times 2$ matrix, we can analytically precompute its inverse, saving computational power. In order to solve the system, we define the following dot products

$$\begin{aligned} a &= \boldsymbol{M}_k^T \boldsymbol{M}_k, \\ b &= \left(\boldsymbol{\varepsilon}_k^{(nm)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)}, \\ c &= \boldsymbol{M}_k^T \boldsymbol{\varepsilon}_k^{(nm)}, \\ d &= \left(\boldsymbol{V}_k^{(n)}\right)^T \boldsymbol{M}_k, \\ e &= \left(\boldsymbol{V}_k^{(n)}\right)^T \boldsymbol{\varepsilon}_k^{(nm)}. \end{aligned} \tag{G.5}$$

Thus, the inverse will have the following expression:

$$S^{-1} = \frac{1}{\det(S)} \begin{pmatrix} b & -c \\ c & -a \end{pmatrix}, \tag{G.6}$$

where $\det(S) = -ab + c^2$. This allows us to solve the system for $\alpha$ and $\beta$. As a final observation, we need to take into account that $\varepsilon_k^{(nm)}$ is a segment, which means that $\beta$ has to be contained in the interval $[0, 1]$. Thus, the final values for $\alpha$ and $\beta$ are:

$$\alpha = \frac{-bd + ce}{\det(S)}, \tag{G.7}$$

and

$$\beta = \begin{cases} 0, & \text{if } \dfrac{-cd+ae}{\det(S)} \leq 0, \\ 1, & \text{if } \dfrac{-cd+ae}{\det(S)} \geq 1, \\ \dfrac{-cd+ae}{\det(S)}, & \text{otherwise.} \end{cases} \tag{G.8}$$

Inserting these values in (5.9) and (5.10) yields $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$.

# Appendix H

# Mathematical development yielding the optimal translation

Let us remind the reader that the object-space collinearity error for a given event $\mathbf{e}_k$ and some estimated pose $\widehat{R}, \widehat{\boldsymbol{T}}$ is given by:

$$\boldsymbol{\xi}_k(\widehat{R}, \widehat{\boldsymbol{T}}) = \boldsymbol{Q}_k - \widehat{\mathbf{V}}^{(i_k)} = (L_k - I)(\widehat{R}\mathbf{V}^{(i_k)} + \widehat{\boldsymbol{T}}). \tag{H.1}$$

As shown in [114], the *Line-of-Sight* projection matrix $L_k$ verifies the following properties:

$$\|\boldsymbol{x}\| \geq \|L_k\boldsymbol{x}\|, \quad \forall \boldsymbol{x} \in \mathbb{R}^3, \tag{H.2a}$$

$$L_k^T = L_k, \tag{H.2b}$$

$$L_k^2 = L_k L_k^T = L_k. \tag{H.2c}$$

The weighted mean of the squared collinearity errors is then:

$$\begin{aligned}
E_k(\widehat{R}, \widehat{\boldsymbol{T}}) &= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\|(L_{k-j} - I)(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}})\|^2 \\
&= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big((L_{k-j} - I)(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}})\Big)^T\Big((L_{k-j} - I)(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}})\Big) \\
&= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}})^T(L_{k-j} - I)^T(L_{k-j} - I)(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}}) \\
&= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}})^T(I - L_{k-j})(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}}).
\end{aligned}$$

Next, by abuse of notation, let us note:

$$I - L_{k-j} = H = \begin{bmatrix}\boldsymbol{h}^{(1)} \\ \boldsymbol{h}^{(2)} \\ \boldsymbol{h}^{(3)}\end{bmatrix} = \begin{bmatrix}\boldsymbol{h}_{(1)} & \boldsymbol{h}_{(2)} & \boldsymbol{h}_{(3)}\end{bmatrix} = \begin{bmatrix}h_{(1,1)} & h_{(1,2)} & h_{(1,3)} \\ h_{(2,1)} & h_{(2,2)} & h_{(2,3)} \\ h_{(3,1)} & h_{(3,2)} & h_{(3,3)}\end{bmatrix}. \tag{H.3}$$

Analogously, let us note $\widehat{R}\mathbf{V}^{(i_{k-j})} = \boldsymbol{P} = [P_x,\ P_y,\ P_z]^T$, and $\widehat{\boldsymbol{T}} = [\widehat{T}_x,\ \widehat{T}_y,\ \widehat{T}_z]^T$.

This yields:

$$
\begin{aligned}
E_k(\widehat{R},\widehat{\boldsymbol{T}}) &= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big[P_x+\widehat{T}_x,\ P_y+\widehat{T}_y,\ P_z+\widehat{T}_z\Big]\begin{bmatrix}\boldsymbol{h}^{(1)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\\ \boldsymbol{h}^{(2)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\\ \boldsymbol{h}^{(3)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\end{bmatrix}\\
&= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(\big(P_x+\widehat{T}_x\big)\Big(h_{(1,1)}(P_x+\widehat{T}_x)+h_{(1,2)}(P_y+\widehat{T}_y)+h_{(1,3)}(P_z+\widehat{T}_z)\Big)\\
&\qquad +\big(P_y+\widehat{T}_y\big)\Big(h_{(2,1)}(P_x+\widehat{T}_x)+h_{(2,2)}(P_y+\widehat{T}_y)+h_{(2,3)}(P_z+\widehat{T}_z)\Big)\\
&\qquad +\big(P_z+\widehat{T}_z\big)\Big(h_{(3,1)}(P_x+\widehat{T}_x)+h_{(3,2)}(P_y+\widehat{T}_y)+h_{(3,3)}(P_z+\widehat{T}_z)\Big)\Big)
\end{aligned}
$$

Next, if we compute the partial derivatives of the error with respect to the components of $\widehat{\boldsymbol{T}}$ we obtain:

$$
\begin{aligned}
\frac{\partial E_k(\widehat{R},\widehat{\boldsymbol{T}})}{\partial \widehat{T}_x} &= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(h_{(1,1)}(P_x+\widehat{T}_x)+h_{(1,2)}(P_y+\widehat{T}_y)+h_{(1,3)}(P_z+\widehat{T}_z)\\
&\qquad +h_{(1,1)}(P_x+\widehat{T}_x)+h_{(2,1)}(P_y+\widehat{T}_y)+h_{(3,1)}(P_z+\widehat{T}_z)\Big) \qquad \text{(H.4)}\\
&= \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(\boldsymbol{h}^{(1)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})+\boldsymbol{h}^{T}_{(1)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\Big).
\end{aligned}
$$

And, analogously:

$$
\frac{\partial E_k(\widehat{R},\widehat{\boldsymbol{T}})}{\partial \widehat{T}_y} = \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(\boldsymbol{h}^{(2)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})+\boldsymbol{h}^{T}_{(2)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\Big), \qquad \text{(H.5)}
$$

$$
\frac{\partial E_k(\widehat{R},\widehat{\boldsymbol{T}})}{\partial \widehat{T}_z} = \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(\boldsymbol{h}^{(3)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})+\boldsymbol{h}^{T}_{(3)}(\boldsymbol{P}+\widehat{\boldsymbol{T}})\Big). \qquad \text{(H.6)}
$$

Making the three partial derivatives equal to zero, we obtain the necessary condition for a translation vector to be a minimum of the error function:

$$
\frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\begin{bmatrix}\boldsymbol{h}^{(1)}+\boldsymbol{h}^{T}_{(1)}\\ \boldsymbol{h}^{(2)}+\boldsymbol{h}^{T}_{(2)}\\ \boldsymbol{h}^{(3)}+\boldsymbol{h}^{T}_{(3)}\end{bmatrix}(\boldsymbol{P}+\widehat{\boldsymbol{T}}) = \frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}(H+H^T)(\boldsymbol{P}+\widehat{\boldsymbol{T}}) = \begin{bmatrix}0\\0\\0\end{bmatrix} \qquad \text{(H.7)}
$$

Which is equivalent to:

$$
\begin{aligned}
&\frac{1}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\Big(I-L_{k-j}+(I-L_{k-j})^T\Big)\Big(\widehat{R}\mathbf{V}^{(i_{k-j})}+\widehat{\boldsymbol{T}}\Big)\\
&= \frac{2}{\Omega_k}\sum_{j=0}^{n-1}\omega_{k,j}\big(I-L_{k-j}\big)\big(\widehat{R}\mathbf{V}^{(i_{k-j})}+\widehat{\boldsymbol{T}}\big)=\mathbf{0}.
\end{aligned} \qquad \text{(H.8)}
$$

Thus, the resulting equation:

$$\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})(\widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}}) = \mathbf{0} \tag{H.9}$$

is the necessary condition for a given translation to yield a minimum (or maximum) error.

From (H.9) we can compute the optimal translation at time $t_k$ for a given estimated pose $\widehat{R}, \widehat{\boldsymbol{T}}$, that we will denote $\boldsymbol{T}_k^*(\widehat{R}, \widehat{\boldsymbol{T}})$:

$$\boldsymbol{T}_k^*(\widehat{R}, \widehat{\boldsymbol{T}}) = \Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\widehat{R}\mathbf{V}^{(i_{k-j})}. \tag{H.10}$$

Next, taking into account that $\widehat{\mathbf{V}}^{(i_{k-j})} = \widehat{R}\mathbf{V}^{(i_{k-j})} + \widehat{\boldsymbol{T}}$, it follows that:

$$\begin{aligned}
\boldsymbol{T}_k^*(\widehat{R}, \widehat{\boldsymbol{T}}) =& \Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)(\widehat{\mathbf{V}}^{(i_{k-j})} - \widehat{\boldsymbol{T}}) \\
=& \Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}^{(i_{k-j})} \\
&- \Big(\Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\Big)\widehat{\boldsymbol{T}} \\
=& \Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}^{(i_{k-j}))} + \widehat{\boldsymbol{T}}
\end{aligned} \tag{H.11}$$

Then, if the previous estimation of the pose was given by $\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}$, from (H.11) we can compute the optimal displacement $\Delta\boldsymbol{T}_k(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1})$:

$$\begin{aligned}
\Delta\boldsymbol{T}_k(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}) &= \boldsymbol{T}^*(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}) - \widehat{\boldsymbol{T}}_{k-1} \\
&= \Big(\sum_{j=0}^{n-1} \omega_{k,j}(I - L_{k-j})\Big)^{-1} \sum_{j=0}^{n-1} \omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}_{k-1}^{(i_{k-j})},
\end{aligned} \tag{H.12}$$

which allows us to make $\Delta\boldsymbol{T}_k(\widehat{R}_{k-1}, \widehat{\boldsymbol{T}}_{k-1}) = A_k^{-1}\boldsymbol{B}_k$, with $A_k$, $B_k$ given by (5.9) and (5.10) respectively.

# Appendix I

# Iterative update

Let us consider the standard set of weights defined in the Appendix A.1:

$$\omega_{k,j} = \omega_j = \omega_0(1-\omega_0)^j \tag{I.1}$$

with $\omega_0 < 1$.

Since the weights decay towards zero with $j$, we can approximate (5.9) by taking into account every past event $\mathbf{e}_{k-j}$, with $j = 0, 1, \dots, k$:

$$A_k = \sum_{j=0}^{n-1} \omega_j (I_3 - L_{k-j}) \approx \sum_{j=0}^{k} \omega_j (I_3 - L_{k-j}). \tag{I.2}$$

Which is the usual expression of the event-based weighted mean. According to this, and since we are considering the standard weighting strategy we can make:

$$A_k \approx \omega_0(I_3 - L_k) + (1-\omega_0)A_{k-1}. \tag{I.3}$$

In the case of $\boldsymbol{B}_k$, (5.10) becomes:

$$\boldsymbol{B}_k \approx \sum_{j=0}^{k} \omega_{k,j}(L_{k-j} - I)\widehat{\mathbf{V}}^{(i_{k-j})}. \tag{I.4}$$

Here, $\widehat{\mathbf{V}}^{(i_{k-j})}$ is the current estimated pose of the corresponding point of the object, whose value is dependent on $\widehat{R}_k$, $\widehat{\boldsymbol{T}}_k$, which change with every incoming event. Consequently, we cannot proceed as in the case of $A_k$ in order to find an iterative expression for $B_k$. Nevertheless, and thanks to the high temporal resolution of the neuromorphic camera, we can assume that the pose changes slightly with each event. Since weights defined in (I.1) decay towards zero with $j$, we can assume that we are only taking into account some recent events for which the estimated pose is approximately constant. Under this assumptions we can approximate the value of $B_k$ in an analogous manner, making:

$$\boldsymbol{B}_k \approx \omega_0(L_k - I)\widehat{\mathbf{V}}^{(i_{k-j})} + (1-\omega_0)B_{k-1}. \tag{I.5}$$

# Appendix J

# Justification of the rotation

When we apply a resulting torque $\boldsymbol{\Gamma}$ to a body, the Newton's Second Law for rotation states that [69]:

$$\boldsymbol{\Gamma} = \frac{d\boldsymbol{H}}{dt}, \tag{J.1}$$

where $\boldsymbol{H}$ is the angular momentum of the body that takes the value:

$$\boldsymbol{H} = J\boldsymbol{w}. \tag{J.2}$$

Here, $\boldsymbol{w}$ is the angular velocity and $J$ is a $3 \times 3$ symmetric matrix known as the inertia tensor. The form of $J$ depends on how the mass of a body is distributed, and it gives an idea of how hard it is to accelerate the object around each one of the axis. However, since we are not modeling the behavior of a real mechanical system, we can imagine the mass of our virtual system to be equally distributed in all directions. This is equivalent to imagining the object to be embedded in a uniform sphere. The inertia tensor of a sphere has the form:

$$J = \alpha I, \tag{J.3}$$

where $\alpha$ is a real value, equal in the case of the sphere to $2ml^2/5$, $l$ being the radius of the sphere and $m$ its mass.

After these considerations (J.1) becomes:

$$\boldsymbol{\Gamma} = \alpha\frac{d\boldsymbol{w}}{dt}, \tag{J.4}$$

which is equivalent to the Newton's Second Law for translational motion. If we integrate this equation for a small period of time $\Delta t$, during which we suppose $\boldsymbol{\Gamma}$ to remain constant, and assuming zero initial angular velocity, we obtain the following rotation $\boldsymbol{r}$:

$$\boldsymbol{r} = \frac{1}{2}\alpha^{-1}\boldsymbol{\Gamma}\Delta t^2 = \phi\boldsymbol{\Gamma}, \tag{J.5}$$

where we make $\phi = \dfrac{\Delta t^2}{2\alpha}$.

# Appendix K

# Maximum torque and optimal $\phi$

Developing (5.23) we get the following expression for the torque $\boldsymbol{\tau}_{k-j}$ associated with event $\mathbf{e}_{k-j}$:

$$\boldsymbol{\tau}_{k-j}(\widehat{R},\widehat{\boldsymbol{T}}) = \beta\omega_{k,j}\Big(\widehat{R}\boldsymbol{V}^{(i_{k-j})} \times L_{k-j}\widehat{R}\boldsymbol{V}^{(i_{k-j})} + \widehat{R}\boldsymbol{V}^{(i_{k-j})} \times (L_{k-j} - I)\widehat{\boldsymbol{T}}\Big), \tag{K.1}$$

which is expressed as the addition of two vectors. The first one of them is equal to the cross product of a vector with its own projection. Taking into account that $\|R\boldsymbol{V}\| = \|\boldsymbol{V}\|$, its norm is thus bounded by the expression:

$$\|\widehat{R}\boldsymbol{V}^{(i_{k-j})} \times L_{k-j}\widehat{R}\boldsymbol{V}^{(i_{k-j})}\| \leq \frac{\|\boldsymbol{V}^{(i_{k-j})}\|^2}{2}, \tag{K.2}$$

where the maximum value is attained when $\widehat{R}\boldsymbol{V}^{(i_{k-j})}$ and the line of sight form an angle of $\pi/4$ radians.

Next, $\|(L_{k-j} - I)\widehat{\boldsymbol{T}}\|$ is equal to the distance between $\widehat{\boldsymbol{T}}$ and the corresponding line of sight. If we assume that the estimated translation is close to its true value $\widehat{\boldsymbol{T}} \approx \boldsymbol{T}$, then it follows that:

$$\|(L_{k-j} - I)\widehat{\boldsymbol{T}}\| \approx \|(L_{k-j} - I)\boldsymbol{T}\| \leq \|\boldsymbol{V}^{(i_{k-j})}\|, \tag{K.3}$$

which takes its maximum value when $R\boldsymbol{V}^{(i_{k-j})}$ is perpendicular to the line of sight.

The maximum torque will therefore be produced when both vectors in (K.1) take their maximum value, that is to say when $R\boldsymbol{V}^{(i_{k-j})}$ is perpendicular to the line of sight and $\widehat{R}\boldsymbol{V}^{(i_{k-j})}$ forms an angle of $\pi/4$ radians with the line of sight. Fig. K.1 shows the state of the system in this case. From the geometry it follows that:

$$\|\boldsymbol{\tau}_{k-j}\| \leq \frac{1+\sqrt{2}}{2}\beta\omega_{k,j}\|\boldsymbol{V}^{(i_{k-j})}\|^2. \tag{K.4}$$

If we call $\rho^{(max)}$ to the maximum norm $\rho^{(max)} = \max_i\{\|\boldsymbol{V}^{(i)}\|\}$, then the resulting torque is bounded by the expression:

$$\|\boldsymbol{\Gamma}_k\| \leq \frac{1+\sqrt{2}}{2}\beta\Omega_k(\rho^{(max)})^2, \tag{K.5}$$

which depends on the dimensions of the considered object.

From the geometry, it follows that the angle formed by $\widehat{R}\boldsymbol{V}^{(i_{k-j})}$ and $R\boldsymbol{V}^{(i_{k-j})}$ is in this case $3\pi/4$ radians. We will accept the optimal value of $\phi$, that we denote $\phi^{(opt)}$, to

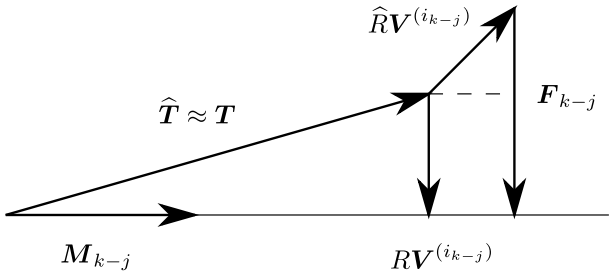



Figure K.1: State of the system when the maximum torque is produced: $\widehat{R}\boldsymbol{V}^{(i_{k-j})}$ forms an angle of $\pi/4$ radians with the line of sight, causing $\|\widehat{R}\boldsymbol{V}^{(i_{k-j})} \times L_k \widehat{R}\boldsymbol{V}^{(i_{k-j})}\|$ to be maximum. At the same time, $R\boldsymbol{V}^{(i_{k-j})}$ is perpendicular to this same line of sight, making the distance between $\boldsymbol{T}$ and the line of sight maximum: $\|(L_{k-j} - I)\boldsymbol{T}\| = \|\boldsymbol{V}^{(i_{k-j})}\|$.

be the one that causes the angle of the rotation applied in this case to be equal to $3\pi/4$ radians. $\lambda_r^{opt}$ is thus given by the expression:

$$\phi^{(opt)} = \frac{3\pi}{2(1+\sqrt{2})} \frac{1}{\beta \Omega_k (\rho^{(max)})^2}. \tag{K.6}$$

Let us note that the maximum value of the torque is very unlikely to be produced. Even if the geometry matches the right one, different points of the object are likely to produce events, usually yielding lower values of the torque. Consequently, $\phi^{(opt)}$ is a conservative value, and we can have stable systems with $\phi$ greater than $\phi^{(opt)}$. However, we consider this expression to be an useful tool to determine the order of magnitude for this parameter.

# List of Figures

# List of Tables